

一种改进算法的低功耗嵌入式系统代码压缩设计

张瑞峰, 马文杰

(天津大学 电子信息工程学院, 天津 300072)

摘 要: 采用一种代码压缩的方法来降低嵌入式系统的功耗. 在分析指令的特性后对目标代码进行指令合并和指令分割, 运用范式 Huffman 算法对处理后的指令进行编码生成索引查找表. 最后, 通过查找表中索引字和指令的对应关系完成目标代码的压缩和解压缩. 实验使用 simplescalar 模拟器对部分嵌入式基准测试程序进行压缩, 用代码压缩率和功耗减少率进行评估, 统计数据表明提出的改进算法可有效节省存储空间、降低系统功耗.

关键词: 代码压缩; 嵌入式系统; 低功耗; 范式 Huffman 算法

中图分类号: TP31

文献标识码: A

文章编号: 1000-7180(2016)05-0085-04

An Improved Code Compression Algorithm for Low Power Embedded System Designs

ZHANG Rui-feng, MA Wen-jie

(School of Electronic Information Engineering, Tianjin University, Tianjin 300072, China)

Abstract: A new code compression scheme was proposed to solve the power consumption problem in embedded systems. After analyzing the characteristics of instructions in target program, instructions will be combined and split, then the canonical Huffman algorithm is used to encode the modified instructions to generate look-up tables. Finally, code will be compressed and decompressed on the base of the corresponding relations between instructions and code words in look-up tables. According to the compression rate and power reduction rate of partial programs of embedded benchmarks compressed by simplescalar simulator, statistical data showed that the proposed algorithm can effectively save the storage space and reduce the power consumption of system.

Key words: Code compression; Embedded system; Low power; Canonical Huffman algorithm

1 引言

精简指令集配合流水线的工作模式使得嵌入式系统的执行速度得到显著提高, 但精简指令集占用的存储空间随嵌入式设备功能的多样性和复杂性呈快速增长. 在嵌入式系统整体能耗中, 存储器部分的信息存储和读取操作消耗的电能占很大比重^[1], 如果对存储的代码进行压缩, 就可以降低系统的存储容量需求, 当需要执行存储的压缩指令时, 存储器的读取功耗和总线传输功耗也得到相应的降低, 这即改善了系统性能也延长设备的使用时间. ARM's Thumb 和 MIPS16e 指令集就是这方面的典型应用.

代码压缩的思想起始于 1992 年 Wolfe 等提出的 CCRP(Compressed Code RISC Processor)^[2] 设计方案, CCRP 改变处理器的硬件架构以实现压缩程序的解压和执行操作, 针对 MIPS 指令集的测试程序, 他们使用 Huffman 算法编码, 压缩率近似为 73%. Y. Nekritch 在范式 Huffman 算法中运用 LUT(Look-up Tables)^[3] 来存放索引字, 提高了压缩率和解码效率. T. Bonny 等^[4] 在此基础上对指令进行分割, 他们的分割算法比未分割就进行范式 Huffman 编码的压缩率平均提高 20%. 本文对统计频率高且相邻指令进行合并^[5], 统计频率低指令进行分割, 运用范式 Huffman 算法生成两个对应的查找表来改善代码压缩率和存储功耗.

2 代码压缩模型

代码压缩是在程序下载到嵌入式设备之前完成,需要根据压缩算法设计对应的解压方案^[6],整体的设计模型如图 1 所示.可以看出在原有的硬件结构基础上,指令预取段多了一个解压器,同时添加一个外部 RAM 用来存储查找表信息以不影响解压效率,对于其他的硬件设计并没有过多改动,这种方案易于在现有的嵌入式系统上得到实现.

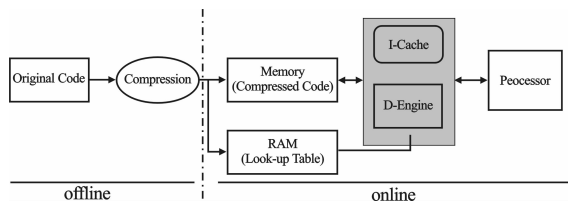


图 1 代码压缩设计模型

图中阴影区域为解压器和指令缓存器,分为指令预取前解压(pre-cache)和指令预取后解压(post-cache)两种^[7]设计结构.在指令预取后解压结构中,I-Cache 存放编码指令,解压后还原成原始指令,这种结构能有效地增加指令缓存器的空间利用率,要求是对压缩指令做到快速解压,不能影响 CPU 的正常工作,这会增大解压硬件的设计难度,不过能较好的减小功耗和提高缓存命中率.本文的设计中会生成查找表来帮助快速解码,因而可以采用指令预取后解压结构.

3 代码压缩衡量标准

压缩率(Compression Ratio, CR)是衡量一个压缩算法对于系统存储空间的有效利用情况,压缩率等于压缩后程序尺寸/原程序尺寸^[7].在有查找表的压缩算法中,代码压缩率计算式如下:

$$CR = \frac{\sum_{i=1}^L N_i \times i + \sum_{i=1}^L C_i \times i + mn}{N \times m} \times 100\% \quad (1)$$

代码压缩后存储器功耗降低情况用功耗减少率 P 衡量,存储器的功耗主要取决于程序目标代码所占用的存储空间.功耗减少率的计算式为

$$P = 1 - \frac{\sum_{i=1}^L N_i \times i + \tau(\sum_{i=1}^L C_i \times i + mn)}{N \times m} \quad (2)$$

式中, N 为原程序指令总数; L 为编码指令位宽不同的个数; m 为原程序中指令位宽; n 为原程序中指令

不同的指令数; N_i 为编码指令中位宽为 i 的个数; τ 为同一条指令在片上存储器和片外存储器上的功耗比, $0.5 < \tau < 0.7$; C_i 为查找表中编码指令位宽为 i 的条目数;

从式(1)可以看出,查找表的空间占用率在索引条目少的时候并不高,但如果索引条目总数 $\sum_{i=1}^L C_i$ 达到 256 这一量级后,解码操作将会占用较多时间和资源,这对于系统的性能而言是一个灾难.所以,合理的减少查找表大小是十分必要的.

式(2)中, $\tau(\sum_{i=1}^L C_i \times i + mn)$ 表示查找表中的目标代码所占用的等价片外存储器的空间大小,同样的如果减小 $\sum_{i=1}^L C_i$ 的尺寸,也会降低片外存储功耗.

4 代码压缩实现过程

4.1 编码算法

本文运用基于查找表的范式 Huffman 算法对目标程序进行编码,并在此基础上进行改进.范式 Huffman 编码是 Huffman 编码的一个分支,它具有两点约束条件:

(1)对于给定长度的代码,其取指的词典顺序与它们代表的符号顺序相同;

(2)根据词典顺序,较短的代码在较长代码前.

因此,解码器只需知道码字的最大长度和每个码字在具有相同长度码字中的序列号^[3,4],再通过逻辑运算就可以完成解码而不需要重构整个范式 Huffman 编码树,这极大简化解码.

通过分析嵌入式系统指令集的特性和执行模式,考虑将程序指令进行先处理后编码.假设,范式 Huffman 算法对程序编码的生成树节点为 $\langle t_1, t_2, t_3, \dots, t_n \rangle$,程序中含有 k 条重复频率相同且相邻的指令,在编码时将其合并为一进行编码,即树节点是 $\langle t_1, t_2, t_3, \dots, t_{n-k} \rangle$, $\sum_{i=1}^{L'} C_i' = n - k$,则此时的压缩率为

$$CR' = \frac{\sum_{i=1}^{L'} N_i' \times i + \sum_{i=1}^{L'} C_i' \times i + mn}{N \times m} \times 100\% \quad (3)$$

($L \geq L'$, $N_i \geq N_i'$, $C_i \geq C_i'$, $i = 1, 2, 3, \dots, L'$)

若 $k \neq 0$, $CR' < CR$.同理, $P' > P$.

如果再将统计频率为 1 的指令以对半方式进行分割,分割后的指令段再用范式 Huffman 进行编

码,图 2 所示为一伪指令序列先合并后分割的算法实现流程. 其中,查找表按标志位 0、1 分两个子表,标志位为 0 的表中存放初始、合并指令及其索引信息,标志位为 1 的表中存放分割后指令段. 值得一提的是,不管合并指令的频率是否为 1,都不会将其再进行分割操作.

假设存在 u 条单独指令. 即:

$$\sum_{i=1}^{L'} C_i' = n - k - u, \sum_{i=L'}^{L''} C_i' = u. \text{ 那么}$$
$$CR' = \frac{\sum_{i=1}^{L'} N_i' \times i + \sum_{i=L'}^{L''} N_i' \times i + \sum_{i=1}^{L'} C_i' \times i + \sum_{i=L'}^{L''} C_i' \times i + m}{N \times m} \times 100\% \quad (4)$$

$$CR^* = \frac{\sum_{i=1}^{L'} N_i' \times i + \sum_{i=L'}^{L''} N^* \times i + \sum_{i=1}^{L'} C_i' \times i + \sum_{i=L'}^{L''} C_i^* \times i + n'm' + N - k}{N \times m} \times 100\% \quad (5)$$

$N - k$ 表示标志位信息占用的片上存储空间,大小为 $N - k$ bit. 正常情况下, $\sum_{i=L'}^{L''} N_i' > \sum_{i=L'}^{L''} N_i^*$, $\sum_{i=L'}^{L''} C_i' > \sum_{i=L'}^{L''} C_i^*$, $mn > n'm'$, 而 $N - k$ 的引入会降低一定的压缩率,以 32 bit 位宽的指令集为例, $m = 32$, $\frac{N - k}{N \times m} \times 100\% \approx 3\%$, 这点影响不大. 所以, $CR^* < CR'$, 同理而言 $P^* > P'$.

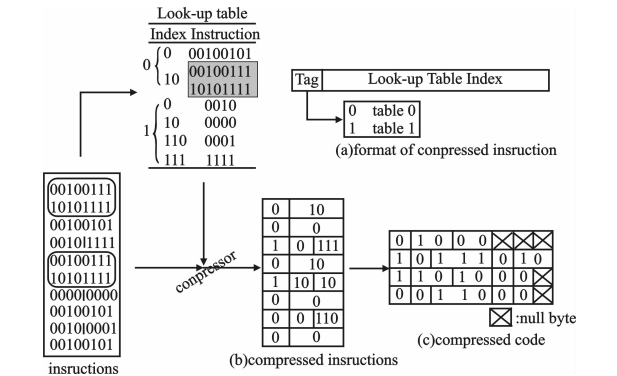


图 2 改进算法示意图

改进算法具体生成流程如图 3 所示. 对原指令进行编码生成查找表之后,用查找表中的索引值来替换原文件中的指令,从而产生一个只含编码指令的文件. 而这个文件就是交由 CPU 执行前解压的编码文件. 由改进算法产生的查找表,在索引字替换的过程中,每个索引字的首位添加标志位 0/1 以表示该索引字属于 table 0/1. 标志位 1 只在两相邻索引字的前者添加,后一索引字默认从 table 1 中译码以降低存储空间和检索时间. 编码指令格式如图 2 (a)所示,由改进算法生成的编码指令如图 2 (b)所示.

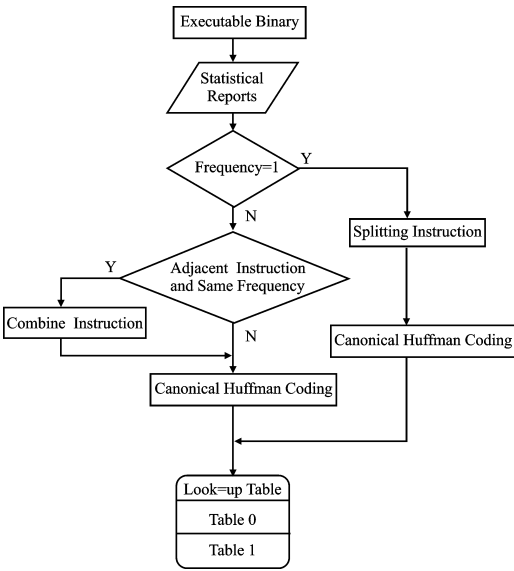


图 3 查找表生成流程

4.2 解压设计

针对本文的压缩算法设计如图 4 所示的解压方案,以 32 bit 位宽指令集为例. 首先, 32 bit 缓存器从预取指令数据流中移交 L bit (L 是编码索引字的最大长度)到 L bit 缓存器中,对于 L bit 缓存器的数据流先判断首位比特是 0/1, 如果为 0, 索引值与查找表中 RAM 0 进行匹配; 如果为 1, 则与查找表 RAM 1 匹配, 且下一个索引字的首比特位不进行判断直接译码, 输出结果由缓冲器交给处理器. 这种解码方式只增加了流水线的深度, 使得处理器对一条指令的执行可以在一个时钟周期完成.

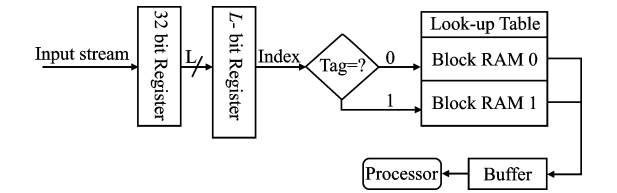


图 4 解压流程

5 实验测试与评估

实验采用体系结构模拟器 simplescalar 在 Linux 系统环境下进行, simplescalar 能模拟 RISC 体系结构 5 级流水线的工作模式. 测试模拟器为 sim-safe, 编译器版本为 gcc-2.7.2.3, 选取嵌入式基准测试程序组 Mibench 中的 search (search_small)、dijkstra、patricia、blowfish、rijndael、sha 程序, 测试程序经编译之后生成对应的二进制可执行文件. 通过编译后生成的是 64 位指令格式, 需要将其映射成对应的 32 位指令. 然后, 用模拟器分别对

目标文件进行不处理、合并处理和合并后分割处理,数值代入式(1)、(3)、(5)中计算得到三种算法的压缩率如表 1 所示(精确到 0.01),同理得到图 5 的功耗减少率,其中, τ 按 0.6 进行计算.

通过比较表 1 中三种方案的字典尺寸,可以看出通过指令分割操作得到的字典最小.而且,即使考虑到标志位信息产生的 3%额外占有空间,改进算法的压缩率和功耗减小率依然很可观,这可以通过表 1 和

图 5 比较得出.需要指出的是,合并指令有助于提高处理器运行速度.因为,程序中分支、跳转指令只占少部分,大多数情况下,要执行的下一条指令都是紧跟上一条指令之后.当预取到合并指令的索引字时,解压后得到的是两条指令,且这两条指令必然顺序执行,那么就减小了一次取值、解压操作;只进行指令分割会产生额外的解压功耗,因为执行一条分割指令需要两次解压.综合应用能得到很好的折中.

表 1 压缩率比较 kB

程序	程序尺寸	C字典尺寸	C'字典尺寸	C*字典尺寸	C	C'	C*
blowfish	216.652	72.246	55.992	14.301	0.77	0.66	0.56
dijkstra	268.452	71.706	56.082	19.4	0.70	0.61	0.55
patricia	310.756	72.559	71.153	16.661	0.67	0.64	0.54
rijndael	270.652	94.723	71.441	10.487	0.79	0.68	0.65
search	221.636	77.211	56.606	10.953	0.78	0.66	0.57
sha	306.712	76.571	56.609	13.294	0.69	0.59	0.53
平均结果	265.81	77.50	61.32	14.18	0.73	0.64	0.57

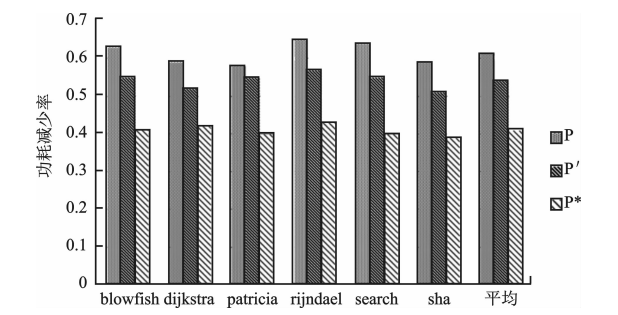


图 5 功耗减少率比较

6 结束语

本文分析程序代码特性之后在原有的算法基础上进行改进,从有效降低压缩率和查找表空间占用率的角度去设计压缩和解压方案,指令合并能提高指令执行速率,减少解压次数;指令分割能得到容量更小的查找表.实验数据表明,改进的算法降低了存储空间和系统功耗.此外,由于系统设计中引入解压操作,需要在实际硬件上测量系统功耗情况^[8],这有待后续研究.

参考文献:

[1] Benini L, Macii A, Nannarelli A. Cached-code compression for energy minimization in embedded processors [C]//International Symposium on Low Power Electronics and Design, California, USA; ACM, 2001: 322-327.

[2] Wolfe A, Chanin A. Executing compressed programs on an embedded RISC architecture [J]. ACM Sigmicro Newsletter, 1992, 23(1-2): 81-91.

[3] Nekritch Y. Decoding of canonical Huffman codes

with look-up tables [C]// Proceedings of the Data Compression Conference, Snowbind, USA; IEEE, 2000: 28-30.

[4] Bonny T, Henkel J. Instruction splitting for efficient code compression [C]// Design Automation Conference, San Diego, USA; IEEE, 2007: 646-651.

[5] Azevedo D W R, Moreno E D, Nattan Palmeira I. A new code compression algorithm and its decompressor in FPGA-based hardware [C]// Integrated Circuits and Systems Design, 2013 26th Symposium on. Citiba, Brazil; IEEE, 2013: 1-6.

[6] Lin C, Lin C H, Wang W J. A power-aware code-compression design for RISC/VLIW architecture [J]. Journal of Zhejiang University Science C, 2011, 12(8): 629-637.

[7] Bonny T, Henkel J. Huffman-based code compression techniques for embedded processors [J]. ACM Transactions on Design Automation of Electronic Systems, 2010, 15(4): 523-530.

[8] Lekatsas H, Wolf W. SAMC: a code compression algorithm for embedded processors. [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2000, 18(12): 1689-1701.

作者简介:

张瑞峰 男, (1974-), 副教授, 硕士生导师. 研究方向为嵌入式测量技术研究. E-mail: wenjieman@tju.edu.cn.

马文杰 男, (1991-), 硕士研究生. 研究方向为嵌入式系统应用开发.