嵌入式处理器寄存器分配的一种混合演化算法*)

吴圣宁　李思昆
(国防科技大学计算机学院　长沙 410073)

摘　要　通用处理器的寄存器分配一般采用图着色的方法。除非考虑特例，优化的图着色是 NP 完全性问题。因此，传统寄存器分配常利用图着色的启发式算法，并能对规则的 RISC 处理器生成质量较高的代码。但由于嵌入式处理器不规则的体系结构特征，这种传统寄存器分配方法生成的代码质量不能满足嵌入式领域的要求。本文提出了一种新的遗传算法和局部搜索相混合的元启发式方法，能较好地克服传统寄存器分配的不足。实验结果表明，这种新的算法比传统图着色寄存器分配算法减少约 30%spill 代码。

关键词　寄存器分配，演化算法，嵌入式处理器

A Hyrid Evolutionary Algorithm for Register Allocation of Embedded Processors

WU Sheng-Ning　LI Si-Kun
(School of Computer Science, National University of Defense Technology, Changsha 410073)

Abstract　A popular approach to the register allocation problem is based on a graph model. Unless considering only special cases, optimal coloring is an NP-complete problem, so that traditional register allocators have to resort to heuristics and can give rather good code for RISC processors, but the quality of code generated by them cannot meet the requirements of embedded domain, due to irregular architectural features. This paper brings forth a meta-heuristic algorithm composed of a new genetic algorithm and local search, which can overcome the shortcomings of traditional register allocators. Experimental results show that this algorithm can reduce about 30% spill code, compared to traditional graph-coloring register allocators.

Keywords　Register allocation, Evolutionary optimization, Embedded processors

1　介绍

嵌入式系统广泛地应用于工业控制、日常电器中，例如飞机和汽车中的控制系统、激光打印机、移动电话等；它们常具有体积小、功耗低、实时性等要求。因此，嵌入式处理器常为应用需求提供体系结构支持，例如复杂指令、双内存部件、地址生成部件等，从而造成不规则的体系结构特征。这增加了编译优化的复杂性，并使得一些传统的编译优化技术不能直接应用于嵌入式领域。

寄存器分配是重要的编译优化技术之一。传统寄存器分配常基于图着色模型。代码选择阶段生成的汇编指令通常并不直接涉及物理寄存器，而是假定有无限数量的 virtual 寄存器。寄存器分配问题是把这些 virtual 寄存器映射到有限的物理寄存器，在不能完成时生成 spill code，把结果放到存储器。寄存器分配问题的一个实例可以表示为一个冲突图。在此模型中，每个节点表示一个 virtual 寄存器，每条边表示所连接的两个 virtual 寄存器有冲突，即在程序某个执行点上是同时活跃的，不能分配到同一个物理寄存器。如果我们把 k 个物理寄存器看作是有 k 种颜色，那么寄存器分配的任务就是用最多 k 种颜色去为冲突图着色，使得任何相邻节点赋予不同的颜色。

除非考虑特例，优化的图着色是 NP-complete 问题。因此，传统寄存器分配常采用简单的启发式方法。这种方法假定同构的寄存器文件，即这些寄存器相互没有区别，在指令操作中可以互换使用。但嵌入式处理器的寄存器文件常是不规则的，不满足同构的假定，而且寄存器的数量也常比 RISC 处理器少。因此，这种简单启发式寄存器分配生成的代码质量难以满足嵌入式系统要求。

嵌入式处理器代码质量的重要性远高于编译速度。传统编译优化技术的简单启发式方法，虽然编译速度快，但只能搜索代码生成问题解空间的一小部分，因此代码质量不够高。元启发式算法(Meta-heuristics)，例如禁忌搜索、遗传算法，通过有效地搜索优化问题的解空间，较好地解决了许多实际应用中的优化问题。本文针对嵌入式处理器寄存器文件的不规则体系结构特征，提出了一种新的遗传算法和局部搜索相混合的元启发式算法，基于图着色原理，有效地解决寄存器分配问题；是以编译时间的增加来换取代码质量的提高。

本文第 2 节给出相关研究；第 3 节描述我们的混合演化算法；第 4 节是实验结果和分析；最后是结论。

2　相关研究

自 Chaitin[1] 的工作开始，对基于图着色的寄存器分配有大量的研究。但大多数都基于规则寄存器文件的假定，即寄存器是独立的和可交换的。只有少数适用于嵌入式处理器不规则体系结构的寄存器分配算法。

Briggs[2] 把 Chaitin 的悲观图着色方法改进为乐观的方法。George 和 Appel[3] 交错使用 Chaitin 的 simplification 和 Briggs 的 conservative coalescing，进一步改善了这类算法。

*)本课题得到国家自然科学基金(9027019)和国家 863 计划(2002AA1Z1480)的资助。吴圣宁　博士生，主要研究领域为嵌入式系统编译器、操作系统、人工智能；李思昆　教授，博士生导师，研究方向为电子 CAD、系统芯片设计方法学、虚拟现实。

这些算法常扩展着色或 spilling 的启发方法。类似的研究还有文[4]。

对不规则寄存器文件，主要研究有图着色和非图着色两类。Briggs[5]描述了可对 aligned 和 unaligned register pairs 进行图着色的寄存器分配算法。Nickerson[6]的算法可处理多个连续的寄存器，称为 clusters。Smith 和 Holloway[7]的算法给冲突图的边增加权值，从而修改图可着色的测试规则，来扩展基本图着色算法。Runeson[8]的可重定向图着色寄存器分配算法采用<p,q>测试，可处理更广泛的寄存器文件特征。Smith 和 Ramsey[9]的图着色寄存器分配通用算法，从寄存器分类的角度考虑冲突图节点的可着色约束，可获得更高质量代码。

一些研究者把寄存器分配问题表示为 integer linear programming(ILP)[10]问题。这些研究的主要局限在于：建模复杂，求解时间开销过大以至于只能求解中小规模问题。

3 寄存器分配的混合演化算法

我们首先介绍 Chaitin 寄存器分配算法的原理，然后说明 Chaitin 及其扩展算法在嵌入式处理器不规则体系结构特征环境中遇到的困难。最后详细描述混合演化算法。

3.1 Chaitin 寄存器分配算法

如图 1 所示，Chaitin 图着色寄存器分配器主要有五个子模块。

(1)build：构造冲突图。

(2)coalesce：移除不必要的 move 指令；当 move 的源和目标操作数没有冲突时，把它们合并为一个节点。

(3)simplify：用简单的启发方法对图着色。假定共有 k 种颜色，从图中重复地移除邻居个数小于 k 的节点，并把它压入堆栈。每次这样的移除也降低了邻居节点的度。

(4)spill：假定 simplify 之后，图中剩余节点的度都大于等于 k。这样 simplify 启发方法不能再应用，需要选择节点把它放到内存中。由于这样修改了程序变量的活跃性关系，则重新建立冲突图，前面的过程重新开始。

(5)select：对图中节点着色。每次从堆栈中弹出一个节点，给它一个和邻居不冲突的颜色。

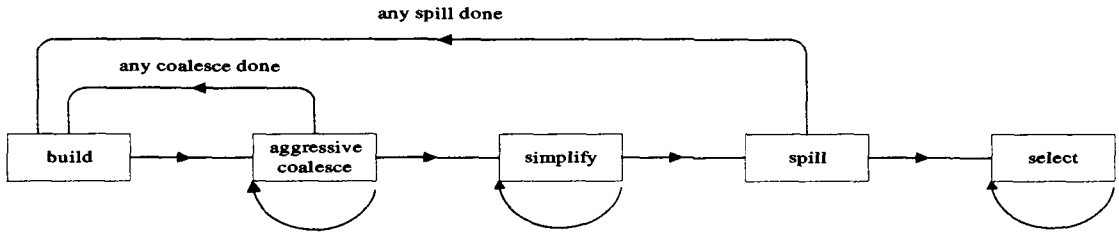

图 1 Chaitin 寄存器分配算法的主要流程

Chaitin 算法对图中节点是否可着色的测试，在于节点邻居的个数小于颜色数 k。满足此条件的前提下，节点邻居所分配颜色的总数必然小于 k，因此，总可以有颜色赋予此节点而不会发生冲突。

但如图 2(a)所示，即使这样简单的图，Chaitin 算法也无法用两种颜色着色而不 spill。两种颜色着色的方案显然存在，例如 a 和 c 着红色，b 和 d 着绿色。

因此，Briggs 扩展 Chaitin 的悲观启发为乐观启发，即使节点邻居个数不小于 k，也可以把节点移除压栈，因为所有邻居所占有的颜色可能有相同的。

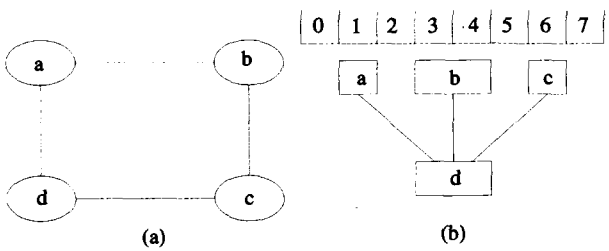

图 2 两种冲突图。(a)中所有节点是同构的；(b)中有两类节点，分别占用 8 个物理寄存器中的 1、2 个

但嵌入式处理器寄存器文件常是不规则的。如图 2(b)所示，各节点所需的物理寄存器个数不同，另外还存在物理寄存器重叠的现象，aligned 和 unaligned 的要求，等。这样，Chaitin 算法中节点可着色的测试变得复杂，基于 Chaitin 的扩展算法虽然有所改进，但不能从根本上克服此困难，造成代码质量的损失。

3.2 混合演化算法

我们既想克服 Chaitin 算法在嵌入式环境中遇到的困难，又想保留图着色的直观性。因此，我们提出了用元启发式算法来解决问题。基本思想是：把冲突图中的节点分类，按类随机赋予各节点颜色，检测图中的冲突数(作为 fitness 函数)，在一定的迭代次数后如果还有冲突，spill 造成冲突数最多的节点，建立新的冲突图，重新开始，直到无冲突为止。

考虑到程序中有大量的不必要的 move 指令，减少图的节点数可以提高效率，我们以 Chaitin 算法作为前端，利用其 aggressive coalesce 合并节点，利用其 simplify 把满足保守的可着色测试的节点压栈。如果此时图为空，则依然走 Chaitin 的流程，成功着色。否则，进入混合演化算法，对图中剩余节点着色。最后，对 Chaitin 前端压栈的节点着色。至此，整个流程结束。如果演化算法需要 spill 节点，则冲突图需重新构造，再次转到 Chaitin 前端，重新开始。混合演化算法的主体由遗传算法和局部搜索组成。

3.2.1 遗传算法

遗传算法的作用是解空间的 exploration，避免局部搜索收敛于局部次优解。其主要过程如下。

(1)fill_population()：按照冲突图各节点的类型，为其随机分配颜色。

(2)stop_global_condition()：图成功着色，或者达到全局最大迭代次数。

(3)stop_local_search_condition()：图成功着色，或者达到局部最大迭代次数。

(4)choose_parents()：从 population 中随机选择两个 individual。

(5)crossover():交错选择两个父 individual 作为当前 individual,对节点的所有类型,每一类中的所有颜色,构造子 individual 的组成子集。选择当前 individual 中元素最多的子集,把其元素从两个父 individual 中删除。把此子集中的元素插入到子 individual 相应的子集中(初始时,子 individual 所有子集为空)。如果对所有类的所有颜色执行上述操作后,父 individual 中还有子集不为空,则把这些子集中的元素随机赋予子 individual 同类型的子集。一对 parents 仅生一个孩子。

(6)local_search():减少子 individual 的冲突,在下节阐述。

(7)update_population():用子 local_search()的结果替换掉 population 中冲突最多的一个元素。

(8)chaitin_assign_colors():即 Chaitin 算法的 select 阶段。

(9)spill():达到全局最大迭代次数后,依然有冲突,则选择所有类所有颜色子集中冲突数最多的元素,放到内存。

(10)Chaitin_Main():转到 Chaitin 算法的开始阶段,整个过程重新开始。

```
Hybrid_Main()
{
successful_flag = false;
fill_population();
if (check_population_conflicts())
        successful_flag = true;
while (not stop_global_condition())
{
        choose_parents();
        crossover();
        while (not stop_local_search_condition())
                local_search();
        if (not successful_flag)
                update_population();
}
if (successful_flag == true) {
        chaitin_assign_colors();
} else {
        spill();
        rewrite_program();
        Chaitin_Main();
}
```

图 3 遗传算法的主要流程

3.2.2 局部搜索

首先找出冲突图中冲突数最多的节点,在其类型的所有颜色中,重新赋予它已有颜色的下一个颜色,或者,当已有颜色是其类型的最大颜色时,赋予它同类型中最小的颜色。和禁忌搜索相比,这种简单的搜索方法节省了禁忌搜索中的 Tabu list 的开销。

4 实验结果和分析

我们的实验基于哈佛大学的 MachineSUIF[11]。MachineSUIF 的寄存器分配器实现了 George 和 Appel 的 Iterated Register Coalescing。我们实现了 Chaitin 和 Briggs 的寄存器分配算法、混合演化算法,用它们代替原来的寄存器分配器。目标体系结构是 x86。编译器框架的其它部分保持不变。

Benchmark 程序来自 SNU-RT benchmark suite[12],这是一组实时程序。对每个程序,用混合演化算法运行 5 次取平均结果。其参数设置为:population_size=50,max_global_generation=8000,max_local_search=20。各寄存器分配算法的比较如表 1 所示。

表 1 各寄存器分配算法对 benchmark 程序的 spill 次数比较

benchmarks	Chaitin		Briggs		George Appel		hybrid algorithm	
	loads		stores		loads		stores	
adpcm	410	320	182	133	180	121	126	98
fft1	155	114	43	17	42	16	25	10
fir	37	27	8	6	7	5	5	4
jfdctint	446	352	158	84	68	38	43	23
lms	3	3	3	3	3	3	3	3
matmul	59	46	20	10-	19	9	13	6
qsort	154	100	10	6	8	4	6	3

如表 1 所示,Briggs 的乐观图着色启发方法比 Chaitin 的悲观启发方法有着显著的改善,而 George 和 Appel 的算法和 Briggs 的算法结果相近,但前者对每个 benchark 程序(lms 除外)的 spill 次数都小于后者,尤其是 ifdctint。混合演化算法的 spill 次数明显小于 George&Appel 算法,减少约 30%,仅在 lms 程序所有算法的结果相同。

结论 嵌入式处理器的不规则体系结构特征给传统的图着色寄存器分配技术带来了新的困难。为满足嵌入式系统对编译生成的代码高质量的要求,本文在保持图着色直观性的前提下,提出了新的混合演化算法进行寄存器分配,取得了良好的效果。

今后的工作包括:此算法对其它嵌入式体系结构的应用,例如 ARM 处理器,DSP 等;提高算法实现的效率,减少编译时间。

参 考 文 献

1 Chaitin G J. Register allocation and spilling via graph coloring. In: Kathryn S. McKinley, ed. 20 Years of the ACM SIGPLAN Conference on Programming Language Design and Implementation 1979-1999, A Selection. ACM, 2004. 66～74

2 Briggs P, Cooper K, Torczon L. Improvements to graph coloring register allocation. ACM Transactions on Programming Languages and Systems, 1994, 16(3):428～455

3 George L, Appel A W. Iterated register coalescing. ACM Transactions on Programming Languages and Systems, 1996, 18(3): 300～324

4 Cooper K D, Simpson L T. Live range splitting in a graph coloring register allocator. In: Kai Koskimies, ed. Proceedings of the International Compiler Construction Conference, Lecture Notes in Computer Science 1383. Springer, 1998. 174～187

5 Briggs P, cooper K, Torczon L. Coloring register pairs. ACM Letters on Programming Languages and Systems, 1992, 1(1):3～13

6 Nickerson B R. Graph coloring register allocation for processors with multi-register operands. ACM SIGPLAN Conference on Programming Languages Design and Implementation, New York, 1990

7 Smith M D, Holloway G. Graph-coloring register allocation for irregular architectures:[Technical report]. Harvard University, 2000

8 Runeson J, Nystrom S. Retargetable graph-coloring register allocation for irregular architectures. In: Andreas Krall, ed. Proceedings of Software and Compilers for Embedded Systems, 7th International Workshop, LNCS 2826. Springer, 2003. 240～254

9 Smith M D, Ramsey N, Holloway G. A generalized algorithm for graph-coloring register allocation. In: William Pugh, Craig Chambers, eds. Proceedings of the ACM SIGPLAN Conference on Programming Languages Design and Implementation. ACM, 2004. 277～288

10 Goodwin D W, wilken K D. Optimal and near-optimal global register allocations using 0-1 integer programming. Software-Practice&Experience, 1996, 26(8):929～965

11 http://www.eecs.harvard.edu/machsuif

12 http://archi.snu.ac.kr/realtime/benchmark