第32卷 第23期 计 算 机 工 程 2006年12月
Vol.32 No.23 Computer Engineering December 2006
·软件技术与数据库· 文章编号：1000—3428(2006)23—0049—03 文献标识码：A 中图分类号：TP314

IA-64 代码翻译中的跳转表恢复技术

齐 宁，赵荣彩
（解放军信息工程大学计算机系，郑州 450002）

摘 要：在对 IA-64 二进制代码进行翻译的过程中，一个重要的问题就是识别和恢复通过跳转表实现的 switch 语句。分析了编译器生成跳转表时采用的策略，提出了前向预取同反向切片及表达式置换相结合以识别和恢复 switch 语句的技术，归纳了用于获取跳转表地址的规格化形式，给出了跳转表分支目标地址恢复方法，从而可将包含跳转表的 IA-64 二进制代码恢复为高级 switch 语句。该技术已经在二进制翻译框架 I2A 上进行了测试，可以处理编译器 gcc 2.96 及 gcc 3.2.3 在多种优化级别下生成的 IA-64 代码。

关键词：IA-64；跳转表；切片；表达式置换

Jump Table Recovery Technique in IA-64 Binary Code Translation

QI Ning, ZHAO Rongcai
(Department of Computer, PLA Information and Engineering University, Zhengzhou 450002)

【Abstract】 In the translation of IA-64 binary code, one of the important problems is that of recognizing the switch statements implemented via jump table. The paper first analyzes the compiler's strategies when creating jump table, then presents a technique for recognizing and recovering switch statements by combination of forward prefetch and backwards slicing plus expression substitution, reduces to a normal form which allows people to determine where the jump table is located, proposes a method to recovery the target addresses of jump table. Using such a technique, it can translate the IA-64 binary code that contains jump table into high level switch statements. The presented technique is tested in a binary translation framework I2A. The test shows that the technique is suitable for IA-64 binary code generated by compiler gcc 2.96 and gcc 3.2.3 with multi optimization options.

【Key words】 IA-64; Jump table; Slicing; Expression substitution

在冯·诺依曼体系结构下，指令和数据均采用相同的方式表示，在进行解码时，一个基本问题就是对指令和数据进行区分。给定一个可执行程序，程序的入口点可以在程序的头部找到，代码段和数据段的信息也在头部给出。然而，数据也可以保存在代码段中，而且这些信息并未保存在可执行程序的头部。因此在对二进制代码进行分析的过程中，必须在代码段中将数据分离出来。

传统的方法是从可执行程序入口点开始，沿着所有可到达路径进行解码，该方法的缺点在于当存在索引跳转或者间接跳转时不能覆盖全部代码空间[1]。

为了解决这一问题，本文在对 gcc 编译器生成跳转表的策略分析基础上，研究了跳转表目标地址的恢复技术，从而可以从包含跳转表的 IA-64 二进制代码恢复为高级 switch 语句。该技术应用在二进制翻译框架 I2A(IA-64 to alpha)中，结果证明对多版本 gcc 编译器在多优化级别下生成的 IA-64 二进制代码均有效。

1 背景及相关知识概要

1.1 I2A 二进制翻译框架

I2A 翻译系统的设计目标是将 IA-64 体系结构下的 elf64 文件格式的二进制可执行文件转换成 Alpha 机器的可执行文件，这里假设源二进制文件是从 c 语言经过编译的到的 elf64 文件，编译器为 gcc 2.96 和 gcc 3.2.3。

在I2A翻译系统中，前端提供了二进制文件解码器，从输入的IA-64 源可执行文件中读入指令流。然后使用解码器将该指令流解码成源机器指令序列。语义映射器基于源机器的语义说明将每条源机器指令翻译成第 1 级中间表示RTL。RTL到HRTL翻译器应用控制转换信息、过程约定以及其他的信息对前面所产生的RTL流进行分析，去除同机器相关的部分，得到第 2 级中间表示HRTL[2]。接下来使用中间语言转换器将HRTL转换为低级C，最后通过目标机器上的优化编译器将低级C程序编译为Alpha可执行程序。

1.2 IA-64 体系结构

IA-64 体系结构是一种全新的 64 位指令集体系结构(ISA)，应用了名为显式并行指令计算(Explicitly Parallel Instruction Computing, EPIC)的处理器体系结构技术，同时兼容IA-32 指令集。其主要特点包括显式并行、投机、谓词执行、低开销的循环软件流水、分支预测等[3]。该体系结构定义了指令束(Instruction Bundle)的概念，每个指令束由 3 条 41 位二进制指令和一个 5 位的指令束模板区域组成。

2 跳转表生成策略

switch 语句是一种 n-条件分支语句，使得程序可以在代码中选择 n 个分支之一来执行。switch 语句通过将索引变量的值同分支标号的值进行比较来选择分支，同时允许使用一个缺省的分支。不同版本的 gcc 编译器在为 switch 语句生成

基金项目：国防科研基金资助重点项目
作者简介：齐 宁(1978－)，男，博士生，主研方向：逆向工程，先进编译技术；赵荣彩，教授、博导
收稿日期：2006-02-21 **E-mail：**qi_ning@126.com

跳转表时采用了不同的策略，以如下所示的 switch 语句为例：

```
#include <stdio.h>
int main()
  {int num;
   printf(“input a number:\n”);
   scanf(“%d”,&num);
   switch(num){
       case 2:printf(“Two!\n”);break;
       …
       case 7: printf(“Seven!\n”);break;
       default: printf(“Other!\n”);break;
      }
 return 0;
  }
```

编译器 gcc 2.96 在为上例生成跳转表时，跳转表中每一项为该项对应的 case 分支的目标地址同跳转表首地址之间的偏移，同时跳转表位于代码段中，紧随在寄存器间接跳转指令之后。

编译器 gcc 3.2.3 在为上例生成跳转表时，跳转表中每一项为该项对应的 case 分支的目标地址同该项的地址之间的偏移，同时跳转表位于只读数据段中。

3 switch 语句识别技术

不同版本的 gcc 编译器在生成跳转表时采用了不同的策略，出于效率和一致性处理的考虑，我们提出了前向预取结合反向切片及表达式置换来识别 switch 语句的技术。在识别跳转表过程中，可同时获取以下信息：跳转表的首地址，跳转表的项数，索引变量的下界，上界。这些信息将用于跳转表恢复。

3.1 前向预取

由 gcc 2.96 编译得到的跳转表可通过前向预取方法进行识别。前向预取是指当解码到寄存器间接跳转指令时，在二进制指令流中继续进行读取，判断读取的连续 4 个 8B 数据片是否具有如下性质：(1)较高位，如高 32 位，全为 0。(2)是否对 16 求余的结果为 0，即低 4 位为 0。

如果条件满足，可判定当前处理的二进制可执行文件为 gcc 2.96 编译得到，且解码到的寄存器间接跳转可恢复为 switch 语句，同时跳转表的首地址为寄存器间接跳转指令之后的指令束的起始地址。此时，只需再反向扫描已解码得到的 RTL 中间表示，寻找比较指令以确定跳转表项数，同时寻找减法指令以确定索引变量的下界，即可获得恢复 switch 语句所需的全部信息，上界可通过下界及跳转表项数计算得到。

如下为第 2 节所示的 C 语言源程序经 gcc 2.96 编译生成的二进制代码的部分汇编表示：

```
400000000000077c: br.few b6
4000000000000780: 30 00 00 00 00 00
4000000000000786: 00 00 60 00 00 00
400000000000078c: 00 00 00 00
4000000000000790: 90 00 00 00 00 00
4000000000000796: 00 00 c0 00 00 00
400000000000079c: 00 00 00 00
40000000000007a0: f0 00 00 00 00 00
40000000000007a6: 00 00 20 01 00 00
40000000000007ac: 00 00 00 00
```

其中跳转表紧随寄存器间接跳转指令之后，位于 4000000000000780 至 400000000000007b0，每项为 8B，其内容为该项对应的跳转目标地址同跳转表首地址之间的偏移。

3.2 反向切片及表达式置换

前向预取方法仅适用于 gcc 2.96 所采用的跳转表生成策略，不能识别 gcc 3.2.2 的情况。如果前向预取判断失败，则进行反向切片及表达式置换。步骤如下：

(1)在寄存器间接跳转处对代码进行切片；

(2)进行表达式置换，从而得到伪高级指令；

(3)将表达式置换得到的结果同预先设定的规格化形式进行比较。

以下为第 2 节所示的 C 语言源程序在 IA-64 体系结构下经 gcc 3.2.3 编译器 0 级优化后生成的二进制代码的汇编代码表示片段(nop 指令均被省略)：

```
4000000000000726: ld4 r14=[r35]
4000000000000730: cmp4.ltu p6,p7=7,r14
400000000000073c: (p06) br.cond.dptk.few 40000000000008a0
<main+0x200>
4000000000000740: ld4 r14=[r35];;
4000000000000746: shladd r15=r14,3,r0
400000000000074c: addl r14=96,r1;;
4000000000000750: ld8 r14=[r14];;
4000000000000756: add r15=r15,r14
4000000000000760: ld8 r14=[r15];;
4000000000000766: add r14=r14,r15
4000000000000776: mov b6=r14
400000000000077c: br.few b6;;
```

3.2.1 切片

IA-64 的可执行代码经由反汇编，得到程序中每个过程的由 RTL 和控制流图组成的中间表示。RTL 通过寄存器转换来描述机器指令的影响和作用。切片过程是在 RTL 中间表示上进行的。

切片的过程是沿着特定表达式使用的寄存器和条件码的传递闭包不断扩充[4]。某特定寄存器在某路径上结束的标准是：该寄存器从内存中加载(例如局部变量、过程参数或全局变量)，从另外一个函数返回或者在到达过程开始的位置时仍然未被定义(此时该寄存器为由caller设置的寄存器)。为了判定使用跳转表，除使用上述切片结束条件外，还增加了一个条件：当索引跳转的下界已经找到，而且其他信息也已经获得，则切片过程不再继续执行。当然，这个条件在下界为 0 时不能满足，因为此时不需要检测下界，切片根据其他条件结束。

3.2.2 表达式置换

当得到一个切片后，则可以通过前向置换的方式来进行表达式置换。表达式置换技术是逆向工程中从汇编代码恢复到高级语句所采用的一种通用技术。

假定在指令i处，寄存器r是通过a_k寄存器集合来定义的，即$r=f_1(\{a_k\},i)$，那么如果i处的定义是在程序到达指令j的所有路径中是唯一的对r的定义，而且在该路径上没有对a_k中的寄存器进行重新定义，则在j处的指令$(s=f_2(\{r,...\}),j)$中对寄存器r的使用可以使用前向置换。在j处的指令经过置换后形式为

```
s=f2({f1({ak},i),..},j)
```

从而不再需要 i 处的指令。例如对于如下的 RTL 代码：

```
r[14] := m[r[14]]
r[14] := r[15] + r[14]
```

```
r[582] := r[14]
```

在经过表达式置换后为

```
r[582] := r[15] + m[r[14]]
```

3.2.3 同规格化形式进行比较

3.2 节所示的汇编指令经切片和表达式置换后得到的伪高级语句可以抽象为如下的规格化形式：

```
jcond (var > upper) X
jmp [w*(var-lower)+T]+T+w*(var-lower)
```

其中，var 是一个局部变量，本例中为 num，upper 是 switch 语句中 case 的上界，本例中为 7，lower 是 switch 语句中 case 的下界，本例中为 0，T 是跳转表的地址，本例中为[r1+96]，即 4000000000000c10，而 w 是机器字长，本例中为 8。基于这些信息，可以得到 switch 语句中分支的数目 upper-lower+1，而跳转表中的内容是各个分支(default 分支除外)跳转目标地址同分支对应的跳转表项的地址间的偏移。

4 switch 语句的恢复

基于在识别 switch 语句过程中获得的跳转表地址、表项数、索引变量上下界即可完成对 switch 语句的恢复。需要注意的是，根据跳转表生成策略的不同，采用不同的计算公式来获取分支跳转的目标地址。例如对于 gcc 2.96，索引变量 var 对应的跳转地址计算公式为

```
[8*(var-lower)+T]+T
```

对于 gcc 3.2.3，计算公式为

```
[8*(var-lower)+T]+T+8*(var-lower)
```

以下为第 2 节所示程序经 gcc 3.2.3 编译得到的二进制代码恢复后的低级 C 程序片段：

```
(int64)v0=(int32)(r14);
r518=(((unsigned int64)7)<(tmp)) ? 1 : 0;
r519=(((unsigned int64)7)<(tmp)) ? 0 : 1;
if (( *(int*)&r518)==(1)) goto L1;
…
switch ((int64)v0) {
  case 0: goto L1;
  case 1: goto L1;
  case 2: goto L2;
  case 3: goto L3;
  case 4: goto L4;
  case 5: goto L5;
  case 6: goto L6;
  case 7: goto L7;
}
```

5 结束语

本文所提出的前向预取结合反向切片及表达式置换技术已经应用在 I2A 二进制翻译框架中，经过 gcc 2.96 和 gcc 3.2.3 两个版本在各种优化级别下生成的包含跳转表的二进制可执行程序的检测，证明是有效的。应用本技术，可有效地解决 IA-64 二进制代码翻译中基于跳转表的寄存器跳转的目标地址的恢复，从而提高翻译过程中对 IA-64 二进制代码处理的覆盖率。

参考文献

1 Cifutens C, Emmerik M. Recovery of Jump Table Case Statements from Binary Code[R]. Technical Report: 444, School of Information Technology and Electrical Engineering, The University of Queensland, 1998.

2 Cifuentes C, Sendall S. Specifying the Semantics of Machine Instructions[M]. IEEE CS Press, 1998.

3 Intel IA-64 Architecture Software Developer's Manual[Z]. Intel Corporation, 2000.

4 Gallagher K B, Lyle J R. Using Program Slicing in Software Maintenance[J]. IEEE Transations on Software Engineering, 1991, 17(8): 751-761.

~~~~~~

（上接第 46 页）

可以看出使用本文提出的小波压缩的图像，其重构后的图像更细腻清晰，视觉效果明显优于 CDF97 小波。限于篇幅，不再给出文本图像压缩后的重构图像。

4 结论

本文构造了基于伸缩矩阵$M_2=\begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix}$，具有四阶消失矩的不可分双正交小波的对称滤波器，其尺度函数具有非常好的 Hölder光滑性，且具有有理系数，并易于实现无浮点计算。本文同时对基于伸缩矩阵M_2的小波分解算法的对称边界延拓方式进行了讨论，给出了完全重构的对称边界延拓方法。实验结果表明，本文构造的关于两坐标轴对称的双正交小波滤波器对二维数据压缩具有较好的效果，其对应的不可分双正交小波是一种性能较好的小波基。

参考文献

1 Lin E B, Ling Y. Image Compression and Denoising via Nonseparable Wavelet Approximation[J]. Computational and Applied Mathematics, 2003, 155(1): 131-152.

2 黎明君，罗建书. 二元双正交梅花小波结合 SPIHT 的图像压缩方法[J]. 信息与电子工程, 2004, 2(3): 176-179.

3 刘　斌，彭嘉雄. 基于具有对称性的非张量积小波图像融合方法[J]. 量子电子学报, 2005, 22(3): 361-367.

4 Ruedin M C. Construction of Nonseparable Multiwavelets for Nonlinear Image Compression[J]. EURASIP Journal on Applied Signal Processing, 2002, 2(1): 73-79.

5 Daubechies I. Ten Lectures on Wavelets[Z]. Philadelphia: SIAM, 1992.

6 Han B. Analysis and Construction of Optimal Multivariate Biorthogonal Wavelets with Compact Support[J]. SIAM Journal on Mathematical Analysis, 2000, 31(2): 274-304.

7 Said A, Pearlman W A. A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees[J]. IEEE Trans. on Cirts. and Syst. for Video Tech., 1996, 6(3): 243-250.

8 Islam , Pearlman W A. Low Complexity Set Partitioning Embedded Coder: SPECK[C]. Proc. of VCIP’99, 1999.
~~~~~~