一种软件流水的反流水算法*

汤志忠[1+], 李文龙[1], 苏伯珙[2]

[1](清华大学 计算机科学与技术系,北京 100084)

[2](William Paterson 大学 计算机科学系,新泽西洲 07470,美国)

A De-Pipeline Algorithm for Software-Pipeline

TANG Zhi-Zhong[1+], LI Wen-Long[1], SU Bo-Gong[2]

[1](Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

[2](Department of Computer Science, William Paterson University, New Jersey 07470, USA)

+ Corresponding author: Phn: +86-10-62772213, E-mail: tzz-dcs@tsinghua.edu.cn, http://www.cs.tsinghua.edu.cn

Received 2003-09-24; Accepted 2004-03-02

Tang ZZ, Li WL, Su BG. A de-pipeline algorithm for software-pipeline. *Journal of Software*, 2004,15(7): 987~993.

http://www.jos.org.cn/1000-9825/15/987.htm

Abstract: Software pipelining is a loop optimization technique that has been widely implemented in modern optimizing compilers. In order to fully utilize the instruction level parallelism of the recent VLIW DSP processors, DSP programs have to be optimized by software pipelining. However, because of the transformation of the original sequential code, a software-pipelined loop is often difficult to understand, test, and debug. It is also very difficult to reuse and port a software-pipelined loop to other processors, especially when the original sequential code is unavailable. In this paper we propose a de-pipelining algorithm which converts the optimized assembly code of a software-pipelined loop back to a semantically equivalent sequential counterpart. Preliminary experiments on 20 programs verify the validity of the proposed de-pipelining algorithm.

Key words: software pipelining; de-pipelining; instruction level parallelism

摘 要: 软件流水是一种循环程序的优化技术,已经广泛应用于现代优化编译器中.为了充分利用 VLIW DSP 处理机的指令级并行性,必须使用软件流水技术对 DSP 程序进行优化.然而,在串行源代码不存在的情况下,对软件流水后的原始代码进行变换、理解、测试和调试,并转换成其他处理机的代码是非常困难的.提出了一种反流水技术,它能够将软件流水后的优化汇编代码反向转换成语义等价的相应代码.通过 20 个程序的初步实验,验证了所提出的反流水算法的正确性.

关键词: 软件流水;反流水;指令级并行

中图法分类号: TP338 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant No.60173010 (国家自然科学基金)

作者简介: 汤志忠(1946－),男,教授,博士生导师,主要研究领域为计算机系统结构,指令级并行算法,并行编译技术;李文龙(1977－),男,博士生,主要研究领域为指令级并行算法;苏伯珙(1939－),男,教授,主要研究领域为并行算法,优化编译技术.

在过去的几年中,数字信号处理机(digital signal processor,简称 DSP)发展迅速[1],由于对提高性能及解决大范围应用程序的持续需要,许多厂商推出了基于 VLIW(very long instruction word)的 DSP 处理机,为了充分利用这些 VLIW DSP 处理机的指令级并行性,DSP 程序一般都要经过软件流水的优化.

软件流水[2~4]通过重叠不同循环体的执行来加快循环程序在指令级并行处理机上的执行速度,这种技术已经研究多年,并在许多优化编译器中得以实现,例如 Texas Instruments 的 TIC6X 和 StarCore 的 SC140 等 DSP 处理机的编译器.

研究反软件流水技术的动机是:首先,由于对原始串行代码的变换,软件流水后的代码很难理解、测试及调试.其次,由于处理机之间指令兼容性的问题,经过软件流水后的循环代码很难在其他处理机中移植和使用.第三,虽然软件流水后的循环程序、CPU 的执行时间效益很好,但在存储器使用上,它的空间效益可能很差.在应用软件流水对循环进行优化时,并没有考虑存储空间对性能的影响.对于某些存储空间有限的应用,软件流水不适合.最后,我们注意到,大量的 DSP 应用程序已被软件流水优化过,并且被测试过,但是这些应用程序的串行源代码已经不存在了.

据我们所知,到目前为止,有关反软件流水算法的研究还很少,但是,这是一个应用前景非常好的研究领域,对于重新获取源代码和在不同处理机之间移植软件流水后的代码提供了可能.

本文提出了一种反软件流水算法.首先使用严格的时序关系来识别循环程序的核心代码,然后利用循环展开技术,构造软件流水循环的数据相关性图(DDG).最后,利用 DDG 来构造一个语义等价的串行循环程序.

本文首先通过一个例子来说明什么是软件流水和反软件流水,然后,具体介绍我们提出的反软件流水算法,接下去是 20 个典型程序的实验结果,最后是结论.

1 软件流水与反软件流水

图 1 是点积函数中的循环经过软件流水后的代码,用 TIC62 处理机的汇编语言[5]表示.

```
    MVK 0X32,B0
    MVC CSR,B8          ZERO A4             ZERO B5
    AND –2,B8,B4
    MVC B4,CSR          SUB B0,5,B0         MVK 0X2,A1
L1: [B0] B   L2
    LDH*++A0(4),A3      LDH*++B7(4),B4
    LDH*+A0(2),A3       LDH*+B7(2),B4       [B0] B   L2
    LDH*++A0(4),A3      LDH*++B7(4),B4
L2: LDH*+A0(2),A3       LDH*+B7(2),B4       [B0] B   L2         MPY B4,A3,B6    [!A1]ADD B6,B5,B5
    LDH*++A0(4),A3      LDH*++B7(4),B4      [B0]SUB B0,1,B0     MPY B4,A3,A5    [!A1]ADD A5,A4,A4   [A1]SUB A1,1,A1
L3: LDH*+A0(2),A3       LDH*+B7(2),B4       MPY B4,A3,B6        ADD B6,B5,B5
                                            MPY B4,A3,A5        ADD A5,A4,A4
                                            MPY B4,A3,B6        ADD B6,B5,B5
                                            MPY B4,A3,A5        ADD A5,A4,A4
                                            MPY B4,A3,B6        ADD B6,B5,B5
                                            MPY B4,A3,A5        ADD A5,A4,A4
                                                                ADD B6,B5,B5
                                                                ADD A5,A4,A4
```

Fig.1 Software pipelined loop of dot-product function in TIC62 assembly code

图 1 TIC62 汇编码表示的点积函数的软件流水循环结果

反软件流水是软件流水的逆向操作,是把已经软件流水的循环代码转换成语义等价的串行代码.对于图 1 中的软件流水循环代码,反软件流水是把它转换成如图 2 所示的语义等价的串行循环代码.

一般来说,一个软件流水后的循环程序由 3 部分组成:装入、核心和排空.与硬件流水线一样,软件流水的装入部分和排空部分只执行一次,核心部分要重复执行.在图 1 中,从 L1 到 L2 是装入部分,L2 到 L3 是核心部分,L3 后面的指令属于排空部分.在装入部分和核心部分存在着严格的时序关系.例如,如果标号 L1 中的指令在时刻 t 发射,那么位于 L1 和 L2 之间的另外 3 条指令一定要在时刻 t+1,t+2 和 t+3 发射.任何阻塞和延迟执行都会破坏程序的语义,传统的控制相关和数据相关图都不能表示这种严格的时序关系.

实际上,在反软件流水的结果出来之前,要理解软件流水循环的语义是非常困难的,连续循环体的折叠以及多个分支指令的延迟使得理解软件流水循环的控制流结构变得更加困难.

如图 1 所示的软件流水循环经过反软件流水之后,生成如图 2 所示的语义等价的串行代码,我们很容易理解图 2 的串行代码,关于代码的详细语义读者可以参照文献[5].

```
        MVK    0X32, B0
        ZERO   A4
        ZERO   B5
LE:     LDH          *++A0(4), A3
        LDH          *++B7(4), B4
        LDH          *+A0(2), A3
        LDH          *+B7(2), B4
       [B0]          SUB   B0, 1, B0
       [B0]          B     LE
        MPY          B4, A3, B6
        NOP
        MPY          B4, A3, A5
        ADD          B6, B5, B5
        ADD          A5, A4, A4
```

Fig.2 The semantically equivalent code of a software pipelined loop of dot product in TIC62 assembly code

图 2 TIC62 汇编码表示的与点积软件流水循环语义等价的串行代码

比较图 1 和图 2 这两段语义完全等价的代码,可以很容易看出,经过软件流水的循环,其时间效益更好,它的执行时间比串行代码要快 5 倍左右,而串行代码的空间效益好,它比软件流水循环节省了 3 倍的程序存储空间.另外,为了执行软件流水循环,必须有大量的功能部件和寄存器.因此,软件流水循环适合在类似于 VLIW 这种提供大量资源的处理机上运行.

2 反软件流水算法

下面结合图 3 的例子,解释我们所提出的反软件流水算法.图 3 给出的是点积函数中循环经过软件流水后的代码,采用 TIC62 处理机的汇编代码[5]表示.图中最左面一列是行号,符号"||"表明当前行的指令可以与上一行上的指令并行执行.

反软件流水算法的操作步骤如下:

第 1 步.循环检测.从给定的代码中,识别出软件流水的循环体,这包括循环的入口和循环体的长度等,识别的准则如下:

如果有一条向回跳转的分支指令,那么分支指令的目标地址就是循环的入口,循环体的长度等于向回跳转的分支指令与循环入口之间的距离再加上分支延迟槽的时间.

如果在循环入口上面的,且代码长度等于分支延迟时间加 1 的区域内包含有向前跳转的分支指令,则分支指令的目标地址就是循环的入口,软件流水循环体的长度等于最近的分支指令和循环入口之间的距离.

在图 3 中,L2 是向回跳转的分支指令的目标地址,因此 L2 就是软件流水循环的入口,同时,软件流水循环体的长度等于 2.

第 2 步.活变量分析.从一个给定的软件流水循环体中找出所有最后执行的指令.通常,活变量包含在最后执行的指令中.

最后执行的指令一般分为下面两种类型:往活变量寄存器中写数据的指令和写存储器的指令.具体寻找活变量的方法是:在循环区域中自下往上搜索所有最后执行的指令.

在图 3 中,Add B6,B5,B5 和 Add A5,A4,A4 是最后执行的指令,因此活变量为 B6 和 A5.

```
                MVK 0X32,B0
                ZERO A4
||              ZERO B5
                SUB B0,5,B0
||              MVK 0X2,A1
        [B0]    B   L2
                LDH *++A0(4),A3
||              LDH *++B7(4),B4
                LDH *+A0(2),A3
||              LDH *+B7(2),B4
||      [B0]    B   L2
                LDH *++A0(4),A3
||              LDH *++B7(4),B4
L2:             LDH *+A0(2),A3
||              LDH *+B7(2),B4
||      [B0]    B   L2
||              MPY B4,A3,B6
||      [!A1]   ADD B6,B5,B5
                LDH *++A0(4),A3
||              LDH *++B7(4),B4
||      [B0]    SUB B0,1,B0
||              MPY B4,A3,A5
||      [!A1]   ADD A5,A4,A4
||      [A1]    SUB A1,1,A1
                LDH *+A0(2),A3
||              LDH *+B7(2),B4
||              MPY B4,A3,B6
||              ADD B6,B5,B5
                MPY B4,A3,A5
||              ADD A5,A4,A4
                MPY B4,A3,B6
||              ADD B6,B5,B5
                MPY B4,A3,A5
||              ADD A5,A4,A4
                MPY B4,A3,B6
||              ADD B6,B5,B5
                MPY B4,A3,A5
||              ADD A5,A4,A4
                ADD B6,B5,B5
                ADD A5,A4,A4
```

Fig.3 A segment of TIC62 assembly code

图 3 TIC62 处理机的汇编代码片段

第 3 步.数据相关性图的构造.构造软件流水循环的数据相关性图(DDG).

首先,展开循环体 $k-1$ 次,k 等于循环体的指令组中最多指令的个数.指令组是指可以并行执行的一组指令,在图 3 的代码段中,由||分隔的指令属于同一个指令组.其次,从第 2 步找到的最后执行的指令开始,使用高度优先的算法以自下向上的方式构造数据相关性.

图 4 是重新构造的点积函数中软件流水循环的数据相关性图.

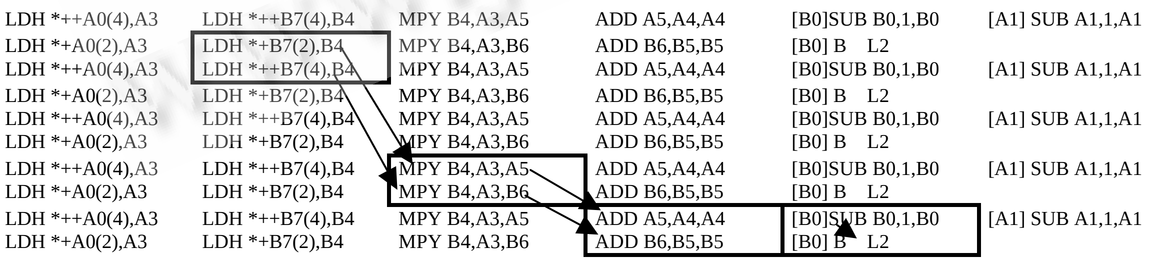

Fig.4 Data dependence graph (DDG) of running example

图 4 运行例子的数据相关性图

第 4 步.软件流水循环的检查.检查一个循环是否经过软件流水.

如果在循环体中存在两条指令 I_i 和 I_j,它们在循环体中的距离小于在数据相关性图中的距离,那么可以说这个循环体是经过软件流水处理的.

比较图 3 和图 4,可以确认,图 3 中所识别出的循环是经过软件流水的循环.

第 5 步.找出装入和排空部分.在给定的代码段中找出软件流水循环的装入部分和排空部分.

(1) 装入部分的查找:从循环的入口开始向上搜索,直到软件流水循环的上边界,在搜索过程中,找出所有包含循环体中指令的指令组,其中最上面的那个指令组是装入部分的上边界.

(2) 排空部分的查找:从软件流水的循环体底部开始向下搜索,直到软件流水循环的下边界,在搜索的同时找出所有包含循环体中指令的指令组,其中最下面的指令组是排空部分的下边界.

在图 3 中,装入部分是编号 6~13,排空部分是编号 25~40.

第 6 步.重新调度.将 DDG 转换为串行代码.

(1) 在从上述第(2)步找到的最后执行的指令中,自下而上利用列表调度法排列 DDG 上关键路径的偏序列表.为了满足所有指令的延迟时间,必要时可以插入 NOP 操作.

(2) 将所有在非关键路径上的指令插入到关键路径中.

(3) 删除装入和排空部分中所有包含在循环体中的指令.

第 7 步.循环次数计算.计算出软件流水循环串行代码的执行次数.

除了要考虑在给定的软件流水循环中所设置的执行次数初始值以外,还必须考虑其他诸多因素,如在装入部分所执行的循环次数,其值等于装入部分中包含的转移目标是循环入口的分支指令个数.另外,在排空部分最后执行的指令条数以及在给定的软件流水循环体中,向回跳转的分支指令与修改循环执行次数指令的相关位置等都影响着串行代码的执行次数.

从第(6)步~第(7)步,我们得到了如图 2 所示的点积函数的串行代码,与图 3 中给定的循环在语义上是等价的.

3 实验结果

利用我们所提出的反软件流水算法进行了 20 个程序的实验,这些代码属于不同的应用程序,循环的长度各不相同,装入和排空部分也各不相同,这些程序均从 TMS320C62x/C62x 的用户编程手册中获取,感兴趣的读者可以参照文献[5]获取相关信息.实验中,我们使用 TIC62 编译器或者手工线性汇编器[5]生成代码段.首先,我们使用反流水技术将这些代码转换成串行代码,接着使用 TIC62 的模拟器来运行原始的汇编代码和转换后的汇编代码.所有的计算结果表明,对于这些程序而言,我们的反流水算法是正确的.表 1 概括了这些软件流水循环的特点以及 20 个程序的反流水结果,其中 Sub 是指修改循环执行次数的指令,branch 是指跳到循环入口的分支指令.注意,其中的“1”表示由编译器生成,“2”表示由线性汇编器生成.

对于测试的每个程序,因为其软件流水循环的特点各不相同,经过反软件流水后,其执行次数、循环体长度都发生了变化,以第 1 个测试程序 dot product_1 为例,因为有显示的装入和排空,以及初始设置的循环执行次数,所以,反流水之后其执行次数为 50,循环体长度为 14.

Table 1 Experimental result

表 1 实验结果

Assembly code	Characteristics	Software-Pipelined loop		De-Pipelining result	
		Initial count	Body length	Count value	Body length
Dot product_1[2]	Normal	43	1	50	14
Dot product_2[2]	No postlude	50	1	50	14
Dot product_3[2]	Sub & branch in prelude only, no postlude	57	1	50	14
Dot product_4[2]	Branch in prelude only, no postlude	51	1	5	14
Dot product_5[1]	Normal	50	1	50	14
FIR[1]	No postlude	32	3	32	15
FIRnorld[1]	No postlude	16	2	16	15
IIR[2]	No postlude	100	4	100	13
Codebook[2]	No postlude,conditional branch in loop body	32	2	32	9
Vec_mpy[1]	Normal	75	3	75	16
Latsynth[1]	Normal	200	5	200	25
WVS[2]	Normal	49	2	50	16
Add_test[1]	No postlude	5	2	5	6
Loop_test_1[1]	Branch in prelude only	50	2	50	6
Loop_test_2[1]	Branch in prelude only	100	2	100	7
Loop_test_3[1]	Branch in prelude only	100	3	100	7
Loop_test_4[1]	Normal	50	5	50	11
Loop_test_8[1]	No prelude	20	9	20	21
Loop_test_12[1]	No prelude	25	23	25	27
Loop_test_16[1]	No prelude	50	17	50	35

4 相关工作和结论

在软件流水的研究领域,文献[6,7]给出的软件流水算法已经研究了多年,并且在很多的优化产品编译器中得以实现.而对于反编译循环软件流水的算法,目前反编译主要是将汇编代码或者二进制代码反编译成源程序,比如在软件工程领域中[8],将较低级的可执行代码转换成更容易理解的高级代码,以及在 Java 应用方面,将字节码转换成源程序[9]等.

我们给出了反软件流水算法及其实验结果.对于 DSP 用户,我们的算法对于理解和调试经过软件流水后的代码是一个非常有用的工具.进一步地,我们提出的反软件流水算法可以进一步扩展用于解决代码兼容性问题,包括 VLIW 体系结构的软件流水循环.虽然可以使用动态重调度[10]来解决兼容性问题,但是它不能解决包含软件流水循环的代码.通过使用反软件流水技术,可以将软件流水循环的代码从一个源 VLIW 处理机转换为一系列语义等价的中间代码级的串行代码,然后这些中间代码可以送给目标 VLIW 处理机的编译器.我们的代码适合于从一种 VLIW DSP 处理机到其他 DSP 处理机[6]汇编代码的转换.

本文的方法主要针对不带分支的、最内层循环软件流水后的代码.如何扩展本文的方法用于处理带分支及多重循环软件流水[11]后的代码是我们进一步的研究工作.

References:

[1] Strauss W. Digital signal processing: The new semiconductor industry technology driver. IEEE Signal Processing Magazine, 2000,17(2):52~56.

[2] Rau R, Fisher J. Instruction-Level parallel processing: History, overview, perspective. Technical Report, HPL-92-132, HP Labs, 1992.

[3] Su B, Ding S, Xia J. URPR—An extension of URCR for software pipelining. In: Daniel D, ed. ACM SIGMICRO Newsletter. New York: ACM Press, 1986. 94~103.

[4] Wang J, Eisenbeis C. Decomposed software pipelining: A new approach to exploit instruction level parallelism for loop programs. In: Michael C, ed. Proc. of the Architectures and Compilation Techniques for Fine and Medium Grain Parallelism. 1993. 3~14.

[5] TMS320C62x/C67x Programmer's guide. Texas Instrument Product Documentation, 1999.

[6] Rau B. Iterative modulo scheduling: An algorithm for software pipelining loops. In: ACM SIGMICRO Newsletter. California: ACM Press, 1994. 63~74.

[7] Richard AH. Lifetime-Sensitive modulo scheduling. In: Budd TA, ed. ACM SIGPLAN Notice. ACM Press, 1993. 258~267.

[8] Cristina C. An environment for the reverse engineering of executable programs. In: Dennis W, ed. APSEC. Washington: IEEE Computer Society, 1995. 410.

[9] Jerome M, Laurie H. Decompiling Java using staged encapsulation. In: Thomas E, ed. WCRE. Washington: IEEE Computer Society, 2001. 368.

[10] Conte T, Sathaye S. Optimization of VLIW compatibility systems employing dynamic rescheduling. Journal of Parallel Programming, 1997,25(2):83~112.

[11] Rong H, Tang Z, Gov Indarajan R, Alban D, Gao G. Single-Dimension software pipelining for multi-dimensional loops. In: Proc. of the Int'l Symp. on Code Generation and Optimization with Special Emphasis on Feedback-Directed and Runtime Optimization. IEEE Computer Society, 2004. 163~174.

****

第 13 届全国信息存储技术学术会议

征 文 通 知

为促进和加强存储技术的学术交流和产品展示，中国计算机学会信息存储技术专业委员会决定于 2004 年 10 月中旬在西安召开第 13 届全国信息存储技术学术会议。本次会议由中国计算机学会信息存储技术专业委员会主办，西北工业大学计算机学院承办。会议将通过学术报告、专题讨论、产品展示等多种形式，就信息存储的最新研究进展和发展趋势开展深入、广泛的学术交流，并特邀著名专家学者作专题报告。

一、 征文范围

欢迎从事信息技术研究、开发、应用的各界人士，就下列领域(但不限于)所涉及的信息存储技术方面的内容踊跃来稿：国内外存储技术的发展现状及趋势、信息存储理论研究与信息存储新技术研究；计算机主存体系结构研究及实现、海量信息存储技术、网络存储技术；存储领域中的核心技术及实现研究、存储相关芯片的设计与应用、智能存储技术；多媒体信息存储技术、数据仓库、数据挖掘；信息存储系统的安全性和可靠性;存储系统解决方案、存储技术及产品的标准。

二、 征文要求

应征学术论文应是未正式发表过的研究成果，字数(含中英文摘要、关键字与参考文献)不超过 8000 字。请注明作者的通信地址、邮政编码、电话和 E-mail 地址。论文格式请参照《计算机研究与发展》格式。

投稿方式：电子投稿。稿件请采用 Word、PDF 文档格式。

电子投稿的 E-mail 地址: zhangyy@nwpu.edu.cn

征文截止: 2004 年 07 月 15 日

录用通知: 2004 年 08 月 31 日

被本次会议录用的学术论文将收录在会议论文集内，优秀论文将在《计算机研究与发展》(增刊)上发表。有关会议的动态信息可通过拨打(029)8493772 或通过 E-mail 询问。

三、参展范围

为了加强产业界、学术界和应用领域间的交流和联系，本次会议将举办信息存储技术交流会及产品展示会。凡与存储相关的产品和技术均欢迎参会参展，有意参展的单位请联系：

联系人：张延园 教授 电话：(029)8493772 传真：(029) 8495772 E-mail: zhangyy@nwpu.edu.cn

四、中国计算机学会信息存储技术专业委员会联系方式

联系人：方粮 博士 通讯地址： 410073 湖南长沙国防科技大学计算机学院

电话： (0731)4575962-801 传真： (0731)4510109 E-mail: Lfang@163.net