一种基于GCC的VLIW编译器指令调度算法

赵贤鹏 李增智 宋涛 袁飞 冯元 屈科文
(西安交通大学计算机系系统结构与网络研究所 西安 710049)

摘 要：指令调度是编译优化过程中的重要技术。对于VLIW机器来讲，由于机器性能与编译器的设计和实现有很大的关系，指令调度就显得尤为重要。指令调度是在保证语义正确的前提下，改变指令执行的顺序，以提高指令级并行的程度。文章在一个DSP芯片C编译器上的工作基础上，介绍了一种行之有效的指令调度算法，并分析了算法的正确性。

关键词：指令调度，指令级并行(ILP)，VLIW，DAG图
中图法分类号：TP311 **文献标识码**：A **文章编号**：1000-7180(2004)01-0062-03

Scheduling Algorithm Used in a VLIW Compiler Based on GCC

ZHAO Xian-peng LI Zeng-zhi SONG Tao YUAN Fei FENG Yuan QU Ke-wen
(The Institute of Computer Architecture and Network of Xi'an Jiaotong University, Xi'an 710049 China)

Abstract: Instruction scheduling is a very important technology during the procedure of compiler optimization. It is especially important to VLIW architecture as for the performance of the machine has something to do with the design and realization of the compiler. To change the execution order of instructions, enchant ILP furthest, will achieve higher performance on the basis of ensuring correctness. This paper mainly discusses a useful scheduling algorithm and its correctness based on the work on a DSP C compiler.

Key words: Instruction scheduling, ILP, VLIW, DAG Graph

1 引言

根据软硬件分工的不同，计算机的体系结构可以分为以下几类：

串行结构：程序不需要明确指出任何并行信息，所有指令依赖关系的判断和调度工作由硬件完成，这大大增加了硬件设计的复杂度，同时也限制了指令级并行度的进一步提高。超标量机器是这种体系结构的代表，从某种程度上讲，向量机也具有以上特征。

独立结构：硬件不提供互锁机制，完全依赖编译器静态的分析程序中指令的依赖关系并制定执行顺序，这类结构的典型就是VLIW (Very Long Instruction Word，超长指令字)，它把传统上由硬件实现或部分实现的功能完全交给软件实现，因此简化了硬件的设计，使机器能达到较高的并行性能。

相关结构：由程序员或编译器发现并指出可执行程序中操作之间存在的依赖关系，而硬件根据这些信息实时地调度执行。这是串行结构和独立结构之间的一个折衷方式，典型的代表是数据流机。

VLIW机器在单个机器周期中同时发射并执行多个操作，从而提高机器的指令并行度，可以更大程度上利用硬件资源，提高VLIW机器的效率。

收稿日期：2003-06-06

2 问题分析

VLIW指的是一种指令集设计思想和技术，它利用编译器把若干简单的、相互之间无依赖的操作压缩到一个长的指令字中。VLIW体系结构是将水平微码和超标量处理两种普遍采用的概念相结合的产物。从指令系统上讲，VLIW的机器指令具有较长的机器指令字，机器指令字具有固定的格式，每条指令字中包含这一个或多个独立的字段，字段中的操作码被送往不同的功能部件；从机器结构上讲，VLIW的机器内部提供多个可以并发的功能部件，所有功能部件共享使用大型寄存器堆，CPU在一个时钟周期内可以发射多条指令。VLIW采用静态指令调度技术，其流水线是非保护的，没有用于防止资源冲突和隐藏流水线延时的硬件和互锁机制，完全依赖编译器静态的分析程序中指令的依赖关系，来决定指令是否可并行执行以及执行顺序。

在VLIW结构中，程序中所包含的ILP在编译阶段就能被发现和利用，通过对软件的设定，可以在更大的范围内搜索可以利用的ILP，这样比由硬件在运行时进行检查和调度更有效；VLIW中指令

的并行操作由编译器显式指定,这样就可以确保同一时间并行执行的操作不会超过机器提供的并行资源限度，这也就简化了指令译码和控制逻辑;这使得 VLIW 支持更高的 ILP,具有更好的并行性能。同样,VLIW 结构也存在明显的缺陷:VLIW 机器提供了超长指令字资源,但由于 VLIW 指令的格式固定或算子依赖等原因,VLIW 指令字可能会包含未用的域,从而导致代码空间和执行速度上的双重损失;由于算子之间的并行关系和执行顺序是由优化编译器静态指定的。因此,VLIW 的性能发挥很大程度上要依赖于编译器的设计和实现。

应用广泛的 GCC (GNU Compiler Collection)是支持多语言、多目标机编译系统中最具有代表性的一个,GCC 具有清晰的前端语法树结构、高度概括的抽象机中间语言、简洁有力的后端机器描述和开放式的整体结构,为实现多语言开发、多平台移植提供了有力的支持。从整体上讲,GCC 由语言预处理器和编译器 CCI 两部分构成,编译结束后,再利用汇编器 GAS 和链接器 LD 生成二进制形式的可执行程序或函数库。为了实现对多语言和多平台的支持。图 1 是 GCC 的工作流程。

通过对 GCC 编译器的分析，可以发现，由于 GCC 支持多语言、多平台的特性,GCC 对于特定体系结构的优化的支持并不充分，因此可以将 VLIW 机器上的优化编译分为两个层次，第一步是由 CCI 在 RTL 一级进行优化，生成优化后的汇编代码,然后再以该汇编代码为输入,根据硬件信息进行体系结构相关的优化,最后由汇编器生成可执行程序。

指令调度是开发目标机指令级并行性提供其性能的核心问题。所谓指令调度就是从顺序程序中识别出指令级可并行的成分,并利用这些可并行性合理安排指令的执行顺序,以达到最大限度的发挥目标机所提供的处理能力的目的。指令调度决定操作执行的相对顺序,各操作的具体执行时间及使用哪些硬件资源等等。下面将介绍在实际工作中用到的一种有效的指令调度算法。

3 相关概念

基本块的 DAG 图 $G=<V, A>$,若$(v1, v2)\in A$,那么弧$(v1, v2)$的权值 $w(v1, v2)$表示指令 $v2$ 必须至少在指令 $v1$ 执行完 $w(v1, v2)$个周期后才能被执行。对于 $G=(V,A)$中的任何一个顶点 $v\in V$,v 的直接前驱顶点集合为$\{n|n\in V$ 且$(n, v)\in A\}$,记为 $prev(v)$。对于 $G=(V, A)$中的任一顶点 $v\in V$,v 的直接后继顶点集合为$\{n|n\in V$ 且$(v, n)\in A\}$,记为 $succ(v)$。对于任何的 $v\in V$,顶点 v 的引入次数 $in(v)=|\{j|j\in V$ 并且$(j, v)\in A\}|$,即顶点的引入弧的条数。

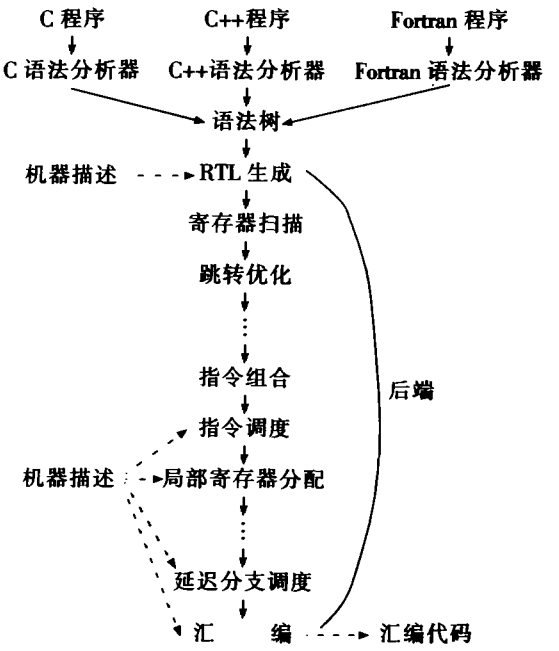

图 1 GCC 工作流程图

基本块的 DAG 图具有以下特性:

(1) G 中至少由一个顶点的引入次数为 0,显然,基本块的第一条指令不依赖于基本块中的任何指令。所有引入次数为 0 的顶点组成集合称为 root,root 一定不是空集。

(2) 若两个顶点 $v1, v2$ 有相同的直接前驱顶点集并且 $v1, v2$ 之间没有弧,则表示 $v1, v2$ 之间没有依赖关系,可以按照任何次序执行。

(3) 若基本块的指令序列为 $I_1, I_2, \cdots, I_n$,则对任意的 $1\leqslant k<j\leqslant n$,$G$ 中不存在弧(I_j, I_k),即指令 I_k 不可能在指令 I_j 执行完后执行。

4 算法描述

该算法以 GCC 优化方式输出的汇编代码为输入进行优化。首先对输入的汇编代码扫描并进行语法分析，划分基本块，针对每个基本块进行调度和优化。由于 DAG 图可以很好地描述指令之间的依赖关系，因此用 DAG 图描述基本块中指令间的依赖关系。在算法开始前用 schedule 表示待调度的指令，算法完成后，schedule 中存放的就是调度后的指令序列，可以根据此时 schedule 中的指令顺序来选择指令执行;用一个数组 starttime[ ]来记录序列中每个指令可以开始执行的时间，如果

schedule中的两条指令i和j有starttime [j]>startime [i]+1,那么需要在i和j中插入starttime[j]-starttime [i]-1个空操作。用一个buffer来存储中间结果。对算法的描述如下:

(1) Schedule schedule = 空序列;

(2) Schedule buffer;

(3) 对基本块的DAG图进行遍历,为每个顶点建立它的直接后继顶点集合;

(4) 将starttime[ ]数组初始化为0;

(5) 找出该DAG图中的root集合,将root中的所有顶点放入buffer中;

(6) while(buffer不为空);

(7) 将buffer中的顶点按其直接后继顶点可以开始的最小时间从大到小排序;

(8) 对buffer中的每个顶点i;

(9) 将i从buffer中删除并将其放入schedule中;

(10) 将i的每一个直接后继顶点j的in(j)减1;

(11) 对于j,它可以开始执行的时间为:

starttime[j]=max{startime[j],startime[i]+w(i,j)};

(12) 如果j的in(j)=0,那么就可以将j加入到buffer中。

5 算法正确性分析

对上述算法的正确性的证明如下:

设基本块的DAG图为$G=<V,A>$,因为V是一个有限集,并且算法对每个顶点最多遍历一次,所以算法一定会在有限的时间内结束。要保证算法的正确性,就是要保证以下两个问题:

(1) 调度过后,所有的指令都被调度;

(2) 调度过后,相互之间有依赖关系的指令的执行顺序正确。

假定算法执行完后未被调度的顶点组成集合$U\neq\Phi$,则已经调度的集合为$V-U$,且U和$V-U$一定是有限集。显然U中所有的顶点一定不在*root*中。若算法在将一个顶点i放入到$V-U$中,同时将i的输出弧从G中去掉。这样,算法执行完后将分成两个子图:U和$V-U$,而且两个子图间不存在边。对于任意顶点$u\in U$,有$in(u)\neq 0$,否则根据算法的12,8和9行,该顶点一定会被加入到$V-U$中。设U中的顶点在原指令序列中为$u_1,u_2,\cdots,u_k$,因为$in(u_1)\neq 0$,所以必存在弧(u_j,u_1),这与基本块DAG图特性(3)矛盾。所以$U=\Phi$,算法执行结束后,基本块中所有的指令都被调度。第一个问题得证。

若基本块中的指令i和j存在依赖关系,$i\in V$,$j\in V$且$(j,i)\in A$,设权值为$w(j,i)$,这样指令i和j在执行时应满足如下条件:

(1) 指令i在指令j的后面;

(2) $starttime[i]\geq starttime[j]+w(j,i)$。

即指令i至少应该在指令j执行完$w(j,i)$个周期后才能执行。根据算法的(10),(11)行有:$starttime[i]=\max(starttime[i], starttime[p]+w(p,i))$,其中$p$为prev$(i)$中的任意一个元素,所以条件(2)成立。若$j$为root中的一个元素,存在边$(j,i)$,因此$i$一定不在root中,根据算法的5,6,9行,可以得出j先被放入schedule中,条件(1)成立。否则,若j不在root中,根据算法的12行,指令i若被调度,i的引入次数必须为0,又因为根据算法8行和10行,若i的引入次数为0,则i所有前驱顶点都已经被调度,因为j是i的前驱顶点,所以j已经被调度,条件(1)也满足。第二个问题也得证。

综上所述,该调度算法是正确的。

6 结束语

GNU的GCC提供了多平台支持,但是对于特定机器的支持不足,本文以实际工作为基础,提出了一种针对DSP芯片VP6(Vision Processor)的指令调度算法。该算法是正确的,有效的,并且比较容易实现。该算法与GCC结合,可以得到效率较高的汇编代码。对于其他的VLIW的CPU,该算法也具有一定的适应性。

参考文献

[1] Richard Stallman. Linux online document GCC info.

[2] Wen-mei W Hwu. Compiler Technology for Future Microprocessors Proc. IEEE[C]. In, Center for Reliable and High-Performance computing, University of Illinois, 1995.

[3] 张嗣元, 晏海华, 王雷. 基于VLIW的机器相关优化编译技术研究. 计算机工程与应用, 2003, 2.

[4] 陆伯鹰, 尹宝林. 一个基于DAG图的指令调度优化算法. 计算机工程与应用, 2001, 12.

赵贤鹏 男,(1978-),硕士研究生。研究方向为网络管理及应用、网络安全技术、分布式系统。

李增智 男,(1938-),教授,博士生导师。研究方向为网络管理及应用、分布式系统、主动网络。