基于优先级着色的寄存器分配

马 威　陈是荣　周修廉
(哈尔滨科技大学)

REGISTER ALLOCATION BY PRIORITY-BASED COLORING

MA WEI　CHEN SHIRONG　ZHOU XIULIAN
(*Harbin University of Science and Technology*)

Abstract　The register allocation can be considered as a graph coloring problem. Every node in the graph stands for candidates of the register, and two nodes are connected by an edge if two candidates interfare with each other, i. e. they live simultaneously at certain points in the objective program. This paper presents a algorithm for the MODULA-2 compiler "Register Allocation by Priority-based coloring" in VAX-11/VMS. As for time, the above algorithm enables the process of the spilling problem to run in the linear level, rather than the exponential level.

摘要　寄存器分配可看作为一个图着色问题，图中每个结点代表了一个硬件寄存器的候选者，如果两个候选者使用寄存器发生冲突则两者之间有一条边，即它们同时在某些点活跃。本文提出了在VAX-11/VMS机器上实现MODULA-2编译中使用的基于优先级着色的寄存器分配算法。使得图着色寄存器分配中处理Spilling问题的时间从指数级变为线性级。

一、引　言

Aho在1986年出版的"Compilers principles, techniques and tools"[2]一书中论述了"用图着色进行寄存器分配"的方法。其思想是对寄存器分配过程使用二遍扫描。首先将中间代码转变为机器代码，对寄存器的使用则假定有无限可用的临时寄存器(T-regs)。对每一个是寄存器的候选者都给分配一个临时寄存器，同时记录有关T-reg使用的各种信息。第二遍扫描则是将物理寄存器分配给T-regs，目标是找到一种具有最小内存消耗的分配方案。在这次扫描中，对于每一个过程，建立一个寄存器冲突图，图中的结点是T-regs，如果其中一个结点是活跃的，而另一个结点在这点有定义，则这两个结点之间用一条边连接。

如下面的四元式代码：

```
      ⋮
10)   T0:=b+c;
```

本文1988年8月8日收到。

```
11>  T1:=d-b;
12>  T3:=T0+f;
      ⋮
```

T0 在第 10 条语句中有定义，在第 12 条语句中用到，则 T0 在第 11 条语句上是活跃的。而 T1 在这点有定义，则 T0 与 T1 之间有条边连接。

着色是使用 K 种颜色对寄存器冲突图的各个结点进行着色，K 是实际可利用的物理寄存器数目，如果每个结点都被着色，且相邻结点之间没有相同颜色，则称这个图是可着色的。一种颜色代表了一个物理寄存器，这保证了在任一给定时刻没有两个活跃的结点分配在同一个物理寄存器里。

判断一个图是否是可着色的，当 K>2 时是一个完全 NP 问题，因为它涉及到为每个结点猜测颜色，如果着色失败，则需要回溯。标准的图着色算法仅当第一次试着色过程成功，它的运算时间为线性的。当冲突图不是 K 种颜色可着色的，或接近于边界情况，它必需回溯来尝试所有种颜色的组合，即这时需要指数级的运算量来证明情况的确是如此。"但是我们可加一些限制条件和用一些启发式的逼近技术来使着色问题实际可行"[2]。

标准的图着色算法总是尽可能多地将寄存器的候选者放入寄存器，并不考虑候选者进入寄存器是否有益，也不考虑程序的循环结构。实际上，变量在执行最频繁的程序段里是应该考虑给予进入寄存器的。当它发现 K 不可着色图时，就要决定将哪个结点(变量)从着色图中排除，即 Spilling 某个结点。这一过程是非常复杂的。通常，Spilling 决策是与着色的决策相互独立的，即它很难预测 Spilling 某个结点会给后面的着色判断带来什么影响。

GREGORY 等人[3]在 IBM SYSTEM/370 PL/I 编译中，将此思想用于寄存器分配过程，取得了很大的收益。"使得产生后的代码接近于手编代码的程序，并且对于大程序来说，可能在某种程度上优于手编代码[3]。但 GREGORY 等人在处理 Spilling 问题时，在一些情况下，运算时间可能成为指数级。

这里，我们提出了一个图着色分配寄存器的算法，这个算法克服了以上提到的标准图着色算法所带来的问题。其基本思想是给每一个进入寄存器的候选者都赋予一定的优先级，并且按照这种优先级的次序分配寄存器。此算法不需要回溯，运算时间与寄存器的数目和可能分配寄存器的 T-regs 的活跃范围数成比例。

二、局部代码生成及寄存器分配

1. 局部代码生成

我们在 VAX-11/VMS 系统上实现 MODULA-2 编译的寄存器分配工作是基于图着色的原理，用了局部代码生成和寄存器分配这两个过程来完成二遍扫描。

局部代码生成过程进行第一遍扫描，首先将四元式中间代码变为机器代码，给每一个寄存器候选者分配一个 T-reg，同时放入有关寄存器候选者对寄存器使用的有关信息。其中主要有 use-priority，其计算公式如下：

$$\text{use-priority}=\sum_{i=1}^{n}\text{priority}_i*(10^{\min(6,\text{loopdepth}_i)})+\text{loopdepth}_i$$

公式中 priority 取值 1 或 4。1 代表了此临时变量可放在寄存器里，也可放在内存中。4 代表了此临时变量必须放在寄存器中。例如：此临时变量用于变址寄存器时，它的 priority 为 4。i 是指第 i 次此临时变量出现。loopdepth_i 指的是第 i 次临时变量出现时所在的循环深度。$\min(6, \text{loopdepth}_i)$ 指的是取 6, loopdepth_i 中较小的一个数。

另外还有 T-reg 的活跃范围：first-use, last-use, 使用寄存器数目及是不是过程参数等信息。

2. 寄存器分配

寄存器分配过程进行第二遍扫描，它首先根据每个临时变量的使用信息得到寄存器分配的启发式信息 weight（权）。weight 的计算公式如下：

$$\text{weight} = \frac{\text{use-priority}}{(\text{final-use} - \text{first-use}) \times \text{number-of-register}}$$

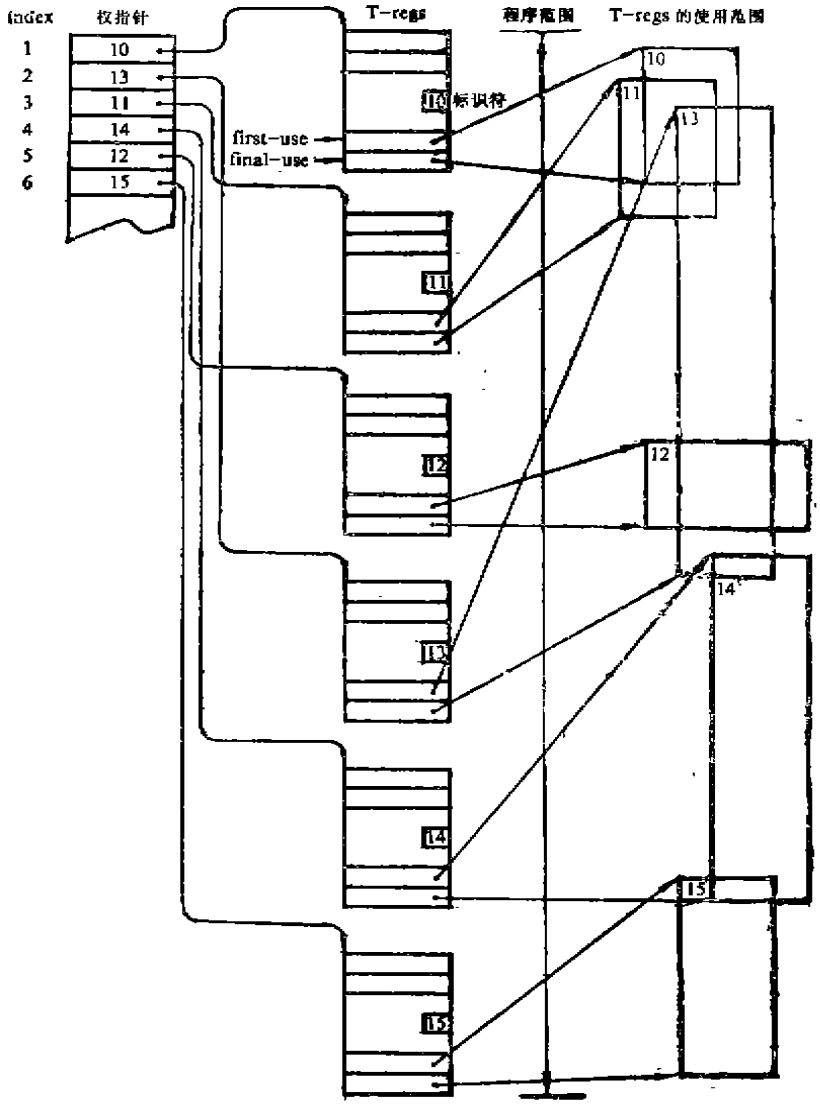

图 1　基于优先级着色的寄存器分配过程

其中 use-priority 如上所述，first-use 和 final-use 为此 T-reg 第一次使用和最后一次使用的使用点，它们之差用来决定此临时寄存器的活跃范围。number-of-register 是指此临时变量占用硬件寄存器的数目。

每一个 T-reg 的 weight 在寄存器分配过程中起着重要作用，weight 最大的 T-reg 优先分得寄存器。标准的图着色算法在寄存器不够的情况下是将变量 Spilling，即分配内存。我们处理这种情况是将 T-reg 的活跃范围分解为若干个小范围，使其最终能够分得寄存器或部分活跃范围分得寄存器。在实现上，我们暂时只对用作基地址寄存器与变址寄存器的 T-regs 加以考虑。

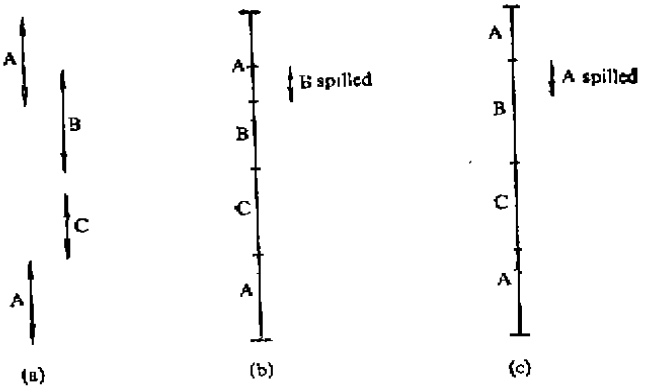

图 2　进一步分解活跃范围

图 1 与图 2 分别说明了基于优先级的寄存器分配过程和分解活跃范围，以便更充分地利用寄存器的过程。

在图 1 中，某一程序段涉及了 6 个 T-regs，它们的标识符依次为 10, 11, …, 15，其活跃范围分别由 first-use 和 final-use 指出，权指针数组里的每一项内容为各个 T-reg 的标识符，T-regs 在权指针数组里出现的顺序是按照各自权的大小，即标识符为 10 的 T-reg 具有最高优先权，因为它排在权指针数组的第一项。

假设现在有 2 个可利用寄存器 R_1, R_2，这时分配过程如下：

T-reg10 首先分到了 R_1，这时由于 T-reg11 与 T-reg13 的活跃范围与之相交，所以，它们不能分到 R_1。第二个是 T-reg13 分得 R_2，由于 T-reg11, T-reg12, T-reg14 的活跃范围与之相交，所以它们不能分到 R_2。第三个是 T-reg11，由于它既不能得到 R_1，也不能得到 R_2，这时，它必须 Spilled。依次处理得 T-reg14 分得 R_1，T-reg12 分得 R_1，T-reg15 分得 R_2。

图 2 表示了在这一程序范围内，有 T-regs A、B、C，它们分别是寄存器的候选者。虽然 T-reg A 有二个活跃范围，但实际上，一开始它被认为是一个活跃范围。即活跃范围覆盖了这一程序范围。假设现在只有一个寄存器来满足这三个 T-regs 的要求。

图 2 中(b)、(c)表示了两种可能的分配结果。图 2(b)是 T-reg A 和 C 首先分到寄存器，这时 T-reg B 的活跃范围被分解为二个活跃范围，它的一部分活跃范围分到了寄存器，而另一部分 Spilled。

图 2(c)表示了 T-regs B, C 首先得到寄存器，T-reg A 的活跃范围被分为二部分，如图 2(a)。这时 T-reg A 底部的活跃范围得到了寄存器。上部活跃范围则继续分为二部分，一部分得到了寄存器，另一部分则 Spilled。选择哪种方案取决于 T-regs 的优先级。

三、基于图着色的寄存器分配算法

假设有 $R_1 \sim Rr$ 个可利用的物理寄存器。

算法如下：

1．按照寄存器分配第一遍扫描里得到的各 T-regs 的 use-priority 及其它信息，计算一个划分(按照活跃范围划分)里的 n 个 T-regs 的权，并按其权的大小在权指针数组里排列。i:=0。

2．重复执行 a 到 b，直到处理完一个划分里的全部 T-regs。

a〉 i:=i+1。从 R_1 到 R_r 判断 T-regi 是否可着色，如果可着色，则对于 T-reg i+1 到 T-reg n 建立与 T-reg i 相关的冲突图。(T-reg i 表示 weight 数组里第 i 项)

b〉 如果 T-regi 不可着色，则 Spilling T-reg i，这时如果 T-reg i 为基地址寄存器或变址寄存器之用，则分解其活跃范围(实际上，一般情况下，作为基地址和变址寄存器之用的T-reg，其权是很大的，将会立即分到寄存器的)。

3．按照着色图分配硬件寄存器，如果处理完全部的 T-regs，则结束。否则，重复第 1 步，处理下一个划分。

算法复杂性估计：

我们主要考虑算法第 2 步 a〉的复杂性，因为它占用了整个算法的主要时间。我们从最坏情况分析。第 i 个 T-reg 指 weight 数组里的第 i 个 T-reg。

一个划分里第 1 个 T-reg 从 R_1 到 R_r 做 r 次判断得到一个可利用寄存器 R_i，这时，需要从第 2 个 T-reg 一直到第 n 个 T-reg (假设一个划分里含有 n 个 T-regs) 通过判断处理来建立寄存器冲突图，执行次数为 $r+(n-1)$。第 2 个 T-reg 从 R_1 到 R_r 判断 r 次得到一个可利用寄存器，需要从第 3 个 T-reg 一直到第 n 个 T-reg 来判断建立寄存器冲突图，执行次数为 $r+(n-2)$。直到最后一个 T-reg 的执行次数为 $r+1$。

加起来得到

$$r+(n-1)+r+(n-2)+r+(n-3)+\cdots+r+1=(n-1)r+\frac{(n-1)n}{2}$$

$$=(n-1)\left(r+\frac{n}{2}\right)$$

算法是

$$O(n(r+n))$$

此算法在处理图着色寄存器分配时，采用了基于优先级分配的方法，使在处理 Spilling 问题时，不需要回溯，使得用图着色原理分配寄存器的方法切实可行。图 3 是寄存器分配过程分配寄存器的一个实例。图中从 MODULA-2 源程序到中间代码的工作是由 Front END 部分做的，从中间代码到寄存器分配——即代码生成是由寄存器分配过程完成的。图里中间代码中 Ti、ti 分别表示了临时变量，ti 是表示对临时变量的间接访问。

四、结 束 语

Edward、Lowry 和 C. W. Medlock 总结早期 OS/370 FORTRAN 编译时说"寄存器分配为优化提供了一种非常重要的技术"[4]。

用图着色原理进行寄存器分配的过程，使得寄存器分配与局部代码生成分开，成为一个相对独立部分，这点是很重要的。我们在实践中，许多收益是很明显的：

i) 能从总体上按照临时变量的使用情况来分配寄存器。

```
S12$SIA2:[CLS. CLS3]MW. C;34
MODULE m0;
  VAR   v1: INTEGER;
        v2: ARRAY [0..3] OF INTEGER;
        v3: ARRAY [0..9] OF INTEGER;
  PROCEDURE p (p1: INTEGER; VAR p2:
    ARRAY OF INTEGER; p3:ARRAY OF
INTEGER);
  BEGIN
    p2[2]: =4;
  END p;
  BEGIN
    p(20, v2, v3);
END m0.
-$12$SIA2:[CLS. CLS3]NAME. Q;70
PH Sp N N
SZ I14 N N
FP N I2 04
FP N I2 012
FP N I2 08
FP N I2 020
AL 020 N 016
FP N 020 016
AD N I12 T0
ID t0 I4 T4
IS T0 I4 T8
AS t8 I4 T12
BC T4 N I2
AS I2 I4 T16
IM T16 I4 T20
IA T12 T20 T24
AS I5 I4 t24
RT N N N
MH Sm0 N N
SZ I60 N N
PR IO N N
AD N I4 T28
AD N I20 T32
PC N Sp N
AP I20 I4 N
AP T28 I4 N
AP I16 I4 N
AP T32 I20 N
AP I40 I4 N
ME N N N
-$12$SIA2: [CLS. CLS3] NAME. E; 173
all 1::   . long ^ Xffffffff
iupper_over::   . ascid/integer upper limit over/
ilow_over::   . ascid/integer low limit over/
over_limit::   . ascid/limit over/
cupper_over::   .ascid/candinal upper limit over/
clow_over::   . ascid/candinal low limit over/
var_error::   . ascid/variant limit over/
tem:
  .blkl 5
temvar:
  .blkl 1000
m0:
  .blkb 60
m0pri:
  .long 0
  .title m0
  :psect m0
'entry Mm0, ^M(r2, r3 , r11)
  moval m0, r11
  moval 4(r11), r3.
  moval 20(r11), r2
  pushl #40
  pushl 16(r2)
  pushl 12(r2)
  pushl 8(r2)
  pushl 4(r2)
  pushl 0(r2)
  pushl #16
  pushl r3
  pushl #20
  calls #9, p
pend::
  $exit_s
  ret
  .end Mm0
-$12$SIA2: [CLS. CLS3]PNAME. E; 47
  .title p
  .psect p
.entry p, ^M(r3, r3, r4, r5)
  subb2 #14, sp
  moval 12(ap), r2
  divl3 #4, (r2), r4
  subl3 #4, r2, r3
  movl (r3), r5
  cmpl r4, #2
  bgeq 1000$
  pushal over_limit
  calls #1, G^lib$put_line
  jmp pend
1000$:
  movl #2, r2
  mull3 r2, #4, r3
  addl3 r5, r3, r2
  movl #5, (r2)
  ret
.end
```

图3　寄存器分配实例

(下转第 36 页)

为了建立FMS与DTR之间的联系，还应开发FMSDTR工具，其目的帮助用户利用FMSDTR中的显示、存储和修改、删除等过程。

我们设计的FMSDTR工具是用BASIC、VA-X的DCL命令语言开发的，其程序处理流程图如图3所示：

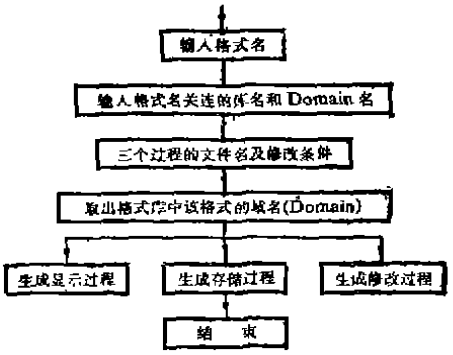

图3 FMSDTR过程生成框图

6. 应用情况

我们研制的汉字FMS软件工具已成功地安装在上海港煤炭装卸公司管理信息系统中。经过一年多时间的运行，用户反映良好。正如上海港煤炭装卸公司MIS开发工作报告中指出的："在物理设计阶段的开发过程中，要敢于创新，要敢于提出新思想，应用新技术[4]。为此开发组进行了FMS使用中文实践的尝试，事实证明：使用汉字FMS软件工具不仅加快设计速度，而且使屏幕格式统一美观。"

参 考 文 献

[1] VAX FMS Reference Manual.

[2] 朱三元等，"国外软件工具研究综述"，《计算机应用与软件》，1984，No. 4，5，6.

[3] 潘荫荣等，"全屏幕输入程序自动生成器"，《计算机工程》，1990年，第3期.

[4] 潘荫荣、范力，"以数据库设计为核心的软件工具的综合使用技术"，《中国计算机用户》，1988，No. 30.

（上接第49页）

ii) 能使用多于16个寄存器(VAX-11/VMS有16个硬件寄存器)。我们在局部代码生成时能够将可成为寄存器的候选者全部放入寄存器临时变量，即认为无限个可利用的寄存器，而不用考虑将其赋予硬件寄存器时所遇到的问题。

iii) 全局优化，局部代码生成，寄存器分配，窥孔优化成为总体上各自独立的阶段。

参 考 文 献

[1] Alo, A. V. and Ullman, J. D., "Principles of Compiler Design", Addison-Wesley, 1977.

[2] Aho, A. V., "Compilers Principes, Techniques, and Tools", 1986.

[3] GREGORY J. CHAITIN, "Register Allocation VIA Coloring", Computer Languages, Vol. 6, pp. 47～57, 1981.

[4] Edward S. Lowry, "Object Code Optimization", Comm. ACM., 12, No. 1, 1969.

[5] 马骁，VAX-11 MODULA-2 编译的实现"，哈尔滨科技大学研究生论文，1988，6.