第4卷第1期
1993年2月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 4, No. 1
Feb., 1993

一种有效的编译优化代码移动算法

陈建华　陈涵生

(华东计算技术研究所，上海 201800)

A EFFICIENT ALGORITHM OF CODE MOVEMENT OPTIMIZATION IN A COMPILER

Chen Jianhua and Chen Hansheng

(*East China Institute of Computer Technology, Shanghai* 201800)

Abstract Code motion is an important global optimizing technology in a compiler. Here a new code motion algorithm is discussed, in which code motion (the elimination of common subexpression and the movement of loop-invariant computation, etc.)can be completed simply by data flow analysis without uncovering the loop structures embedded in the control flow graph, therefore it is a very efficient method.

摘要 代码移动方法是编译程序全局优化的一个关键技术. 本文将介绍一种新的代码移动算法. 用此算法就可实现公共子表达式的删除和循环不变运算的移动，而且此算法无需检测循环控制结构，只要通过数据流分析就可实现代码移动，因此这种方法十分有效.

§1. 引　言

代码移动是编译程序全局优化的一部分内容. 传统上，代码移动是指把代码从执行频率高的区域前移到执行频率较低的区域，而执行频率高的区域往往是指循环区域，移动的是循环不变量，而且代码移动是指前移而不包括后移，故也称作代码前提. 而本文所描述的代码移动方法就没有以上这些限制. 我们把代码移动分成前移和后移两种，代码前移实现公共子表达式的删除和不变运算的前提；代码后移实现冗余赋值的删除.

本文将集中论述公共子表达式删除技术和循环不变运算外移技术. 通常情况下，编译程序全局优化中公共子表达式的删除和循环不变运算的外移是分别进行的，并且不变运算是一个循环一个循环地从里往外移动的. 本文我们将把这两项工作合并起来，一起完成，并且只要通过数据流分析就可以把每个不变运算直接外移到循环的最外层.

这种思想方法最早是由 Morel 或 Renvoise 在 1979 年提出的[1]，并作了证明. 1983 年斯坦

本文1990年5月28日收到，1991年2月8日定稿.作者陈建华，1989年硕士毕业于华东计算所，主要研究领域为软件开发环境、编译及编译优化. 陈涵生，研究员，主要研究领域为软件工程、编译技术等.

福大学的一位博士生 Frederick C. Chow 用这种方法成功地实现了一个全局优化器[2],后来又把此成果转让给了 MIPS 公司,目前 MIPS 公司的 RISC 编译器都使用了此优化技术.但是前人所描述的算法还存在一些问题,有碍于优化的效率和质量.本文将参考 Morel、Renvoise 的思想方法和 Frederick C. Chow 的实现算法,从另一条途径来描述和改进代码移动的算法,从而得到了一种更有效、更彻底的代码移动方法.

§2. 属性定义

我们把布尔属性的思想应用到变量、表达式和赋值语句,并且依据基本块来定义这些属性.我们把属性分成局部和全局两种,局部属性以基本块内的某一点作为参考点,而全局属性则以整个基本块作为参考点,并且全局属性是通过一系列基本块的相互作用来决定的.

设 i 为一基本块;则基本块 i 的前趋集和后继集分别用 Pred(i)和 Succ(i)表示.如果 Succ(i)为空,则块 i 是程序(或过程)的出口块;如果 Pred(i)为空,则块 i 为入口块.

另外,我们在此文中将用"·"和"Π"表示布尔交,用"+"和"Σ"表示布尔并,用"~"表示非.局部属性:

关于变量的局部属性定义如下:

ANTLOC(局部先行)——如果在一基本块中可以把一个变量的引用移到该基本块的入口,而丝毫不影响该变量的值,则称该变量是局部先行的.记为:该变量的 ANTLOC 为 TRUE(真),简称该变量为 ANTLOC.

AVLOC(局部可用)——如果在一基本块中可以把一个变量的引用移到该基本块的出口,而丝毫不影响该变量的值,则称该变量是局部可用的.

ALTERED(局部可变)——如果在一基本块中变量的值可通过执行该基本块的代码来改变,则称该变量在此基本块中是可变的.

以上三种属性定义也适用于表达式,只要把以上定义中的"变量"用"表达式"替代即可.

注意,出现在基本块中的常数总是 ANTLOC、AVLOC 和~ALTERED(即其 ANTLOC 和 AVLOC 为真,ALTERD 为假).由于 ANTLOC 和 AVLOC 是通过代码前移和代码后移来定义的,因此 ANTLOC 是前趋属性,而 AVLOC 则是后继属性.

对赋值语句 a←⟨expr⟩,我们把其局部属性定义如下:

ANTLOC——如果一赋值语句 a←⟨expr⟩出现在某一基本块中,且若把该赋值语句移至该基本块入口,并不影响程序执行的结果,则称该赋值语句是局部先行的.换句话说,赋值表达式⟨expr⟩是可先行的(ANTLOC),并且赋值变量 a 是不可变的(~ALTERED),在基本块中赋值语句 a←⟨expr⟩之前的任何地方也都没有引用该变量.

AVLOC——如果一赋值语句 a←⟨expr⟩出现在某一基本块中,且若把该赋值语句移至该基本块的出口,并不影响程序执行的结果,则称该赋值语句是局部可用的.换句话说,赋值表达式⟨expr⟩是可用的(AVLOC),赋值变量 a 是不可变的(~ALTERED),并且在基本块中赋值语句 a←⟨expr⟩之后的任何地方不再引用该变量.

ALTERED——如果通过执行一个基本块的代码就可以改变赋值表达式⟨expr⟩或赋值变量 a 的值,或者在此基本块中存在赋值变量 a 的引用,则称该赋值语句是可变的.如果此赋值语句本身就在此基本块中,则不考虑此赋值语句本身的代码.

PAVLOC——如果一赋值语句 a←⟨expr⟩出现在某一基本块中，赋值变量 a 在该基本块出口之前一直拥有表达式⟨expr⟩的值，而且该表达式在出口之前也没有改变，则称该赋值语句是部分局部可用的. 换句话说，赋值表达式⟨expr⟩和赋值变量 a 的值在该基本块内赋值语句 a←⟨expr⟩后面的代码中不再改变.

ABSALTERED——如果通过执行一个基本块的代码就可能改变赋值表达式⟨expr⟩或赋值变量 a，并且假若此赋值语句本身就在此基本块中也不考虑此赋值语句本身的影响，则称此赋值语句是绝对可变的.

属性 PAVLOC 和 ABSALTERED 不同于属性 AVLOC 和 ALTERED，前者不考虑在相关域中是否引用赋值变量，它们的定义不依赖于代码移动. PAVLOC 是一个比 AVLOC 弱的属性，所以是 PAVLOC 的赋值语句不一定是 AVLOC，而是 AVLOC 的赋值语句一定是 PAVLOC. 相反，ABSALTERED 要比 ALTERED 强.

全局属性：

我们对全局属性的定义只扩展先行性和可用性的含义：

ANTGLOB——变量、表达式或赋值语句被称为在给定点是先行的，如果从该点出发的所有路径都包含这种计算的实例，并且在这些路径上任何地方所设置的这种计算都产生相同的结果.

AVGLOB——变量、表达式或赋值语句被称为在给定点是可用的，如果导向该点的所有路径都包含这种计算的实例，并且在这些路径上任何地方所设置的这种计算都产生相同的结果.

而部分先行性和部分可用性则是比较弱的属性：

PANTGLOB——变量、表达式或赋值语句被称为在给定点是部分先行的，如果至少从该点出发的一条路径包含这种计算，并且在这条路径上任何地方所设置的这种计算都产生相同的结果.

PAVGLOB——变量、表达式或赋值语句被称为在给定点是部分可用的，如果至少有导向该点的一条路径包含这种计算，并且在这条路径上任何地方所设置的这种计算都产生相同的结果.

全局属性通常应用于基本块的入口和出口，即给定点是基本块的入口和出口. 在基本块的入口，用 ANTIN、AVIN、PANTIN 和 PAVIN 表示这些属性；在基本块的出口，用 ANTOUT、AVOUT、PANTOUT 和 PAVOUT 表示这些属性.

以下布尔恒等系统定义了全局可用性属性的计算方法，这些属性基于相应的局部属性. 下标 i 表示是第 i 个基本块的属性.

可用性系统：

$$AVIN(i)=\begin{cases}FALSE & \text{基本块 i 是入口块；}\\ \prod\limits_{j\in Pred(i)} AVOUT(j) & \text{基本块 i 不是入口块.}\end{cases}$$

$$AVOUT(i)=AVLOC(i)+\sim ALTERED(i)\cdot AVIN(i)$$

第一个等式是说，一个计算项(指变量、表达式或赋值)在基本块 i 的入口是可用的，当且仅当它在基本块 i 的所有直接前趋的出口都是可用的. 第二个等式是说，如果一个计算项在基本块

i 中是局部可用的，或者在该基本块的入口可用并且在该基本块中没有被改变，则该计算项在基本块 i 的出口是可用的.

另外几组全局数据流属性可用相同的方法来计算：

先行性系统：

$$ANTOUT(i)=\begin{cases}FALSE & \text{基本块 i 是出口块；}\\ \prod\limits_{j\in Succ(i)} ANTIN(j) & \text{基本块 i 不是出口块.}\end{cases}$$

$$ANTIN(i)=ANTLOC(i)+\sim ALTERED(i)\cdot ANTOUT(i)$$

部分可用性系统：

$$PAVIN(i)=\begin{cases}FALSE & \text{基本块 i 是入口块；}\\ \sum\limits_{j\in Pred(i)} PAVOUT(j) & \text{基本块 i 不是入口块.}\end{cases}$$

$$PAVOUT(i)=AVLOC(i)+\sim ALTERED(i)\cdot PAVIN(i)$$

部分先行性系统：

$$PANTOUT(i)=\begin{cases}FALSE & \text{基本块 i 是出口块；}\\ \sum\limits_{j\in Succ(i)} PANTIN(j) & \text{基本块 i 不是出口块.}\end{cases}$$

$$PANTIN(i)=ANTLOC(i)+\sim ALTERED(i)\cdot PANTOUT(i)$$

以上数据流等式可用迭代算法来实现. 对于使用“Π”操作符的系统来说，在迭代之前先把未知量初始化为 TRUE(即真值)；而对于使用“Σ”操作符的系统来说，在迭代之前先把未知量初始化为 FALSE(即假值).

§3. 代码前移算法

本文我们将把代码移动看成是程序中计算的移动，这里的“计算”具有较广泛的含义，不仅包含表达式计算，而且还包含赋值(甚至还可以扩展为内存和寄存器的访问). 这样我们就可以把代码移动看成是一种程序数据流分析问题，通过数据流分析来确定计算插入和删除的位置，然后在插入点插入计算，在删除点删除计算，从而实现代码移动. 因此问题的关键是确定计算插入和删除的位置. 为此，我们再来定义一种全局属性：

PP (可能位置)如果计算 e 在 p 点处是可先行的，并且通过在 p 点插入计算 e，或在 p 点和过程内其它地方插入计算 e，而使所有可先行的计算 e 变成了冗余的 e，其中这些插入满足以下条件：插入总在这种地方进行，e 是可先行的，并且计算 e 插入之后，以前可先行的 e 就变成了冗余的 e，则称计算 e 在 p 点为 PP.

为了本文的需要，我们把上述 PP 定义中的点 p 指定为基本块的入口和出口，并且入口属性和出口属性分别用 PPIN 和 PPOUT 表示.

由文献[2]可知，进行代码移动的算法为：

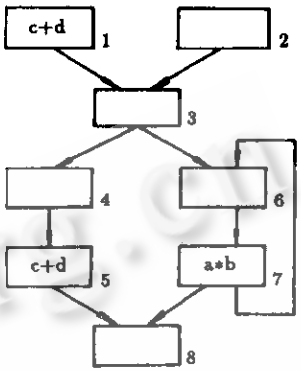
$$PPIN(i)=\begin{cases}FALSE & \text{基本块 i 是入口块；}\\ ANTIN(i)\cdot PAVIN(i)\cdot \prod\limits_{j\in Pred(i)}(PPOUT(j)+AVOUT(j)) & \\ \quad\cdot(ANTLOC(i)+\sim ALTERED(i)\cdot PPOUT(i)) & \text{否则.}\end{cases}$$

$$PPOUT(i)=\begin{cases}FALSE & \text{基本块 i 是出口块；}\\ \prod\limits_{k\in Succ(i)} PPIN(K) & \text{否则.}\end{cases}$$

$$INSERT(i)=PPOUT(i)\cdot\sim AVOUT(i)\cdot(\sim PPIN(i)+ALTERED(i))$$
$$DELETE(i)=ANTLOC(i)\cdot PPIN(i)$$

其中,INSERT 表示计算 e 是否在基本块 i 的入口插入,DELETE 表示计算 e 是否从基本块 i 中删除.这样就可根据算得的 INSERT 和 DELETE 实现代码移动,即在 INSERT 为 TRUE 的地方插入计算 e,在 DELETE 为 TRUE 的地方删除计算 e.

但是,通过分析我们发现此算法还存在一些问题,如图 1 所示,根据以上的算法就无法把结点 7 中的 a * b 提到循环 6—7—6 的外面,所以它还有碍于优化的彻底性.我们通过反复推敲和验证后,把上述算法改进为:



表达式 a * b 的 INSERT 属性在所有的结点上均为 FALSE.

图 1

(1)利用局部属性计算 AVOUT 和 PAVIN,其迭代方程为:

$$AVIN(i)=\begin{cases}FALSE & \text{基本块 i 是入口块;}\\ \prod\limits_{j\in Pred(i)} AVOUT(j) & \text{基本块 i 不是入口块.}\end{cases}$$
$$AVOUT(i)=AVLOC(i)+\sim ALTERED(i)\cdot AVIN(i)$$
$$PAVIN(i)=\begin{cases}FALSE & \text{基本块 i 是入口块;}\\ \sum\limits_{j\in Pred(i)} PAVOUT(j) & \text{基本块 i 不是入口块.}\end{cases}$$
$$PAVOUT(i)=AVLOC(i)+\sim ALTERED(i)\cdot PAVIN(i)$$

(2)利用局部属性和 PAVIN 求 PPIN 和 PPOUT,其迭代方程为:

$$PPOUT(i)=\begin{cases}FALSE & \text{基本块 i 是出口块;}\\ \prod\limits_{j\in Succ(i)} PPIN(j) & \text{基本块 i 不是出口块.}\end{cases}$$
$$PPIN(i)=PAVIN(i)\cdot(ANTLOC(i)+\sim ALTERED(i)\cdot PPOUT(i))$$

(3)对 PPIN 和 PPOUT 进行修正:

$$PPIN(i)=PPIN(i)\cdot\sum_{j\in Pred(i)}(PPIN(j)+AVOUT(j))$$

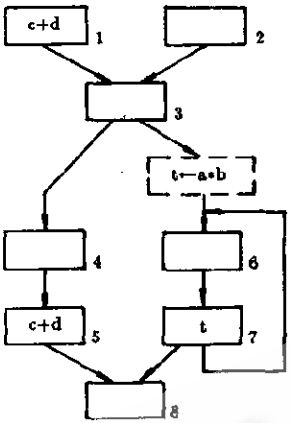
如果计算后发现 PPIN(i)为 FALSE,则给其所有直接前趋的 PPOUT 置 FALSE.

(4)利用以上求得的结果计算 INSERT、EDGE_INSERT 和 DELETE:

$$INSERT(i)=PPOUT(i)\cdot\sim AVOUT(i)\cdot(\sim PPIN(i)+ALTERED(i))$$
$$EDGE\_INSERT(j,i)=\sim PPOUT(j)\cdot PPIN(i)\cdot\sim AVOUT(j)$$

(EDGE_INSERT(j,i)表示是否要在结点 j 和 i 之间的边上插入计算,其中基本块 j 是基本块 i 的直接前趋)

$$DELETE(i)=ANTLOC(i)\cdot PPIN(i)$$



表达式 a＊b：它在结点 6 和结点 7 上的 PAVIN 和 PPIN 均为 TRUE，在其它结点上为 FALSE；

PPOUT（6）＝ TRUE，其余的 PPOUT 为 FALSE；AVOUT 和 INSERT 在所有的结点上均为 FALSE；

EDGE_INSERT（3，6）＝ TRUE，其余的为 FALSE；DELETE(7)＝TRUE，其余的为 FALSE.

表达式 c＋d：PAVIN(3)、PAVIN(4)、PAVIN(5)为，其余的为 FALSE；AVOUT、PPIN 和 PPOUT 在所有的结点上均为 FALSE；INSERT 和 DELETE 在所有的结点上均为 FALSE；EDGE_INSERT 在所有的边上均为 FALSE.

图　2

这样，按 INSERT 和 EDGE_INSERT 插入计算，按 DELETE 删除计算，就实现了更有效的代码前移（见图 2 所示，即把结点 7 中的 a＊b 提到了循环 6－7－6 的外面）．修改后的算法不仅为编译程序提供更多的优化机会，而且减少了算法的迭代次数．

§4. 代码后移算法

通过分析可知，表达式计算的冗余是一个部分可用性问题，而赋值的冗余则是一个部分先行性问题．这样我们就自然想到：既然表达式计算的冗余可通过代码前移来消除，那么赋值冗余是不是可通过代码后移来消除呢？事实证明了这一点，请看图 3，由于循环内任何地方都没有引用变量 a，因此循环过程中语句 a←i＋9 对 a 的赋值出现冗余．图 3 说明，我们不能把 a←i＋9 前移到循环体外，而后移却能得到良好的效果．因此我们将用代码后移的方法来消除前移无法删除的赋值冗余．

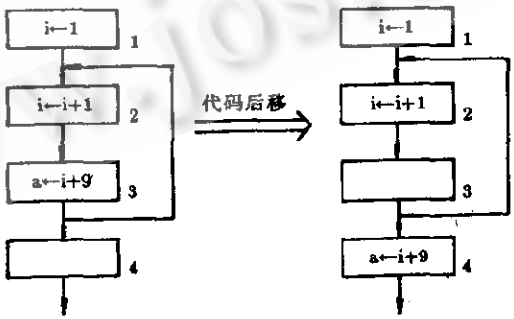


图　3

事实上，赋值冗余和表达式冗余没有本质的差别，只是移动的方向不同而已．假如我们把方向颠倒过来，就可以用上节介绍的方法来消除赋值冗余．就是把所有参数和属性的方向颠倒

过来：OUT(==)IN、AND(==)AV 和 Pred(==)Succ. 另外，这里我们不把赋值插入到基本块的出口，而是插入到基本块的入口. 这样，我们就可得到消除赋值冗余的数据流方程：

$$PPIN(i)=\begin{cases}FALSE & \text{基本块 i 是入口块；}\\ \prod_{j\in Pred(i)} PPOUT(j) & \text{基本块 i 不是入口块.}\end{cases}$$

$$PPOUT(i)=\begin{cases}FALSE & \text{i 是出口块；}\\ PANTOUT(i)\cdot(AVLOC(i)+\sim ALTERED(i)\cdot PPIN(i))\cdot\sum_{j\in Succ(i)}(PPOUT(j)+ANTIN(j)) & \text{i 不是出口块.}\end{cases}$$

$$INSERTIN(i)=PPIN(i)\cdot\sim ANTIN(i)\cdot(\sim PPOUT(i)+ALTERED(i))$$

$$EDGE\_INSERT(j,i)=\sim PPIN(j)\cdot PPOUT(i)\cdot\sim ANTOUT(j)$$

$$DELETE(i)=AVLOC(i)\cdot PPOUT(i)$$

在应用以上方程进行数据流分析时，有一种情况必须考虑到：对赋值语句 a←⟨expr⟩来说，如果 a 在不同的路径上被赋于不同的值，而且这些路径将会聚于某一点，则该赋值语句不能移到会聚点处. 为此，在求解 PPIN 和 PPOUT 时必须增加一些限制以得到正确的结果. 我们的解决办法是，当有多条路径会聚于一点时，就把该点处 a←⟨expr⟩的 PPIN 初始化为 FALSE. 这样，PPIN 在整个迭代过程中将保持“FALSE”值，从而不会把赋值语句移过会聚点.

代码前移和代码后移都能消除冗余赋值，虽然它们的方法和原理基本相同(只是方向不同而已)，但是它们所执行的优化却不发生一点重迭.

§5. 结束语

代码优化的概念很早就有人提出来了，前人在这方面已做了不少工作，但是由于优化的难度和复杂性都比较大，因此能真正实用的优化程序比较少，而包括大部分优化技术的优化器那就更少了. 到现在为止，虽然在杂志和其它刊物上发表的各种优化算法也不少，但是大多是孤立的. 若要把所有优化纳入一个优化器就比较困难，要么难以实现，要么效果差. 本文所描述的算法为全局优化开辟了一条新的途径. 这种代码移动算法除了可简化个别程序变换处理外，还可把以前一些独立的全局优化确定为一些公共处理的特殊情况，所以采用这种代码移动方法的优化程序只要用少数几遍就能完成所有的全局优化工作. 与常规的技术相比，这种方法会减少优化程序的复杂性和实现操作，并且这种方法的优化速度与循环的数量和嵌套深度无关. 所以这种方法具有很大的实用价值.

笔者花了近两年时间在 ADA 的中间语言 AIM 上作了一个全局优化的实验性系统，实现了上述的代码移动，效果良好.

参考文献

[1]E. Morel, C. Renvoise. Global Optimization by Suppression of Partial Redundancies. Communications of ACM, 22 (1979), 96－103.

[2]F. C. Chow. A Portable Machine－Independent Global Optimizer——Design and Measurements. PhD Thesis, Dep.

of Electrical Engineering and Computer Science, Stanford University, CA, Technical Report No. 83—254.
[3]陈建华,编译程序全局优化算法的研究与实现,硕士论文,华东计算技术研究所,1988 年 12 月.

第五届全国逻辑设计自动化学术会议——LDA'93

征 文 通 知

中国计算机学会 CAD 与 CG 专业委员会逻辑设计自动化学组定于 1993 年 10 月在浙江省宁波市召开第五届全国逻辑设计自动化学术会议,由上海科技大学主办,欢迎有关专家、学者、研究人员踊跃投稿,兹将有关事项通知如下:

一、征文范围:硬件描述语言及模拟技术,微程序设计技术,高级逻辑综合理论,逻辑设计中数据库管理技术,组合逻辑综合,逻辑设计形式验证,时序逻辑综合,可测试逻辑设计,智能化逻辑设计技术,硅编译器技术,PLA 分解及折叠,数字逻辑系统的划分,其他有关逻辑设计自动化专题。

二、征文要求:1. 论文要求反映新的研究思想与成果,未在其他会议或刊物上发表。2. 正文字数一般为 5000 字,最多不超过 6000 字,请附 300 字以内的论文摘要,来稿请自留底稿。3. 会议将出版论文集,打印清样格式与出版费在录取时另行通知。4. 投稿地址为:**上海科学技术大学计算科学系　王芳雷　收**　邮政编码:201800。5. 论文截止日期:1993 年 5 月 30 日。

中国计算机学会
CAD 与 CG 专业委员会
逻辑设计自动化学组
1992 年 11 月 5 日