第49卷第1期 中国科学技术大学学报 Vol. 49, No. 1
2019年1月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Jan. 2019

文章编号:0253-2778(2019)01-0015-06

一种先进的扁平化谓词及编译优化方法

王向前[1],郑启龙[2],张仁高[2],韩东科[2]

(1.安徽大学互联网学院,安徽合肥 230039;2.中国科学技术大学计算机科学与技术学院,安徽合肥 230027)

摘要:谓词执行是有效挖掘控制流程序指令级并行性的一种机制.经典的谓词实现一般局部地逐个进行谓词计算而不能进行多谓词控制,有谓词计算路径过长等问题.针对经典谓词存在的问题,提出一种先进的扁平化谓词的实现方法,这种扁平化谓词可以全局地进行谓词计算,可以自然地进行多谓词控制.在此基础上,研究扁平化谓词的编译优化方法,给出了扁平化谓词编译优化框架.实验表明,本文提出的扁平化谓词及编译优化框架可以很好地提高多条件控制程序的执行效率.

关键词:谓词;多谓词;扁平化谓词;编译优化

中图分类号:$Tp_3 14$ **文献标识码**:A doi:10.3969/j.issn.0253-2778.2019.01.003

引用格式:王向前,郑启龙,张仁高,等.一种先进的扁平化谓词及编译优化方法[J].中国科学技术大学学报,2019,49(1):15-20.

ANG Xiangqian, ZHENG Qilong, ZHANG Rengao, et al. An advanced flat predicate mechanism and compiling optimization method[J]. Journal of University of Science and Technology of China, 2019, 49(1):15-20.

An advanced flat predicate mechanism and compiling optimization method

WANG Xiangqian[1], ZHENG Qilong[2], ZHANG Rengao[2], HAN Dongke[2]

(1. *School of Internet, Anhui University, Hefei* 230039, *China*;
2. *School of Computer Science and Technology, University of Science and Technology of China, Hefei* 230027, *China*)

Abstract: Predicate execution is a valid method to develop instructions parallelism in programs with control flow. Predicate computation is done one by one locally in the process of classic predicate for which multi-predicate control is forbidden. This may cause some problems such as long predicate implementation computation path. To solve these problems, an advanced flat predicate mechanism is proposed, which can do predicate computation globally, and perform multi-predicate control naturally. Then, compiling method for the flat predicate mechanism is researched, and a compiling framework for the flat predicate mechanism is presented. Experiments show that the flat predicate mechanism and compiling implementation framework can better enhance the executive efficiency of multi-condition programs.

Key words: predicate; multi-predicate; flat predicate; compiling optimization

收稿日期:2018-05-29;**修回日期**:2018-09-18

基金项目:国家核高基重大专项(2012ZX01034001-001)资助.

作者简介:王向前,男,1985年生,博士/高级工程师,研究方向:编译优化,E-mail: forward@mail.ustc.edu.cn

通讯作者:郑启龙,博士/副教授.E-mail: qlzheng@ustc.edu.cn

0 引言

谓词执行[1]是一种有效消除分支跳转的机制，它将程序进行控制依赖到数据依赖的变换. 分支跳转是进行指令级开发的基本障碍. 分支跳转的消除可以提高程序的执行性能. 从硬件体系结构上，分支跳转的消除可以减小甚至消除由分支预测失败导致的硬件开销；从编译层面讲，分支跳转的消除可以扩大调度范围，允许多个条件路径指令的重叠并行执行，挖掘跨越多个程序分支路径的指令级并行性.

本文针对数字信号处理器研究谓词的高效实现机制以及编译优化框架，针对传统经典谓词存在的问题，提出了一种创新的谓词实现方法，并研究编译优化方法，系统地给出了编译优化框架.

1 相关工作

谓词执行是一种硬件体系结构技术，有多种处理器体系结构支持谓词执行. ARM 处理器的指令集[2]支持所有指令的条件执行. 每条指令有 4 位条件域，用于说明指令执行的上下文. 通过检查指令的条件域和处理器状态寄存器的条件码，计算该指令的执行条件. 条件码一般通过执行比较指令设置.

Cydra 5[3]是 VLIW、支持谓词执行的多处理器系统. 一个指令行包括 7 条指令，每条指令可以分别条件控制. 每条指令增加一个额外的源操作数作为谓词.

TI 公司的 TMS320C6678[4]处理器可以在谓词寄存器真或假的情况下支持谓词执行，提供了 5 个通用谓词寄存器.

编译系统通过引入谓词指令消除分支指令挖掘多个分支的并行性，但是盲目地引入谓词指令不仅有可能降低程序性能，还会导致生成代码体积膨胀的问题.

AUGUST 提出了 Partial Reverse If-Conversion 谓词转换框架[5]，把谓词转换阶段放在调度阶段. 第一个阶段先进行激进的谓词转换. 第二阶段是调整阶段，把部分谓词形式根据代价模型重新转成跳转控制的基本块.

Han 等[6-7]分析了传统谓词控制在功耗和性能上的缺陷，包括较长的指令字设计和不必要的指令译码，将谓词控制引入到功耗控制，提出了一种低功耗的谓词控制机制，大幅降低程序运行时的功耗.

Rajendranradhika 等[8]针对粗粒度可重构阵列中分支语句通过谓词执行时需要执行每个分支上指令所导致的资源浪费问题，提出一种只执行一个分支的高效实现方法. 基于此，Yin 等[9]针对嵌套条件进行了深入研究，提出了实现思路.

Jordan 等[10]深入对比了基于编译器中间语言级的谓词转换与基于机器汇编级的谓词转换的区别，在中间语言级进行谓词转换具有较高的自由度，而在机器汇编级进行转换则有较多限制.

2 扁平化谓词实现

一般的经典谓词指令形式如下：

$p_1, p_2 = R_m \text{ cond } R_n$ （谓词定义指令）

(p_1) op_1 （谓词控制指令）

(p_2) op_2 （谓词控制指令）

如果条件计算 R_m cond R_n 成立（其中 cond 为某种比较计算，例如大于），则 p_1 为 0，p_2 为 1，从而指令 op_1 作废，op_2 正常运行；反之，如果条件 R_m cond R_n 不成立，则 p_1 为 1，p_2 为 0，从而指令 op_1 正常运行，op_2 作废.

这种谓词实现形式具有较强的通用性，也便于编译优化支持，但是也有一定的局限性. ①不能消除谓词控制的条件跳转语句. 如图 1 所示程序，涉及多个条件的谓词计算时，需要谓词控制的跳转语句，不能彻底消除跳转语句（注意指令 $p_1, p_2 = a > b$ 的含义是，当 $a > b$ 为真，p_1 为假，p_2 为真；反之，p_1 为真，p_2 为假，以下皆同）. ②有可能导致较长的谓词计算依赖路径. 如图 2 所示，谓词 p_3/p_4 的计算依赖于谓词 p_1/p_2，而 p_5/p_6 的计算又依赖于 p_3/p_4，最终导致 3 个条件计算还是串行执行.

数字信号处理器[11]一般提供较为丰富的逻辑比较运算资源，为了挖掘条件分支执行的并行性，本文设计了一种多谓词控制执行的谓词形式如下：

$(p_1, p_2, \cdots,)$ $R_s = R_m \text{ op } R_n$，

支持一个到多个谓词控制的指令执行，称为扁平化谓词形式. 指令 (p_1, p_2) $b = 10$ 的含义为谓词 p_1，p_2 都为真时 $b = 10$ 才执行成功. 与图 1(b)，图 2(b)相对应的扁平化谓词形式为图 3(a)，图 3(b)所示的谓词形式（注意：$p_1, p_2 = a > b \,||\, p_3, p_4 = c > d \,||\, p_5, p_6 = a > c$ 的含义是这 3 条谓词定义指令并行执行，数字信号处理器一般支持显示指令级并行）. 图 3(a)与图 1(b)相比，主要是消除谓词控制的条件跳

```
int muliconditon ( int a, int b, int c, int d ) {
    int re;
    if(a> b  && c> d || a>c)
        re = 2;
    else
        re = 3;
    return re;
}
```

(a)

```
BB 1:
 p1,p2  = a > b
 (p1) br_ cond BB 3
BB 2:
 p3, p4  = c > d
 (p4 ) br_ cond  BB  4
BB 3:
  p5,p6 =a>c
 (p5 ) br_ cond BB 5
BB 4:
    re = 2
     B ExitBB 6
BB 5:
     re = 3
ExitBB 6:
```

(b)

图 1　多条件计算谓词代码

Fig. 1　Predicate code for multi-condition

```
void func ( int a, int * ret ) {
int b;
if (a< 10 ){
 b = 10;}
 else if (a< 30 ){
 b = 20; }
 else if (a< 60 ) {
 b = 30; }
else
 b = 40;
* ret = b;
}
```

(a)

```
p1, p2 = a<10
(p2 ) b =10
(p1 )p3 , p4 = a<30
(p4 ) b =20
(p3 )p5 ,p6 =a<60
(p6 ) b =30
(p5 ) b =40
```

(b)

图 2　长路径计算谓词代码

Fig. 2　Predicate code with long-path

转指令，图 3(b)与图 2(b)相比，消除了较长的谓词计算依赖路径；

综上可见，扁平化谓词形式可以充分地挖掘条件计算的并行性，它比经典谓词形式更灵活、高效.

```
p1, p2 =a>b || p3, p4 = c>d || p5, p6 =a>c
(p2,p4) re= 2 ||( p6) re=2 ||(p1,p3) re=3 ||(p2,p3,p5) re=3
```

(a)

```
p1, p2 =a< 10 || p3, p4 = a<30 ||p5, p6 =a<60
(p2) b=10|| (p1, p4) b=20||(p1,p3,p6) b=30||(p1,p3,p5) b=40
```

(b)

图 3　扁平化谓词代码

Fig. 3　Flat predicate code

3　编译支持框架

一般的经典谓词编译支持框架如图 4 所示，主要分为 4 个阶段. 主要的过程是：先选择适合谓词转换的 region[12]，然后进行数据流分析，识别每个基本块的谓词，根据基本块谓词之间的依赖关系，插入相应的谓词定义语句，此时可能会引入谓词控制的谓词定义指令或跳转指令，最后为每个基本块的指令放置谓词控制寄存器，并进行基本块合并.

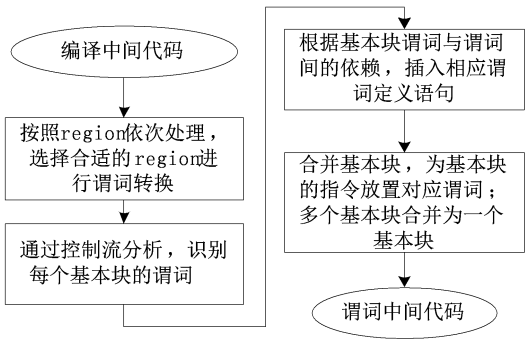

图 4　经典谓词编译框架

Fig. 4　Classic predicate compiling framework

经典的谓词编译提供了一个可借鉴的实现框架，但是并不能适用于扁平化谓词，这是因为：

(Ⅰ)它支持的是经典谓词形式，并不适合本文提出的扁平化谓词；

(Ⅱ)经典谓词形式的计算过程是局部的、嵌套的，而扁平化谓词形式是全局的、扁平化的；

(Ⅲ)扁平化谓词形式较为特殊，支持多个条件的扁平化合并，更为高效.

基于此，我们提出了新的面向扁平化谓词的编译优化框架，如图 5 所示.

图 5 所示的面向扁平化谓词的编译框架与图 4 所示的经典谓词编译框架的主要区别在第 3 个处理过程. 扁平化谓词的谓词计算是全局的，所以需要找到每个基本块的全部的绝对控制谓词路径.

下面以图 1(b)程序为例说明扁平化谓词编译

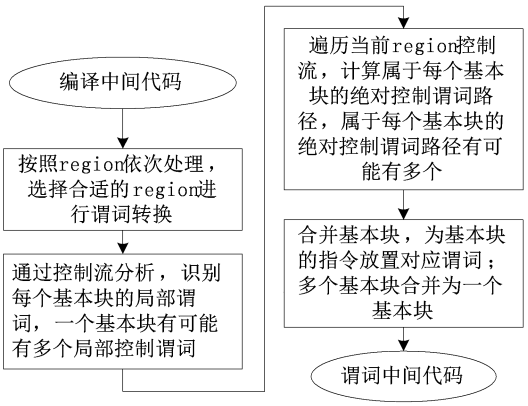

图 5 扁平化谓词编译框架

Fig. 5 Flat predicate compiling framework

支持框架的核心流程，如图 6、图 7 所示. 通过控制流分析，识别每个基本块的控制谓词，一个基本块有可能有多个局部控制谓词. 如图 6 所示，控制块 BB2，BB5 的局部控制谓词分别为 p_2，p_5；而控制块 BB3，BB4 则有多个局部控制谓词，其中 BB3 有 p_1，p_3 两个局部控制谓词，BB4 有 p_4，p_6 两个局部控制谓词.

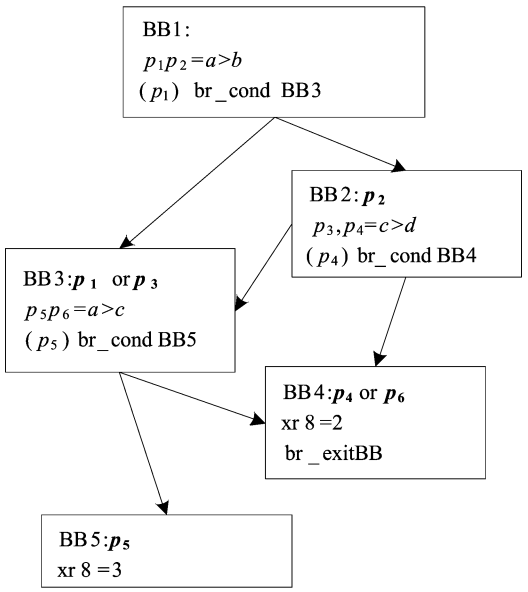

图 6 识别局部谓词

Fig. 6 Recognition of local predicate

接着计算每个基本块的绝对控制谓词路径，沿着控制流方向依次计算，该过程是迭代的，直至找到每个控制块的全部控制谓词路径. 如图 7 所示，控制块 BB2 的绝对控制谓词路径只有一个 p_2，而控制块 BB3，BB4，BB5 则有多个绝对控制路径. BB3 的绝对控制谓词路径为有两个，分别是 p_1、p_3p_2，意思是基本块 BB3 指令的控制谓词的要么是 p_1，要么是 p_3，p_2（多谓词控制执行）. BB4 的绝对控制谓词路径有 3 个，p_4p_2、p_6p_1、$p_6p_3p_2$. BB5 的绝对控制谓词路径有两个，p_5p_1、$p_5p_3p_2$.

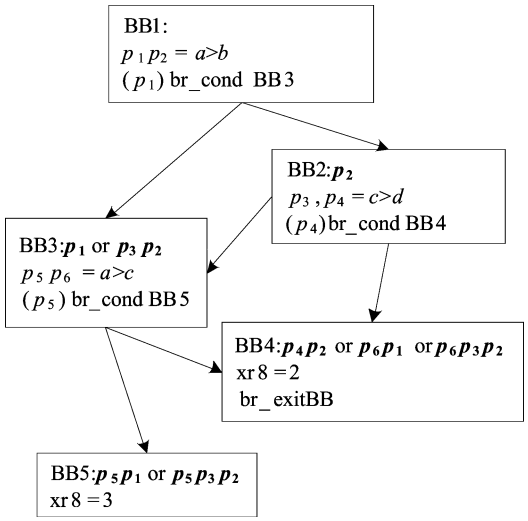

图 7 识别谓词控制的绝对路径

Fig. 7 Recognition of absolute predicate path

找到每个基本块的绝对控制谓词路径是为了适应扁平化谓词形式. 在合并基本块阶段，就可以把全部的条件计算放置合并基本块的靠前位置，而把有效的多谓词控制指令放置在合并基本块的靠后位置，最终形成扁平化的谓词计算形式，如图 8 所示. 图 8 与图 3(a)有少许区别，但两者是等价的，不需要生成谓词控制的谓词定义指令：(p_2) $p_3, p_4 = c>d$ 类型的指令，而是直接把 $p_3, p_4 = c>d$ 类型指令直接移动到谓词寄存器块的靠前位置. 在扁平化谓词的支持下，不会生成谓词控制的谓词定义指令，直接从机制上消除了谓词之间的数据依赖. 谓词之间只存在两种关系，互斥关系、对等关系. 例如图 8 中 p_1 与 p_2 是互斥关系，而 p_1 与 p_3 是对等关系.

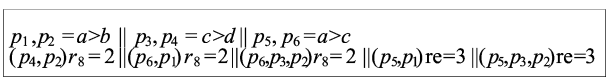

图 8 谓词基本块

%Fig. 8 Predicate block

4 性能测试

魂芯 DSP[13-14]一款向量化的、支持超长指令字的数字信号处理器. 在魂芯 DSP 软件模拟器分别实现经典谓词和扁平化谓词的原型，分别记为 simulator with classic predicate (SCP)和 simulator with flat predicate(SFP). 这两款模拟器运行于 window 命令行下，通过执行编译器生成的可执行程序，返回运行的周期数. 基于魂芯 DSP C 编译器

实现对经典谓词和扁平化谓词的编译优化框架，分别记为 compiling for classic predicate（CCP）和 compiling for flat predicate(CFP). 通过选项控制，为用例分别生成在经典谓词和扁平化谓词优化支撑的可执行程序，分别运行于 SCP 和 SFP 模拟器下获得用例的执行周期，用例的性能是运行周期的倒数. 选取如表 1 所述的用例作为性能测试集合对比验证两种谓词机制的性能表现. 这些用例主要来源于图像图像[15]的经典操作、语音编解码[16]等. 为了更好地分析实验结果，有必要说明测试用例的条件执行的核心操作，如表 2 所示.

表 1　谓词优化测试集合

Tbl. 1　Benchmark for predicate optimization

测试用例	描述
binaryassign	二值量化，把一组数据量化为 0 或 1
compareintersect	求两组数据对应较小的值的和
overlaparea	计算两个矩形的重叠面积
weightsum	求权和
adpcm	自适应差分脉冲编码调制

表 2　测试集合条件执行特征

Tbl. 2　Characteristics of condition execution in the testsuite

测试用例	核心操作
binaryassign	if(a[i] < pivot) b[i]=1;else b[i]=0;

compare-intersect	if(hist1[i] <=hist2[i])
	result+= hist1[i];
	else result+= hist2[i];

overlaparea	if(cond1 &&cond2 && …) sum++;

weightsum	if(i < len1) sum += 2 * a[i];
	else if(i < len2) sum += 3 * a[i];
	else if(i < len3) sum += 4 * a[i];
	elsesum += 5 * a[i];

adpcm	—

用例 binaryassign 是二值量化，选自图形图像处理领域的经典运算，核心操作是根据条件比较结果向一个数组写 0 或 1；用例 compareintersect 求两组数据对应较小的值的累加和，选自图形图像处理领域的经典操作，核心操作是根据比较进行累加操作；用例 overlaparea 计算两个矩形的重叠面积，选择图形图像处理领域，核心操作是根据条件进行累加操作；用例 weightsum 是求权和，核心操作根据条件对数据乘以不同的权值，并累加；用例 adpcm 是自适应差分脉冲编码调制算法，核心操作是依据调制协议提供的各种条件判断依次输出相应的值到输出数组.

在进行性能测试时，每个测试用例的问题规模确定如下，其中 binaryassign、compare intersect、weightsum、adpcm 的数组大小均为 10 000，overlaparea 计算的矩形方框大小为 10000×10000；然后进行两种谓词机制及编译框架的性能测试对比，测试结果如图 9 所示.

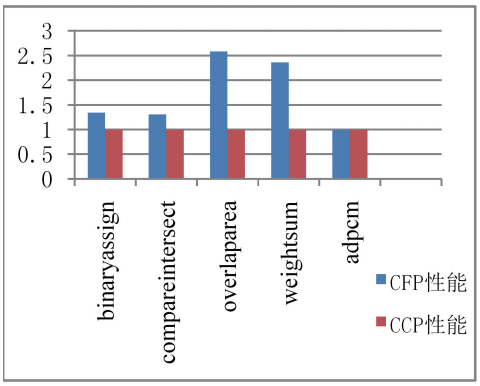

图 9　扁平化谓词机制性能

Fig. 9　Performance of flat predicate mechanism

测试用例 adpcm 在两种谓词框架上性能持平，这是由于 adpcm 算法核心只含有单条件控制语句，CFP 和 CCP 产生的谓词控制代码是一致的. 与经典谓词相比，用例 binaryassign、compareintersect 在扁平谓词的支撑下性能均得到提升，提升幅度超过 30%，这是因为这两个用例的循环体是两条件控制，在扁平谓词的支撑下 CFP 实现了两个条件的扁平化并行执行，缩短了谓词计算路径，从而提升了代码的生成效率. 性能提升最明显的用例是 overlaparea 和 weightsum. 用例 overlaparea 在生成经典谓词代码时，实际上产生了若干谓词控制跳转语句，并未实质消除跳转语句；在生成扁平谓词时，使用了多条件谓词控制指令，消除了跳转语句，性能得到了大幅提高，提升幅度超过 150%. 用例 weightsum 在经典谓词条件下会生成较长的谓词计算依赖路径；而在扁平谓词条件下则实现了 4 个条件计算的并行，提升幅度超过 100%.

5　结论

本文对经典谓词进行深入的研究，基于数字信号处理器提供的丰富的比较逻辑运算资源基础上，

提出了一种先进的扁平化多谓词控制实现机制,并给出了扁平化谓词编译优化支持框架.实验结果表明,扁平化谓词机制及编译优化框架能够很好地挖掘条件计算的并行性,大幅提升多条件控制程序或具有较长谓词依赖路径程序的性能.

参考文献(References)

[1] SHEIKH R, TUCK J, ROTENBERG E. Control-flow decoupling: An approach for timely, non-speculative branching[J]. IEEE Transactions on Computers, 2015, 64(8):2182-2203.

[2] GOODACRE J, SLOSS A N. Parallelism and the ARM instruction set architecture[J]. Computer, 2005, 38(7): 42-50.

[3] AIKEN A, BANERJEE U, KEJARIWAL A, et al. Overview of ILP Architectures[M]. USA: Springer, 2016.

[4] MOU X G, WEI G H, ZHOU X. Parallel programming and optimization based on TMS320C6678 [J]. Applied Mechanics & Materials, 2014, 615: 259-264.

[5] AUGUST D I, HWU W M W, MAHLKE S A. The partial reverse if-conversion framework for balancing control flow and predication[J]. International Journal of Parallel Programming, 1999, 27(5): 381-423.

[6] HAN K, PARK S, CHOI K. State-based full predication for low power coarse-grained reconfigurable architecture [C]//Proceedings of the Design, Automation and Test in Europe Conference and Exhibition. Dresden, Germany: IEEE, 2012: 978-3-9810801-8-6.

[7] HAN K, AHN J, CHOI K. Power-efficient predication techniques for acceleration of control flow execution on CGRA[J]. ACM Transactions on Architecture & Code Optimization, 2013, 10(2): 1-25.

[8] RAJENDRANRADHIKA S H, SHRIVASTAVA A, HAMZEH M. Path selection based acceleration of conditionals in CGRAs[C]// Design, Automation & Test in Europe Conference & Exhibition. Grenoble, France: EDA Consortium, 2015: 121-126.

[9] YIN S, ZHOU P, LIU L, et al. Acceleration of nested conditionals on CGRAs via trigger scheme [C// Proceedings of the IEEE/ACM International Conference on Computer-Aided Design. Austin, USA: IEEE, 2015: 597-604.

[10] JORDAN A, KIM N, KRALL A. IR-level versus machine-level if-conversion for predicated architectures [C]// Proceedings of the 10th Workshop on Optimizations for DSP and Embedded Systems. Shenzhen, China: ACM, 2013: 3-10.

[11] WALRAVENS C, DEHAENE W. Low-power digital signal processor architecture for wireless sensor nodes [J]. IEEE Transactions on Very Large Scale Integration Systems, 2014, 22(2): 313-321.

[12] YOU Y P, CHEN J R. A static region-based compiler for the Dalvik virtual machine[J]. Software Practice & Experience, 2015, 46(8): 1-23.

[13] 洪一, 方体莲, 赵斌,等. “魂芯一号”数字信号处理器及其应用[J]. 中国科学信息科学, 2015, 45(4): 574-586.

[14] 洪一. 魂芯一号:国产处理器勇攀高峰[J]. 科技纵览, 2015, (9): 55-57.

[15] 杨帆. 数字图像处理与分析[M]. 北京:北京航空航天大学出版社, 2015.

[16] 徐晓亮, 梁维谦. 基于 Ezairo DSP 的 ADPCM 语音解码器设计[J]. 计算机应用, 2015,35(2): 319-321.