

****

**编译原理课程设计申优文档**

|  |  |
| --- | --- |
| 院（系）名称 | 计算机学院 |
| 专业名称 | 计算机科学与技术 |
| 学号 | 16061170 |
| 姓名 | 宋卓洋 |

2019年1月

## 代数性质优化

通常情况下在进行语义分析的时候，对于常量的处理一般是采用和全局变量一样的处理手段，将其存至内存中的某一区域。由于最终的目标语言是支持对于立即数的计算操作的，显然在这一阶段完全可以将常量进行内联，直接加入至中间代码中。如果不采取内联的方式处理常量，在目标代码的对于常量的计算或多或少的会需要对内存进行操作，带来不必要的开销。如果将常量内联至中间代码，可以更加直观地进行对于常数的优化，如进行常量的传播。此外对于可以直接计算出结果或直接产生比较结果的操作，编译器完全能够在编译期间完成这些操作，直接将结果编入中间代码中，简化中间代码。此外，由于无需再维护程序中的常量，这对于目标程序的运行栈结构进行了一定的简化。

## 运行栈改进

在我最初的对运行时环境的设计中，对于每个栈帧的定位是靠$fp和$sp两个寄存器共同完成的。使用$fp寄存器记录每个栈帧的起始地址，使用$sp寄存器记录栈顶地址。每次压栈退栈操作时需要进行一次内存的读写操作。但如果出现了多次调用函数的情况，这带来的开销是巨大的。事实上$sp寄存器在这种运行栈的设计中并未发挥作用，仅仅是作为压栈的“跳板”。而仅通过一个寄存器也能完成正常的压栈退栈操作。为减少函数调用带来的运行栈维护的开销，我改为只用单个寄存器记录栈帧开始地址，通过相对偏移来进行寻址和压退栈操作。这使得每次函数调用与返回时节省了两个计算指令和两个访存指令。

## DAG图设计

在实际情况中，对于DAG图的设计与理论中情况有着较大的差别。这些差别主要表现在显示中的代码种类多样，在抽象为DAG图的结点时存在着一定的困难。就课设中的文法而言，由于在划分基本块时未将函数调用语句作为基本块划分的标准，基本块中的数组赋值语句、读写操作、函数调用语句加入到DAG图中的难度较大。

首先是数组的赋值语句。由于我并未对数组进行寄存器的分配，每次数组赋值操作都必定将新值写回之内存中相应的位置。如果后续代码需要使用这一数组的变量就需要使用数组取值语句将其从内存中取出，产生相应的DAG图结点。因此数组复制语句无需加入至DAG图中。

其次是读操作语句。读操作语句会对一个变量进行赋值操作。按照DAG图的定义，需要产生一个新的结点保存该变量。但是如果直接采取这个操作，就会导致DAG图中会存在一各孤立的单个节点。此外如果后续代码没由使用这一变量，也不需要产生这个节点。因此，对于读操作语句我采取了延迟操作策略，即在运行读语句时先只是将相应变量从结点表中删除，若后续语句使用了该变量再按照DAG图导入策略生成相应的结点。

对于写语句，由于写语句需要立刻使用写语句变量中的值，进而需要在执行该语句前进行一次DAG图的导出。生成与写语句需要的变量相关的计算语句。

当进行有参数的函数调用时需要将参数传入相应区域，同写语句一样，在这一时刻也需要进行一次导出，生成相关的代码。此外，由于无法确定被调用函数对全局变量的使用情况，在进行函数调用前需要将与全局变量相关的中间代码进行导出，写回新计算的值；当完成函数调用后，也无法确定被调用函数对全局变量的更改，因而需要采取同读语句一样的操作，将全局变量从结点表中删除。

经过了以上的更改基本解决了基本块中非计算代码的DAG图的生成。

## DAG图导出策略

在我最初的设计中，DAG图的导出策略是按照课本上的启发式导出策略。但经过分析发现，该启发式导出策略是基于x86架构的优化到处策略，因而我根据MIPS的架构设计了新的DAG图导出策略。

由于前面对于DAG图的设计，DAG图需要具有即时导出与继续更新的功能，因而需要重新设计DAG图的导出策略。最后我采用了树的后序遍历的方式导出DAG图，并且对DAG图的每个结点进行了导出状态标记。当需要导出DAG图时，只需按照结点更新的先后顺序遍历结点表，对为导出的根节点采用递归导出策略即可。在DAG图中，由于其可重复导出的特性，根节点的定义由原来的修正为无未导出父节点的结点。

采取以上方法DAG图便能够适应之前对于DAG的扩展设计。

## 复写传播与死代码删除

复写传播以及死代码删除的定义在此不多赘述。在我的理解下，该优化策略最主要的作用时“收紧”代码，将代码进行“极小化”操作。在进行主要的优化之前，编译器就对最初版本的中间代码进行了迭代多次的复写传播和死代码删除，简化中间代码的语法成分，最重要的是减少了原先通过常量内联计算出的死分支，减少了后续基本块的划分，提高了局部公共子表达式消除的力度，为之后的优化提供了较好的起点。

此外，完成各种优化后进行一次迭代的复写传播和死代码删除是有必要的。在中间的优化过程中，尤其是DAG图生成的到处阶段，会产生许多多余的赋值语句。如果不在完成优化后进行一次复写传播和死代码删除操作，产生的新代码会抵消优化的效果。

## 指令选择与取舍

在目标代码生成阶段，对于MIPS指令的选择会很大程度上影响生成程序的质量。比如对于比较跳转语句。如果不加处理直接使用与中间代码相应的指令，尤其是选用了仿真器提供的伪指令时，仿真器会产生等效的基础指令替换。这会导致每次判断时会执行一次无用赋值语句。而且比较跳转语句大多存在于循环的边界，当循环次数过大时会带来巨大的浪费。因而需要根据中间代码专门设计相应基础指令序列作为生成的目标代码。