代码优化设计

12061154 冯飘飘

一、代码优化设计方案

1.1 基本块内部的公共子表达式删除

1.1.1 基本块划分

在生成中间代码四元式时，我已经根据生成的四元式进行判断，看是否要进入下一个基本块，比如有新的跳转标签生成，则划分到下一个基本块

```
//中间代码，结构如 result arg1 op arg2
typedef struct MidCode
{
    char result[MAX_STR_LEN];    // result
    char arg1[MAX_STR_LEN];      // arg1
    char op[MAX_STR_LEN];        // op
    char arg2[MAX_STR_LEN];      // arg2
    int block;  //划分的基本块的编号
}MidCode;
```

一次性实现基本块的划分：

例如函数开始标签_funcname、$L、if、goto 时，则 blocknum++，blocknum 是对基本块块数的计数，统计有多少基本块并进行标记。

1.1.2 dag 图构建

设计 dag 图的节点结构如下

```
//节点表中的节点项
typedef struct NodeEntry
{
    char name[MAX_STR_LEN]; //节点名
    int node;       //所在节点
}NodeEntry;

//节点表
typedef struct NodeTable
{
    NodeEntry nentry[MAX_STR_LEN];  //节点项数组
    int size;       //节点表中几个节点项
//  int nodenum;//DAG图中有几个节点
}NodeTable;

//DAG图中节点结构体
typedef struct Node
{
    int node;   //第几个节点
    char op[MAX_STR_LEN];   //符号 or name
    char repr[MAX_STR_LEN]; //代表该节点的名字
    int reprnum;//标识符数量
    int left;       //左节点
    int right;      //右节点
    bool ready;     //是否已入栈，false为未入栈
}Node;

//DAG图结构
typedef struct DAGTree
{
```

```
    Node nodes[MAX_STR_LEN];//节点数组
    NodeTable nodetable;     //节点表
    std::stack<int> nodestack;  //中间节点序列
    int nodenum;             //节点数量
}DAGTree;
```

构建 DAG 图的方法：

1. 首先建立节点表，该表记录了变量名和常量值，以及它们当前所对应的 DAG 图中节点的序号。该表初始状态为空。
2. 从第一条中间代码开始，按照以下规则建立 DAG 图。
3. 对于形如 z = x op y 的中间代码，其中 z 为记录计算结果的变量名， x 为左操作数， y 为右操作数， op 为操作符：首先在节点表中寻找 x，如果找到，记录下 x 当前所对应的节点号 i；如果未找到，在 DAG 图中新建一个叶节点，假设其节点号仍为 i，标记为 x（如 x 为变量名，该标记更改 x0）；在节点表中增加新的一项(x, i)，表明二者之间的对应关系。右操作数 y 与 x 同理，假设其对应节点号为 j.
4. 在 DAG 图中寻找中间节点，其标记为 op，且其左操作数节点号为 i，右操作数节点号为 j。如果找到，记录下其节点号 k；如果未找到，在 DAG 图中新建一个中间节点，假设其节点号仍为 k，并将节点 i 和 j 分别与 k 相连，作为其左子节点和右子节点；
5. 在节点表中寻找 z，如果找到，将 z 所对应的节点号更改为 k；如果未找到，在节点表中新建一项(z, k)，表明二者之间的对应关系。
6. 对输入的中间代码序列依次重复上述步骤 3～5。

1.1.3 DAG 图导出中间代码

利用 dag 图完成消除局部公共子表达式后，还需要把 dag 图重新导出为中间代码，方便此后进行的其他优化。

1、 初始化一个放置 DAG 中间节点节点的队列。

2、 如果 DAG 图中还有中间节点尚未进入队列，则执行 3，否则执行 5

3、 选取一个尚未进入队列，但其所有父节点都已进入队列的中间节点 n，将其加入队列。或选取没有父节点的中间节点，将其加入队列。

4、 如果 n 的最左子节点符合 3 的条件，将其加入队列；并沿着当前节点的最左边，循环访问其最左子节点，最左子节点的最左子节点等。将符合 3 条件的中间节点依次加入队列。如果出现不符合的，转到 2

5、 将所有中间节点队列逆序输出，便得到中间节点的计算顺序。

先利用上述原则将所有非叶节点排序放在 dagtree.nodestack 栈中，然后分三步将导出优化后的中间代码：

1、 输出所有子节点的标识符赋值

2、 根据 dagtree.nodestack 的倒序输出四元式：

3、 每个节点中的标识符赋值也要输出。

1.2 寄存器的分配

寄存器的分配分为全局寄存器的分配和临时寄存器池的合理使用。

1.2.1 临时寄存器

临时寄存器池的使用采用即用即取，用完即释放的原则，对于所有临时变量都有计算其使用次数。使用一次 times-1，当 times = 0 时释放该临时寄存器。

1.3 窥孔优化

生成 DAG 图导出优化后的代码消除公共子表达式优化后，调用函数 kuikong() 对生成的所有四元式进行窥孔优化，优化对象主要是赋值形式的语句，由于在消除公共子表达式中，第二步用中间变量代替实参参加运算，第三步将所有实参赋值，这样会产生一些冗余代码。

比如 if (1) … else …则 else 后的代码即可删除

1.4 常量合并和传播

常数合并是将能在编译时计算出值的表达式用其相应的值的表达式用其相应的值替代，即如果在编译时编译程序知道这一表达式的所有操作数的值，则此表达式就可以由其计算出的值替代。

例如 a = 2 +3 ; 即可用 a = 5 代替