第34卷 第16期 计 算 机 工 程 2008年8月
Vol.34 No.16 Computer Engineering August 2008
·软件技术与数据库· 文章编号：1000—3428(2008)16—0044—03 文献标识码：A 中图分类号：TP311

IA-64二进制翻译中软件流水代码消除技术

汪 淼[1,2]，赵荣彩[1]，蔡国明[3]
(1. 解放军信息工程大学信息工程学院，郑州 450002；2. 解放军信息工程大学理学院，郑州 450001；
3. 解放军信息工程大学电子技术学院，郑州 450004)

摘 要：IA-64体系结构使用软件流水提高程序的执行性能，但产生的二进制代码跟机器特性紧密相关，给代码跨平台移植造成了困难。该文针对IA-64体系结构下软件流水的特点，提出2种软件流水代码消除方法，它能够将软件流水代码转换成语义等价无硬件依赖的串行代码，实验验证了这2种方法的有效性。

关键词：二进制翻译；软件流水；语义映射；反软件流水

Elimination Technology of Software Pipelined Codes in IA-64 Binary Translation

WANG Miao[1,2], ZHAO Rong-cai[1], CAI Guo-ming[3]
(1. Institute of Information Engineering, PLA Information Engineering University, Zhengzhou 450002; 2. Institute of Sciences, PLA Information Engineering University, Zhengzhou 450001; 3. Institute of Electronic Technology, PLA Information Engineering University, Zhengzhou 450004)

【Abstract】Software pipelining is used to improve the performance of programs in IA-64, but the codes generated are relevant to the hardware plat, which makes them difficult to be migrated between different platforms. This paper analyzes the software-pipelined mechanism of IA-64, and presents two methods to eliminate software pipelined codes from optimized IA-64 executables. These two methods can convert the software pipelined codes into semantically equivalent sequential counterparts without dependence on the hardware. Experiments verify the validity of the proposed methods.

【Key words】binary translation; software pipelining; semantic mapping; software de-pipelining

1 概述

二进制翻译是软件逆向工程领域的一个分支，它将一种体系结构的二进制代码翻译成另一种体系结构的二进制代码，从而实现不同平台间的软件移植。本质上，二进制翻译的主要问题，来源于要处理的多平台间的特性差异。例如，源机器支持软件流水优化，提供旋转寄存器、谓词执行的硬件支持，而目标机不支持软件流水优化，没有相应的硬件支持。源机器和目标机的这种差异给二进制翻译造成了很大的困难。

笔者参与研制的静态二进制翻译系统I2A，是将IA-64体系结构上的可执行程序翻译为Alpha体系结构上的可执行程序。IA-64体系结构是Intel与HP联合开发的新一代体系结构，它引入了旋转寄存器、谓词执行、专用软件流水分支指令等新的特性，并结合投机技术，为实现软件流水提供了很好的支持，同时使该平台下的软件流水代码具有自己的特点，不能被直接移植到Alpha中。因此，在翻译经过高级优化的IA-64可执行文件时，需要先消除其中的软件流水代码。

关于软件流水代码消除技术，尤其是IA-64体系结构下的软件流水代码消除技术，目前研究甚少。文献[1-3]提出了一种反软件流水算法，但它仅适用于VLIW DSP处理机上的软件流水代码。由于IA-64处理机与TIC62处理机的流水代码存在本质的区别，因此该算法不适用于IA-64体系结构。本文针对IA-64体系结构提出了2种软件流水代码消除方法。

2 关键技术

2.1 软件流水

软件流水[4]是一种循环程序的优化技术，它通过并行执行来自不同循环体的指令来加快循环程序的执行速度。在软件流水算法中，模调度是一类重要的启发式算法。它对一个循环体中的指令进行调度，使得相继的循环体在以固定的启动间距(Initiation Interval, II)开始执行时，不发生资源冲突和相关冲突。模调度由装入(prolog)、核心(kernel)和排空(epilog)3个阶段组成。在装入阶段，循环体依次启动，直到出现一个重复模式(即核心)为止。核心阶段以II为周期反复执行，直到所有的循环体都被启动为止。排空阶段完成已经启动但未执行完毕的循环体。

2.2 旋转寄存器

IA-64为软件流水提供了旋转寄存器的硬件机制，使得编译器不必显式地进行寄存器重命名。在IA-64中，指令指定的是逻辑寄存器编号，其实际要访问的物理寄存器编号等于逻辑寄存器编号加上寄存器重命名基寄存器(*rrb*)的值。*rrb*的值由IA-64中的软件流水分支指令控制。每执行一条软件流水分支指令，*rrb*减1，使得循环体I中逻辑寄存器*X*的内容，在循环体I+1中移到了逻辑寄存器*X*+1中。在IA-64体

基金项目：国家“863”计划基金资助项目(2006AA01Z408)
作者简介：汪 淼(1977－)，女，博士研究生，主研方向：计算机软件与理论；赵荣彩，教授、博士生导师；蔡国明，博士研究生
收稿日期：2007-09-20 **E-mail：**yi_sanshengshi@sohu.com

系结构中，通用寄存器 r32～r137，浮点寄存器 f32～f127、谓词寄存器 p16～p63 都可以旋转。

2.3 谓词执行机制

IA-64 利用谓词来控制指令的条件执行。条件执行是指为语句增加一个限定谓词，运行时根据限定谓词的真假值来决定其是否执行。根据谓词的作用，可将它分为 2 类：条件谓词和阶段谓词。条件谓词用于控制条件分支的执行。阶段谓词用于软件流水循环中控制不同流水阶段的执行。软件流水代码被划分成若干个阶段。同属于一个阶段的指令分配一个特定的阶段谓词。通过阶段谓词值为 1 或 0 来控制各个阶段的执行。每执行一次软件流水循环体，阶段谓词就发生旋转。

2.4 IA-64 下软件流水的实现原理

根据循环判断条件的不同，软件流水循环可分为 2 类：

(1)count 型循环通过一个计数器来控制循环的执行。这类循环可以事先确定循环次数。

(2)while 型循环通过判断某个条件是否满足来决定是否执行循环。循环次数不能事先确定。

IA-64 提供了专用的软件流水分支指令[5]：br.ctop/br.cexit用于实现count型循环，如图 1 所示，br.wtop /br.wexit用于实现while型循环，如图 2 所示。

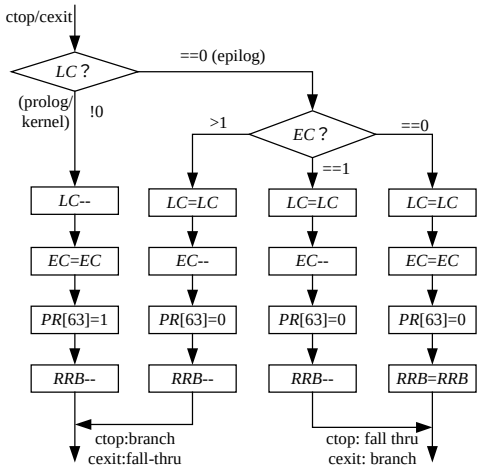


图 1　ctop/cexit 执行流程

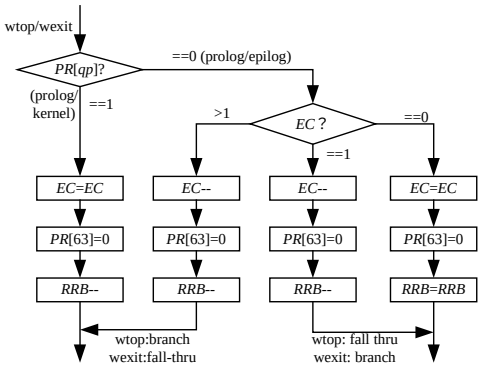


图 2　wtop/wexit 执行流程

这 2 种循环在实现上有以下不同点：count 型循环利用 ar.lc 寄存器控制装入和排空阶段的执行，其中，ar.lc 初始赋值为循环次数-1，每执行一次循环，递减 1；通过 ar.ec 控制排空阶段的执行。每个迭代被划分为若干个阶段，p16 是第 1 个阶段的谓词，p17 代表第 2 个阶段的谓词……。while 型循环使用分支谓词来决定软件流水分支指令的行为。在核心和排空阶段，分支谓词分别为 1 和 0。在装入阶段，分支谓词值可能为 1 或者 0。

3　消除软件流水特性的方法

针对 IA-64 中软件流水代码的特点，本文提出了 2 种用来消除软件流水特性的方法：直接语义映射方法和反软件流水方法。

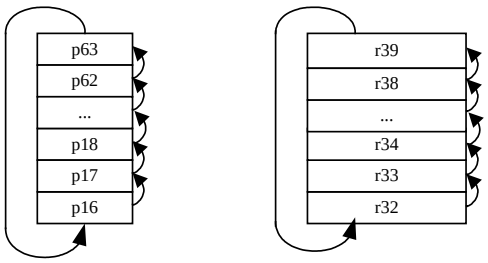
3.1 直接语义映射方法

直接语义映射方法的思想是完全映射软件流水分支指令的语义。这种方法不需要改变指令的顺序和恢复旋转寄存器，只需依据每种软件流水分支指令的语义，将其映射为对应的 C 语句。

例如：br.ctop 的语义映射为

```
if(lc> 0){lc--; p63=1;    RRB--; goto Lbegin;}
else if ( ec>1) {ec--; p63=0; RRB--; goto Lbegin;}
else if ( ec==1) {ec--; p63=0; RRB--;}
 else if (ec==0) {p63=0;}
```

这种方法要解决的关键问题是如何用 C 语句模拟寄存器的旋转。在 IA-64 中，可旋转的寄存器包括谓词寄存器、通用寄存器和浮点寄存器，如图 3 所示。



(a)谓词寄存器的旋转　　(b)8个通用寄存器的旋转

图 3　寄存器的旋转

(1)谓词寄存器的旋转

IA-64 提供了由 64 个 1 bit 谓词寄存器 p0~p63 组成的谓词寄存器组。这 64 个谓词寄存器既可以独立地作为 1 bit 寄存器使用，又可以用 64 bit 的向量进行整体的赋值和使用。基于谓词寄存器组的使用特点，使用如下的数据结构来表示谓词寄存器组。

```
struct prregbit{
unsigned int    p0:1;
unsigned int    p1:1;
…
unsigned int    p63:1;};
union _prstruct{
prregbit i;
int64 a;}pr;
```

使用这种表示方法，谓词寄存器的旋转可以用左移 1 bit 来表示：pr.a <<=1。

(2)通用寄存器和浮点寄存器的旋转

在 IA64 中，通用寄存器可分为 2 堆：静态寄存器堆和旋转(堆栈)寄存器堆。其中静态寄存器的个数是固定的，而旋转寄存器的个数是可调的。当进行软件流水时，可以用 alloc 指令从 96 个堆栈寄存器中分配 8 的整数倍个数的寄存器用作旋转寄存器。alloc 指令形式如下：

alloc r1=ar.pfs, ins, locals,outs,rots(rots 表示分配的旋转寄存器个数)

例如：指令 alloc r40=ar.pfs,45,44,8 表示有 8 个旋转寄存器，Gr32~Gr39。

本文使用赋值语句模拟通用寄存器的旋转。首先根据 alloc 指令获得旋转寄存器的个数。然后使用赋值语句从后到前依次将 r[*x*]寄存器的值赋给 r[*x*+1]寄存器。最后一个旋转寄存器的值赋给 r32。

浮点寄存器的处理与通用寄存器的处理类似。只是浮点旋转寄存器的个数是固定的。

由于实际使用的旋转寄存器总是小于可使用的旋转寄存器个数，因此为了减少赋值语句的个数，可先遍历软件流水循环体代码，获得实际使用的旋转寄存器个数后再进行处理。

3.2 反软件流水

反软件流水是软件流水的逆向操作，是把已经过软件流水优化的循环代码消除软件流水特性，转换成语义等价的串行代码。反软件流水算法的主要思想是：根据阶段谓词和比较谓词，调整软件流水循环代码的指令顺序，然后根据旋转寄存器的实现特征，调整旋转寄存器号，最后消除阶段谓词。

反软件流水算法的步骤是：

(1)识别软件流水循环。

(2)获取循环相关信息。主要指 2 个应用寄存器 ar.lc, ar.ec 的值、分配使用的旋转寄存器的个数和阶段谓词的初始值。

(3)指令重排序。根据指令所处的阶段，重新排列指令在代码中的位置。

(4)反旋转。恢复被动态重命名的寄存器号。

(5)处理归约变量。

(6)删除阶段谓词。

(7)计算 liveout 变量，并调整循环体外的相应寄存器号。

(8)产生相应的代码。

3.3 比较

直接语义映射方法实现简单，它无须改变循环体的指令顺序和调整寄存器号，仅对流水分支指令进行处理，适用于 count 型和 while 型循环。但这种方法增加了大量的基本块和赋值语句，生成的目标代码膨胀率较大。反软件流水方法生成的目标代码膨胀率较小，但算法实现复杂，需要进行数据流分析、调整指令顺序和恢复寄存器号，如果软件流水循环中含有条件分支或应用了投机优化，指令调整和恢复寄存器号会更加复杂。

4 实验结果

直接语义映射方法和反软件流水算法是在静态二进制翻译系统 I2A 中实现的。实验环境是一台拥有 4 个 1.3 GHz 的 Intel Itanium2 CPU 、内存为 2.0 GB 的服务器和一台 Alpha DS20E 666 MHz(×2)的服务器。实验中，将 C 程序在 IA-64 上编译成可执行的二进制文件，优化级别为 O2。然后使用本文的二进制翻译系统将可执行文件转换成低级 C，将低级 C 在目标机 Alpha 上用本地编译器编译得到新的可执行文件。最后在 Alpha 上运行此可执行文件。若翻译前后程序的运行结果相同，说明生成的目标代码是正确的，从而证明了软件流水代码消除技术的正确性。

表 1 为直接语义映射方法的实验结果。测试用例来自 SPECint-2000 基准测试集中的 8 个例子。表中列出了每个例子中出现的软件流水分支指令个数以及代码膨胀率和执行时间比。其中，代码膨胀率是指翻译后与翻译前的代码量大小的比率。执行时间比是翻译后的运行时间与翻译前的程序在 IA-64 平台上执行时间的比率。从实验结果可以看出翻译后的代码有明显的膨胀。

表 1 直接语义映射方法的实验结果

程序	ctop	Cexit	wtop	wexit	代码膨胀率	执行时间比
Bzip2	20	0	27	1	4.47	2.49
Gzip	11	0	32	0	5.25	2.73
Mcf	3	0	16	0	7.88	2.30
Crafty	28	0	27	2	3.39	2.65
Parser	9	0	63	10	2.24	2.04
Vortex	3	0	29	0	2.20	4.33
Gap	86	0	429	93	1.52	3.81
Twolf	103	0	179	3	1.23	6.52

另外，针对 count 型软件流水循环实现了反软件流水算法，分别统计了应用反软件流水算法及应用直接语义映射方法生成的控制流图在基本块个数和指令数减少的情况，如表 2 所示。实验结果表明，应用反软件流水算法可以明显提高二进制翻译生成的中间代码的质量：对于每条 ctop 指令，平均减少基本块 5.6 个、指令 30 条。

表 2 count 型循环反软件流水的实验结果

测试用例	减少的基本块个数	减少的指令条数
Bzip2	121	600
Gzip	55	567
Mcf	18	126
Crafty	140	669
Parser	45	108
Vortex	21	126
Gap	430	3 840
Twolf	674	6 831

5 结束语

IA-64 体系结构中的旋转寄存器、谓词执行和专用软件流水分支指令等特性为软件流水提供了很好的支持，本文根据 IA-64 体系结构的特点及软件流水的实现原理，提出了 2 种消除软件流水代码的方法：直接语义映射方法和反软件流水算法，并在静态二进制编译系统 I2A 中进行了实现。实验结果表明，直接语义映射方法适用于消除 count 型和 while 型软件流水代码，而反软件流水算法目前仅适用于循环次数固定的 count 型软件流水循环。下一步将将研究如何扩展反软件流水算法，使其用于循环次数不固定的 while 型软件流水循环。

参考文献

[1] Su Bogong, Wang Jim, Wang Erhwen, et al. De-pipeline a Software-pipelined Loop[C]//Proc. of ICASSP'03. New Jersey, NJ USA: [s. n.], 2003.

[2] Manzano J. Software-pipelined Loop Between Two VLIW DSP Processors[C]//Proc. of the International Signal Processing Conference. [S. l.]: IEEE Press, 2003.

[3] Su Bogong, Wang Jim, Wang Erhwen, et al. Software De-pipelining Technique[R]. New Jersey, NJ, USA: University of New Jersey, 2004: 7-16.

[4] Allen V H, Jones R B, Lee R M, et al. Software Pipelining[J]. ACM Computing Surveys, 1995, 27(3): 367-432.

[5] Intel Corporation. Intel Itanium™ Architecture Software Developer's Manual(Volume 1): Application Architecture[Z]. 2001.