Gzip压缩的硬件加速电路设计

李 冰,王超凡,顾 巍,董 乾
(东南大学集成电路学院,江苏南京 210096)

摘 要: 硬件无损压缩技术可以发挥专用电路的速度和功耗优势,被广泛应用于大数据计算以及通信领域.本文以GNUzip(Gzip)数据无损压缩技术为原型设计了一种硬件压缩电路.通过采用双Hash函数、并行匹配处理、面向硬件存储的LZ77压缩存储格式、高效数据拼接器等加速方法,发挥并行计算和流水线结构优势,提升压缩速率.该硬件压缩电路基于Verilog HDL设计,使用现场可编程门阵列(FPGA)进行测试和验证.测试数据表明:与软件压缩方式相比,该硬件压缩电路在获得适中压缩率(65.9%)的同时,其压缩速率得到显著提升,平均压缩速率达171Mb/s,满足网络通信、数据存储等实时压缩应用需求.

关键词: 无损压缩;Gzip;硬件;LZ77;FPGA
中图分类号: TP391.1;TN492 **文献标识码:** A **文章编号:** 0372-2112 (2017)03-0540-06
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2017.03.005

Hardware-Accelerated Circuit Design for Gzip Compression

LI Bing, WANG Chao-fan, GU Wei, DONG Qian
(*School of Integrated Circuit, Southeast University, Nanjing, Jiangsu* 210096, *China*)

Abstract: The hardware implementation of lossless data compression is wildly used in big data computing and communication, since it combines the speed and power advantage of the dedicated circuit. This paper proposed a hardware compression circuit based on GNUzip(Gzip) lossless data compression algorithm. The dual Hash functions, parallel match processing, hardware storage oriented LZ77 compression data format and high-performance data adaptor were involved to accelerate the compression speed with the advantages of parallel calculation and pipeline structure. The hardware compression circuit, based on Verilog HDL, was tested and verified by field programmable gate array(FPGA). The test data shows that, compared with software implementation, the compression speed of hardware circuit is improved significantly while the compression rate is 65.9%. The average speed is up to 171Mb/s that can satisfy the real-time compression requests of network communication and data storage.

Key words: lossless compression; Gzip; hardware; LZ77; FPGA

1 引言

数据无损压缩技术可节约存储空间、降低数据传输带宽需求[1],且不影响数据重构质量,常以软件方式实现.该方式配置灵活,易获得较好的压缩率,但存在资源消耗多、功耗大、处理速率低等性能瓶颈,无法满足大数据环境下的实时压缩处理需求.

基于硬件的数据无损压缩实现方式,可充分利用其并行计算和流水线的性能优势,以很小的压缩率损失为代价获得极高的处理效率,同时几乎不占用上位机的运算、存储资源.因而被国内外院校和企业关注和研究[2,3];清华大学和百度公司使用分布式随机存储器(RAM)构成字典,基于定制的FPGA,实现了高速多通道的Gzip无损数据压缩[4];微软公司和华盛顿大学合作设计了新型LZ77算法的Hash处理结构,取代了原有的链表结构,有效提升了Hash的处理效率[5];滑铁卢大学完成了动态Huffman编码的硬件实现等工作,进一步提升了压缩率性能[6].

Gzip是一种基于字典[7,8]和熵编码[9]的无损压缩方式.本文以提高数据压缩速率为主要目标,兼顾硬件资源消耗,使用双Hash函数、并行匹配处理方法、面向硬件存储的LZ77压缩存储格式、高效数据拼接器等多

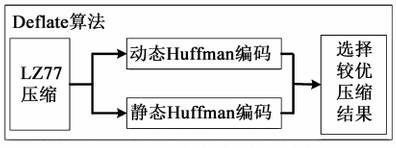
收稿日期:2015-08-17;修回日期:2016-01-15;责任编辑:覃怀银
基金项目:国家自然科学基金(No.61571116)

种提速方法,基于 Xilinx Vertix-6 ML605 硬件平台实现了 Gzip 数据无损压缩. 测试发现其平均压缩速率可达 171Mb/s,相较无损压缩软件的平均处理速率提升可达 8 倍多.

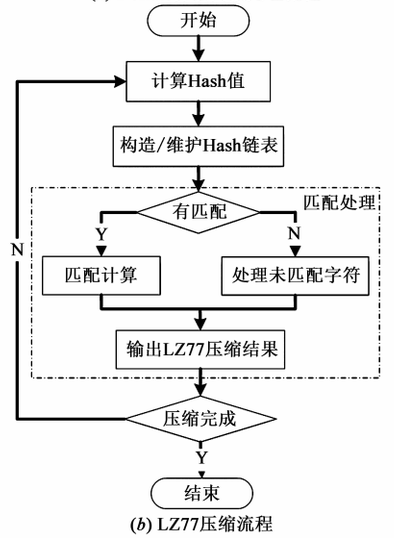
2　Gzip 数据无损压缩技术

Gzip 是一种基于 Deflate 算法的两级无损压缩方式,该两级处理流程如图 1(*a*)所示,待压缩数据经第一级 LZ77 压缩后,再分别经过第二级的动态、静态 Huffman 编码处理,最后选择压缩率较好的结果输出.

LZ77 是一种基于字典的数据无损压缩算法[7,8],通过使用索引替代重复出现的字符串达到压缩的目的,其流程如图 1(*b*)所示,哈希(Hash)函数被引入用来实现快速查找重复字符串:顺序计算每个字符串前 3 个字节(byte)的 Hash 值,根据 Hash 值构造、维护链表,并沿着链表顺序搜索匹配. 若匹配完成时,未找到重复字符串,则输出字符编码(LIT),反之则输出由指回距离(DIS)与匹配长度(LEN)构成的编码.



(*a*) Deflate算法两级处理流程



(*b*) LZ77压缩流程

图1　压缩算法流程

LZ77 压缩是一个循环迭代的过程,由于各个步骤间数据的依赖性,使得并行计算难以开展;加之 Hash 函数"多对一"的特征,不可避免地出现 Hash 值相同而字符串不同的"伪匹配"情况. "伪匹配"导致不必要的匹配比较工作将极大限制 LZ77 压缩处理速率. 此外,LZ77 匹配处理的执行效率也是影响 LZ77 压缩处理速率的关键因素.

Deflate 算法第二级包括动态、静态 Huffman 编码处理. 动态 Huffman 是一种基于频率统计的编码方式,其频率统计、基于频率编码的"两遍"处理需求限制了流水线结构的使用. 另外,动态 Huffman 编码过程中对存储资源需求大且利用率不高. 本文以提升无损压缩速率为主要目标,兼顾资源消耗,选择使用静态 Huffman 实现第二级数据无损压缩的方式. 以少量压缩率损失为代价,获得硬件资源的优化和压缩速率的极大提高.

静态 Huffman 编码根据预设编码表[6,9],如表 1 所示,对 LZ77 压缩输出数据(LEN、DIS、LIT)进行编码,并按 Gzip 文件格式封装发送. LZ77 压缩输出数据的组织形式也是影响编码速率的关键因素之一.

表 1　静态 Huffman 编码树

字符编号	编码位数	编码值
0 – 143	8	00110000 – 10111111
144 – 255	9	110010000 – 111111111
256 – 279	7	0000000 – 0010111
280 – 287	8	11000000 – 11000111

3　Gzip 数据无损压缩硬件加速方案

针对第 2 节所述各种影响 Gzip 无损压缩速率的关键问题,本文相应使用 4 种加速方法,以达到数据无损压缩硬件加速的目的.

3.1　双 Hash 函数提升 LZ77 匹配成功率

使用 Hash 函数构造链表时,3byte 变量与 15 位(bit)Hash 值"多对一"的映射关系会导致"伪匹配"情况的出现,降低字符串匹配的成功率. 本文通过使用 2 种不同 Hash 函数计算字符串前 3byte 的 Hash 值,当且仅当所得 Hash 值的指向地址相同时维护链表. 以此降低"伪匹配"出现概率,提高匹配成功率.

电路结构如图 2 所示,链表信息存储于链头存储器(Head_Ram1、Head_Ram2)和回溯存储器(Prev_Ram)中. Hash 计算模块,包含 2 个不同的 Hash 函数计算逻辑,可并行计算两路 Hash 值. 链表构造/维护模块根据两个 Hash 值相应地从 Head_Ram1、Head_Ram2 中获取两路 Hash 值各自的指向地址,当所指向的地址相同时维护链表,更新 Head_Ram1、Head_Ram2 和 Prev_Ram.

同时,将两路 Hash 值都指向的相同地址(Mat_Ad-

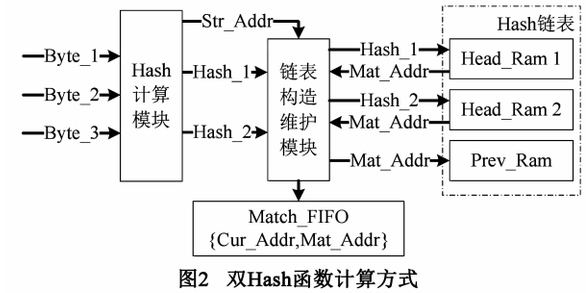


图2　双Hash函数计算方式

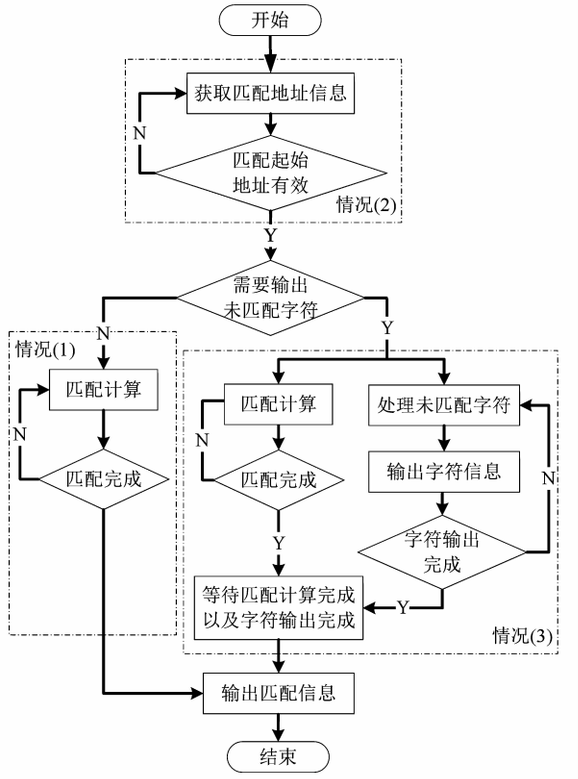
dr),和当前匹配起始地址(Cur_Addr)拼接存储于匹配地址信息缓冲区(Match_FIFO)中,使 Hash 计算与匹配计算相对独立,Hash 计算无需等待相应匹配计算完成即可进行.

使用双 Hash 函数判断是否维护链表,较传统单一 Hash 函数的方法,增加了一级验证方式,可有效降低"伪匹配"出现的概率,减少不必要的匹配工作;同时得益于硬件电路的并行性,极大地提升了无损压缩速率.

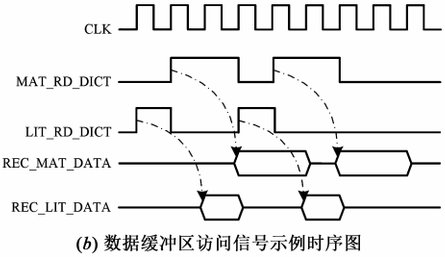
3.2 LZ77 数据匹配处理方法

LZ77 数据匹配计算,是沿链表回溯搜索匹配字符串,并逐对比较匹配的过程. 若匹配成功,择优输出 LEN+DIS 组合编码;反之,则输出 LIT.

匹配计算时,从 Match_FIFO 获取距离待处理字符地址(Next_Addr)最近的当前匹配起始地址(Cur_Addr),根据 Cur_Addr 与 Next_Addr 的关系分为 3 种情况分析:



(*a*) 匹配处理模块主要工作流程



(*b*) 数据缓冲区访问信号示例时序图

图3 LZ77匹配处理方法

(1) Cur_Addr = Next_Addr:表示待处理字符位置处恰巧可能存在匹配字符串,该情况出现概率极低,直接根据指向地址(Mat_Addr)取值匹配比较即可;

(2) Cur_Addr < Next_Addr:表示待处理字符位置超越了本次从 Match_FIFO 中获取的当前匹配起始字符串的位置,则再次从 Match_FIFO 中取值更新 Cur_Addr,直至该情况转变为情况 a 或情况 c 为止;

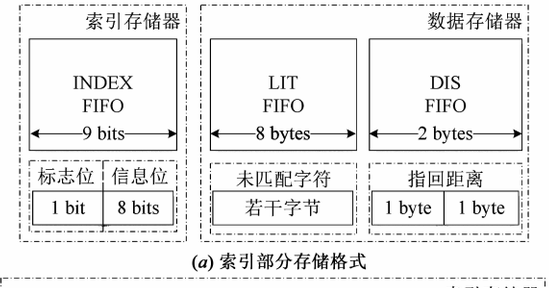
(3) Cur_Addr > Next_Addr:表示待处理字符位置尚未到达本次从 Match_FIFO 中当前匹配起始字符串的位置. 这是最普遍的一种情况,此时区间[Next_Addr, Cur_Addr)内的字符均为 LIT. 本文针对该情况,充分发挥硬件并行计算优势,提出了一种在匹配计算的同时输出 LIT 的匹配处理方法,流程图如图 3(*a*)所示.

由于匹配计算和输出 LIT 都需要访问数据缓冲区以获取待压缩数据,可能会导致读请求冲突. 为此,本文设计了读信号仲裁模块,时序示意如图 3(*b*)所示. 匹配计算模块具有较高的读请求(MAT_RD_DICT)优先级,而在匹配计算模块的处理间隙发送读信号(LIT_RD_DICT),获取 LIT(LIT_DATA). 可以节约获取 LIT 的时间,提升整个匹配处理过程的效率.

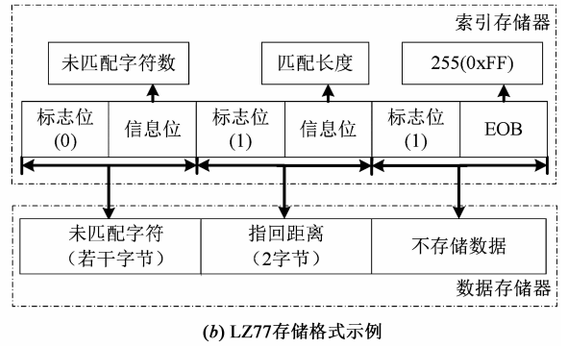
3.3 面向硬件存储的 LZ77 压缩存储格式

本文提出一种 LZ77 压缩结果的存储格式,以提高第一级 LZ77 的输出效率和第二级的 Huffman 编码效率.

LZ77 的压缩结果被拆分后存储在索引存储器和数据存储器中. 索引存储器(INDEX_FIFO),如图 4(*a*)所示,位宽为 9bits,标志位用以指示后续 8bits 信息位的功能:标志位为 1,信息位保存 LEN;标志位为 0,信息位保



(*a*) 索引部分存储格式



(*b*) LZ77存储格式示例

图4 LZ77压缩结果存储格式

存以 byte 为单位的 LIT 数量. 数据存储器包括字符存储器(LIT_FIFO)和指回距离存储器(DIS_FIFO),其中 LIT_FIFO 位宽为 64bits,一次读写操作可以处理 8bytes 的 LIT;DIS_FIFO 用于存储 DIS,其位宽为 2bytes.

存储 LZ77 压缩数据时,如图 4(*b*)所示,LEN、DIS、LIT 有独立的数据通路和存储空间,可以并行执行写操作,写操作过程中也无需添加任何指示数据类型的标志位. 并且,LIT_FIFO 与数据缓冲区位宽相同,可实现快速存储.

执行 Huffman 编码时,LIT 与 LEN 使用同一张编码表,DIS 使用另一张编码表. 传统编码流程是:逐个识别数据类型,根据类型使用相应的编码表编码. 而使用上述存储方式后,可一次识别若干连续数据类型,并且可以超前识别后续数据类型. 执行当前 Huffman 编码时,读取 INDEX_FIFO,若得到 LIT 字节数,便连续从 LIT_FIFO 中读取相应字节数的 LIT,等待下次 Huffman 编码;若获得 LEN,便从 DIS_FIFO 读取相应 DIS,等待下次 Huffman 编码.

3.4　高效数据拼接器

本文设计了一种数据拼接电路,来应对 Huffman 这种变长编码的存储和发送问题. 如图 5 所示,移位次数计算器获得并统计各 Huffman 变长编码的长度,并据此控制相应的编码移位器对 Huffman 变长编码进行移位. 最后,移位结果由数据拼接器完成拼接、字节翻转并输出. 数据拼接器每次输出为 64bit,数据溢出部分将被保存在溢出控制器中,作为下次拼接的一部分;该数据拼接电路与静态 Huffman 编码模块紧密耦合,以流水线方式工作,可有效提升 Huffman 编码结果发送速率.

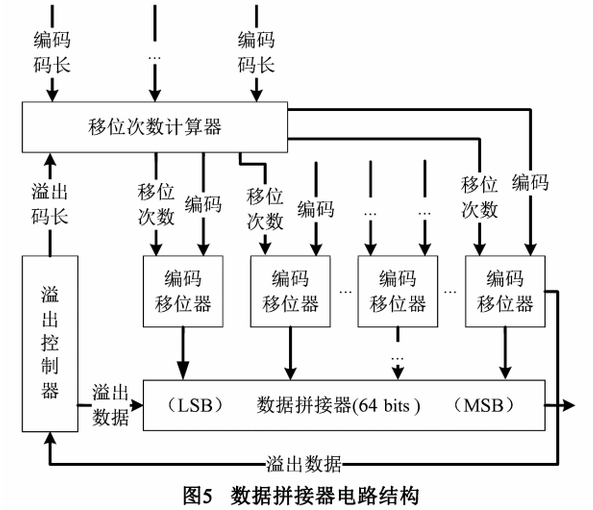


图5　数据拼接器电路结构

4　测试与性能

压缩率和压缩速率作为硬件压缩电路关键性能指标[2~5]除与压缩源自身特征有关外,还受多种压缩参数的影响. 对同一压缩源,若字典越大、回溯次数越多、最大匹配长度越长,则有可能获得更高的压缩率;但同时也意味着需要更多相关的计算周期. 压缩率和压缩速率这两项关键性能指标之间并没有明确的线性关系,但是欲获取更优的压缩率,则必须在压缩速率指标上做出牺牲. 测试时,针对各种测试源,设置不同的压缩参数进行测试比较,以期在获得适中的压缩率的同时,尽可能的优化压缩速率,从而完成整个 Gzip 无损压缩硬件实现工作.

4.1　设计测试平台

Xilinx ML605[10]是工业中被广泛采用的 FPGA 评估开发板之一,板上包括一片 Virtex6 240T FPGA 芯片,其内部包含 15 万个 6 输入查找表以及 14Mbit SRAM 资源,可以满足绝大部分数字集成电路开发验证的要求;其中 ML605 开发板所提供的 PCIe Gen2 X8 接口,也符合我们对数据传输速率的要求.

测试平台如图 6 所示,包含 Gzip 压缩电路和数据通路控制模块的测试板卡,通过 PCIe 接口接驳在 PC 上[11].

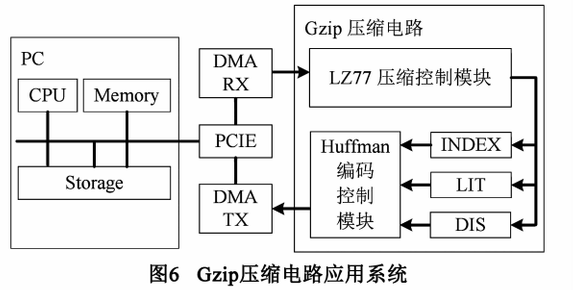


图6　Gzip压缩电路应用系统

平台的简单工作流程如下:板卡响应 PC 机发起压缩请求后,从 DMA_RX 电路接收 PC 机以 DMA 方式发送的待压缩数据. Gzip 压缩电路对数据进行压缩并发送至 DMA_TX 电路,同时发起 DMA 写请求,上位机响应请求并接收压缩数据.

4.2　标准压缩测试源

Calgary Corpus[12]是由加拿大卡尔加里大学在 1987 年创建并于 1990 年代开始被广泛应用于测试数据压缩算法的性能,成为评估无损数据压缩性能的标准测试源. 如表 2 所示,Calgary Corpus 由文本文件、二进制文件、程序源码、图像文件等 18 个文件组成,能够对压缩算法进行有效的评测.

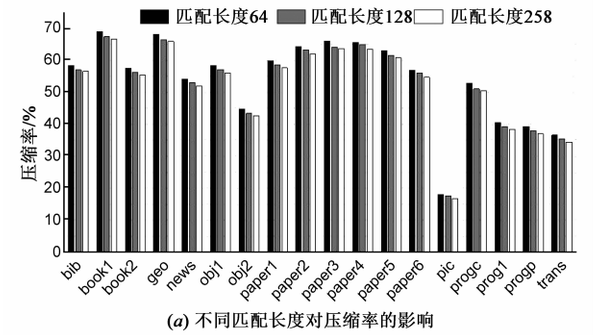
4.3　测试结果

设计中,首先设置固定的字典大小(32KB)和回溯次数(24),针对不同匹配长度,对标准压缩测试源进行测试评估;其结果如图 7(*a*)所示,对于各种类型的压缩源,增加最大匹配长度可获得更优的压缩率,但相对最

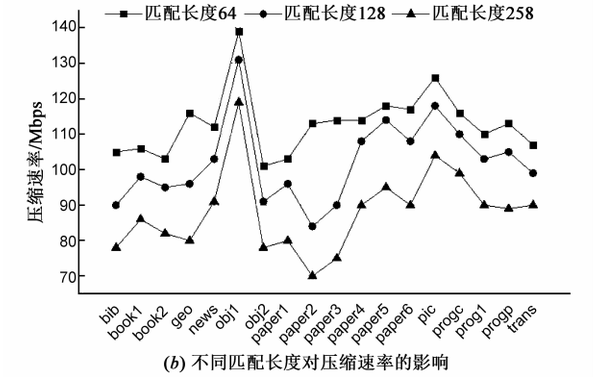
大匹配长度参数成倍增加,获得的压缩率指标的提升就显得微乎其微了;图 7(*b*)表示使用上述不同匹配长度参数时相应的压缩速率,较大的匹配长度需要消耗更多的时间来进行匹配串的查找计算,使得压缩速率的衰减较为明显,其最大压缩速率衰减达到了约 37%.

表 2 Calgary corpus 标准测试源

测试源	大小(KB)	测试源描述	测试源	大小(KB)	测试源描述
bib	109	"refer"文本	paper3	46	"troff"文本
book1	751	普通文本	paper4	13	"troff"文本
book2	597	"troff"文本	paper5	12	"troff"文本
geo	100	地图数据	paper6	38	"troff"文本
news	369	普通文本	pic	502	BMP 图片
obj1	21	可执行程序	progc	39	C 源码
obj2	242	可执行程序	progl	70	Lisp 源码
paper1	52	"troff"文本	progp	49	Pascal 源码
paper2	81	"troff"文本	trans	92	控制字符



(*a*) 不同匹配长度对压缩率的影响



(*b*) 不同匹配长度对压缩速率的影响

图7 最大匹配长度对压缩性能的影响

采用不同最大匹配长度参数、字典大小和回溯次数参数进行设计测试时,其数据如表 3 所示,较大的字典和回溯次数提高了发现更多更好匹配串的可能性,从而提高了压缩率性能. 各项压缩参数设置与软件实现方式相近时,可获得与其相当的平均压缩率(51.9%);但较大字典、较多回溯次数以及多次计算匹配都需要消耗更多的计算周期,明显地降低压缩速率,最低仅为 88.1Mbps,同测试中的最高速率(171.2Mbps)相比,衰减接近 50%.

综合考虑各个参数对压缩速率和压缩率的影响,使用如下参数设置:字典大小 16KB、回溯次数 8、最大匹配长度 64,与无损压缩软件实现方式相比,以较小压缩率损失为代价获得压缩速率性能指标的极大提升.

与软件进行性能对比测试时,分别在上述硬件测试平台和一台 CPU 为 Core i5-3470 CPU@3.2GHz 的 PC 机以及一台 CPU 为 Core i7-3632QM@2.2GHz 的笔记本上,对 Gzip 压缩硬件电路和 Gzip 压缩软件进行了对比测试. 测试结果如表 4 所示,本文所设计的 Gzip 压缩电路够在 Xilinx ML605 平台上以 125MHz 的核心频率稳定工作,其平均压缩速率达 171Mb/s,超过 PC 平台软件处理速率 8.4 倍、笔记本平台软件处理速率 9.5 倍.

表 3 字典大小和回溯次数对压缩性能的影响

字典大小(KB)	回溯次数	最大匹配长度	压缩率(%)	压缩速率(Mbps)
16	8	64	65.9	171.2
16	24	64	57.7	133.4
32	24	64	54.2	110.9
32	24	128	52.6	103.6
32	24	258	51.9	88.1

表 4 Gzip 压缩电路性能测试数据

测试源	压缩率(%)		压缩速率(Mbps)		
	软件	硬件	Core i5	Core i7	硬件
bib	32.1	71.6	16	15	172
book1	40.9	84.9	36	33	154
book2	33.8	74.9	38	31	172
geo	67.0	82.5	17	16	176
news	38.5	69.1	35	27	172
obj1	52.4	66.7	9	10	168
obj2	33.1	56.4	25	20	169
paper1	36.6	73.1	17	22	172
paper2	37.0	77.8	22	20	174
paper3	39.1	80.4	15	11	175
paper4	46.1	76.9	6	6	173
paper5	41.7	75.0	7	6	177
paper6	36.8	71.1	13	19	175
pic	11.2	22.9	39	28	194
progc	35.9	61.5	6	9	171
progl	22.8	48.6	21	18	168
progp	22.5	44.9	17	13	169
trans	20.7	47.8	28	23	151
平均	36.1	65.9	20.4	18.1	171.2

由于 Gzip 软件实现方式使用基于字符分布频率统计的动态 Huffman 编码,进而获得压缩率指标的提升,但动态 Huffman 编码除消耗大量的资源外[6],还限制了硬件并行计算优势的发挥,将降低整个系统的压缩速率. 故仅以静态 Huffman 编码为原型优化设计完成了 Gzip 无损压缩硬件模型,并获得了令人满意的压缩速率性能指标.

5 结论

本文基于硬件平台,提出了一种 Gzip 压缩硬件实

现方案,使用多种加速方法,通过发挥硬件并行计算和流水线处理优势,达到提升压缩速率的目的. 测试表明:

(1)双 Hash 函数结构的使用,能有效减少“伪匹配”数量,显著降低无效匹配次数,提高硬件压缩速率;

(2)使用高效 LZ77 匹配处理方法以及面向硬件优化的匹配结果输出存储方式,可以提高 LZ77 输出速率和 Huffman 编码效率;

(3)利用与编码模块紧耦合的数据拼接器,可以较好的解决变长编码的对齐调整、字节翻转和发送问题;

(4)不同的压缩参数会对压缩率和压缩速率带来不同的影响;针对不同的优化目的,选择相应的压缩参数配置;

(5)所设计的 Gzip 硬件压缩电路,使用的多种加速方法,在取得较高压缩速率(平均达 171Mb/s)的同时保证适中的压缩率,可满足实时压缩需求.

参考文献

[1] Salomon D. Data Compression: The Complete Reference [M]. London: Springer, 2007. 241 – 246.

[2] Harnik D, Khaitzin E, Sotnikov D, et al. A fast implementation of deflate[A]. Data Compression Conference Proceedings[C]. Snowbird: IEEE, 2014. 223 – 232.

[3] 汤晓东. 基于 LZ77 算法的数据无损压缩的硬件设计[D]. 南京:东南大学,2013.
Tang Xiao-dong. Hardware Design of Lossless Data Compression Based on LZ77 Algorithm[D]. Nanjing: Southeast University, 2013. (in Chinese)

[4] Ouyang J, Luo H, Wang Z L, et al. FPGA implementation of GZIP compression and decompression for IDC services [A]. 2010 International Conference on Field-Programmable Technology[C]. Beijing: IEEE Computer Society, 2010. 265 – 268.

[5] Fowers J, Kim J Y, Burger D, et al. A scalable high-bandwidth architecture for lossless compression on FPGAs[A]. 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines[C]. Vancouver: IEEE, 2015. 52 – 59.

[6] Rigler S. FPGA-Based Lossless Data Compression Using GNU Zip[D]. Canada: University of Waterloo, 2007.

[7] Ziv J, Lempel A. A universal algorithm for sequential data compression[J]. IEEE Transactions on Information Theory, 1977, IT-23(3): 337 – 343.

[8] Ziv J, Lempel A. Compression of individual sequences via variable-rate coding[J]. IEEE Trans Information Theory, 1978, IT-24(5): 530 – 536.

[9] Huffman D A. A method for the construction of minimum-redundancy codes[J]. Resonance, 2006, 11(2): 91 – 99.

[10] Xilinx. ML605 Hardware User Guide[EB/OL]. http://www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf, 2015-06-18.

[11] Xilinx. Bus Master Performance Demonstration Reference Design for the Xilinx Endpoint PCI Express Solutions [EB/OL]. http://www.xilinx.com/support/documentation/application_notes/xapp1052.pdf, 2015-06-18.

[12] Calgary Corpus. Calgary Corpus Database[EB/OL]. http://en.wikipedia.org/wiki/Calgary_Corpus, 2015-06-18.

作者简介

李　冰　男,1968 年生于江苏南京. 现为东南大学教授、博士生导师. 主要研究方向为大数据云计算以及高效、安全的信息集成电路系统.
E-mail: bernie_seu@seu.edu.cn

王超凡　男,1990 年生于陕西咸阳. 现为东南大学集成电路学院硕士研究生. 主要研究方向为无损数据压缩.
E-mail: 289082877@qq.com

顾　巍　男,1989 年生于江苏南京. 现为东南大学集成电路学院硕士研究生. 主要研究方向为无损数据压缩.
E-mail: weslay@seu.edu.cn

董　乾　男,1982 年生于江苏泰州. 现为东南大学集成电路学院博士研究生. 主要研究方向是高效数据处理硬件算法设计与验证.
E-mail: ic_qiand@seu.edu.cn