有限状态熵编码的 VLSI 设计与实现

黄海¹⁾, 邢琳²⁾, 那宁²⁾, 张国良³⁾, 赵石磊¹⁾, 刘志伟¹⁾

¹⁾ (哈尔滨理工大学软件与微电子学院 哈尔滨 150080)

²⁾ (哈尔滨理工大学计算机科学与技术学院 哈尔滨 150080)

³⁾ (中国航天科技集团有限公司第九研究院 北京 100094)

(ic@hrbust.edu.cn)

摘 要: 在处理海量数据时, 以软件方式实现的 Z 标准(Zstd)无损压缩算法难以满足特定应用领域对压缩速度的需求. 对 Zstd 进行硬件加速设计是解决这一问题的有效方案, 尤其是针对 Zstd 的有限状态熵编码(finite state entropy, FSE)的硬件加速. 因此, 提出一种适用于 Zstd 的 FSE 压缩、解压硬件实现架构, 采用固定压缩表实现最优的硬件加速步骤; 通过增加序列映射的硬件模块来降低存储空间并提高传输速度; 采用软硬件协同设计方案, 并对硬件实现架构进行 7 级流水设计. 通过 Visual Studio 与 Modelsim 的联合验证平台进行验证, 实验结果表明在 TSMC 55 nm 的工艺下, 系统最高频率可达到 750 MHz. 与软件实现相比, 整体压缩速度提高了 9 倍以上, 整体解压速度提高了约 100 倍.

关键词: 无损压缩算法; 有限状态熵编码; Z 标准; 硬件加速

中图分类号: TP391 **DOI:** 10.3724/SP.J.1089.2021.18575

Design and Implementation of VLSI for Finite State Entropy Encoding

Huang Hai¹⁾, Xing Lin²⁾, Na Ning²⁾, Zhang Guoliang³⁾, Zhao Shilei¹⁾, and Liu Zhiwei¹⁾

¹⁾ (School of Software and Microelectronics, Harbin University of Science and Technology, Harbin 150080)

²⁾ (School of Computer Science and Technology, Harbin University of Science and Technology, Harbin 150080)

³⁾ (Ninth Research Institute, China Aerospace Science and Technology Corporation, Beijing 100094)

Abstract: The Zstd (Zstandard) lossless compression algorithm that implemented by software is difficult to meet the demand of compression speed in specific application field when processing massive data. It is an effective solution to this problem by using the hardware acceleration scheme, especially for the hardware acceleration of FSE (finite state entropy). Thus, a hardware implementation of the compression and decompression in FSE is proposed for Zstd. This scheme determines the optimal hardware acceleration step by fixing the size of compression table, reduces the storage space and improves the transmission speed by adding hardware modules of sequence mapping, enhances the time of parallel processing by dividing the seven-stages of flow and realizes the architecture by software and hardware collaboration. The proposed architecture is implemented in the TSMC 55 nm process, and the highest frequency can reach 750 MHz. The experimental results show that compared with the software implementation, the speed of the whole compression is more than 9 times faster, the speed of the whole decompression is more than the 100 times faster.

收稿日期: 2020-08-13; 修回日期: 2020-11-12. 基金项目: 国家重点研发计划“光电子与微电子器件及集成”重点专项(2018YFB2202100); 黑龙江省自然科学基金优秀青年项目(YQ2019F010); 黑龙江省普通本科高等学校青年创新人才培养计划(UNPYSCT-2017081); 黑龙江省博士后科研启动基金(LBH-Q18065). 黄海(1982—), 男, 博士, 教授, 硕士生导师, CCF 会员, 主要研究方向为信息安全、数字信号处理、集成电路设计; 邢琳(1995—), 女, 硕士研究生, 主要研究方向为计算机网络、信息安全; 那宁(1995—), 男, 硕士研究生, 主要研究方向为信息安全、集成电路设计; 张国良(1987—), 男, 硕士, 高级工程师, 主要研究方向为集成电路设计; 赵石磊(1979—), 男, 博士, 副教授, 主要研究方向为信息安全、集成电路设计; 刘志伟(1987—), 男, 博士研究生, 主要研究方向为数字信号处理、信息安全.

Key words: lossless compression algorithm; finite state entropy encoding; Zstandard; hardware acceleration

随着大数据时代的来临,在物联网、人工智能等特定应用领域,对海量数据处理的低时延性要求不断提高,无损数据压缩技术也越来越重要.无损数据压缩按压缩原理可分为基于数据统计算法和基于字典的算法.其中,基于数据统计算法包括 Shannon-Fano 编码^[1-2], Huffman 编码^[3],算术编码(arithmetic coding, AC)^[4]、游程编码(run length code, RLE)^[5]和有限状态熵编码(finite state entropy, FSE)^[6]等;基于字典的算法包括 LZ77(Lempel-Ziv 77)编码^[7]和 LZ78(Lempel-Ziv 78)编码^[8]等.

为提高普适性,数据压缩方案中通常采用2种及以上的压缩算法进行混合压缩.快速无损压缩算法 Zstd(Zstandard)^[9]就是一种由 LZ77 编码, Huffman 编码和 FSE 组成的混合压缩算法.与其他压缩算法(如 Deflate 算法^[10], Bzip2 算法^[11], Brotli 算法^[12])相比, Zstd 具有更好的压缩性能.此外, Zstd 属于开源算法,提供了 22 个压缩等级,用于权衡压缩速度与压缩率,在 Linux 内核、FreeBSD 操作系统和 AWS Redshift 数据库等应用领域得到了广泛的应用.

基于软件实现方式的无损压缩技术具有较高的灵活性、普适性和低成本等优点,但软件执行方式只能顺序执行,导致中央处理器(central processing unit, CPU)在处理海量数据时资源长时间被占用,使得压缩速度大幅降低,难以满足特定应用领域对海量数据实时压缩处理的需求.采用硬件实现是解决上述问题的有效途径^[13-18],受益于硬件固有并行处理特点,可以达到提高传输速度、资源使用率和安全性的目的.

对 Zstd 中 3 个主要组成部分: LZ77, FSE 和 Huffman 进行统计分析,其压缩时间占比约为 4:1:1.其中 FSE 占比虽然不大,但对 Zstd 的性能影响较大.由于采用了 FSE,因此 Zstd 比其他混合压缩算法具有更好的压缩性能.此外,针对 LZ77 和 Huffman 编码的硬件加速方案较为成熟, FSE 作为一种新型的压缩算法,它既具有类似算数编码的精度,还具有 Huffman 编码的压缩速度,而且对于符号的重新编码可以精确到小数位,计算中不需要使用乘法和除法更新状态.因此,研究 FSE 的硬件加速架构对实现 Zstd 算法整体加速具有重要意义,也是满足特定应用领域需求的有效方法.

FSE 属于非对称数字系统(asymmetric numeral systems, ANS)^[19-20]中 tANS(table asymmetric nu-

meral systems)的一种熵编码.现有研究主要针对 tANS 和 ANS 的重要组成部分 uABS(uniform asymmetric binary systems),而鲜有对 FSE 硬件架构的研究.2016年, Yokoo^[21]提出了一种 uABS 熵编码,对 ANS 进行进一步优化.同年, Duda 等^[22]提出了一种基于 ANS 加密的轻量级压缩方法.文献[23]提出了一种 tANS 压缩硬件实现架构,该架构的性能取决于字母表中的符号数量,其压缩最高频率可达到 289 MHz.文献[24]中提出的 tANS 的解压硬件实现架构,其压缩频率可达到 125 MHz.文献[25]中的 tANS 解压的硬件实现方案的吞吐率可以达到高性能 Huffman 编码的 4 倍.

综上所述,为满足特定应用领域对压缩性能日益增长的需求,本文针对 Zstd 中的 FSE 展开研究,采用软硬件协同实现的方式,全面提高 Zstd 算法的压缩和解压的速度.按照 Zstd 规范标准,提出了适于硬件实现的 FSE 压缩和解压架构设计方案,从而有效地减少硬件开销和提高硬件利用率.由于 FSE 压缩和解压原理区别较大,在压缩过程中,本文对 FSE 进行 7 级流水线设计,使 FSE 具有更高的硬件复用率和并行处理能力;在解压过程中,被压缩数据之间具有强依赖性,难以实现流水线设计,采用具有 11 种状态的有限状态机来实现控制逻辑.对所设计硬件实现方案进行 Verilog 建模,通过 Visual Studio 与 Modelsim 的联合验证平台进行功能验证,并通过 EDA 工具得到了硬件实现的具体性能指标.实验结果表明,所提出的 FSE 硬件实现架构,对 Zstd 硬件加速设计与实现的解决方案具有理论研究的前瞻性、可行性,以及现实的应用价值.

1 FSE 在 Zstd 中的应用

Zstd 算法压缩流程如图 1 所示.压缩时将原数据初始化为不大于 128 KB 的多个数据块,依次对初始化后的数据块进行压缩.首先, Zstd 对每个数据块进行 LZ77 压缩,得到 2 个输出: Literals 和三元组序列(II, of, ml).其中, Literals 表示为所有未匹配的文本; II 表示未匹配的文本长度; of 表示距离匹配文本的偏移量; ml 表示匹配的文本长度.然后,根据相关判断对 Literals 进行 Huffman 编码压缩,对三元组序列和 Huffman 编码中压缩表的权重进行 FSE 压缩.最后,根据相关判断选择最优的

图 1 Zstd 算法的压缩流程

压缩结果构造出压缩后的 Block 数据块。

Zstd 的解压可以看做是压缩的逆过程, 根据压缩流 bitStream 中存储的信息进行解压。从图 1 可以看出, FSE 在压缩或解压过程中可能出现 Huffman 压缩表和 LZ77 压缩输出的三元组序列 2 个数据源。由于 Huffman 编码是通过查找压缩表实现, 其所对应的符号数最多为 256 个, 而处理的数据量较少, 适合采用软件方式实现。因此, FSE 对三元组序列进行压缩首先需要进行判断。若三元组序列中的各序列符号均为同一个数, 则采用 rle 模式, 直接保存该符号和符号出现的次数, 不进行 FSE 压缩; 若三元组序列可以使用上一次的压缩表进行压缩, 则进入 repeat 模式; 若三元组序列中的符号个数和所选择的模式等参数较少, 则进入 basic 模式, 采用默认 FSE 压缩表进行压缩。若以上 3 种情况均不满足, 则进入 compressed 模式, 采用实时建表进行 FSE 压缩。上述 4 种模式中, 前 3 种以判断操作为主适合采用软件方式实现。compressed 模式以数据处理为主, 适合进行硬件加速设计。在 compressed 模式下, FSE 压缩流程如图 2 所示。

Zstd 压缩、解压过程中, 首先对数据进行预处理, 即将三元组序列进行序列映射后保存到映射三元组序列(ll_code, of_code, ml_code)。其次, 对映射三元组序列中不同序列分别进行符号次数统计, 并将得到的统计次数进行归一化, 然后对预处理后的数据建立 FSE 压缩表。最后, 根据建立的 FSE 压缩表对映射三元组序列进行压缩。FSE 解压与压缩互为逆过程, 即对三元组序列分别建立解

图 2 FSE 压缩的主要步骤

压表后, 根据解压表进行序列解压。

2 FSE 压缩加速方案设计

为了便于硬件实现并充分发挥硬件加速的优势, 本文根据算法原理对压缩表大小进行了修改, 在对 FSE 压缩各步骤进行运行时间占比统计和分析的基础上, 进行精准加速设计。其特点如下:

(1) 为了充分发挥硬件实现的优势, 三元组序列的压缩表大小采用固定长度。由 FSE 的压缩原理可知, 随着压缩表大小的增加, 压缩率将会提高, 而建表时间将会增加。采用硬件加速可以弥补建表时间增加带来了性能损失。

(2) 压缩表大小确定后, 根据 FSE 压缩各步骤所需时间的统计结果确定应进行硬件加速的步骤。

(3) 为了提升硬件加速性能, 在 FSE 压缩硬件模块中增加了序列映射模块; 根据性能指标确定输入/输出的位宽, 以减少硬件资源的开销。由于静态随机存取存储器(static random-access memory, SRAM)具有速度快和随机读写的优点, 设计中的存储器选用 SRAM。SRAM 用于输入/输出数据和压缩表的存储。针对特定应用压缩表可预先经过训练得到并存入 SRAM 中, 工作时可以根据需要对压缩表进行更新维护, 以确保预先训练的压缩表的准确性以及安全性。

(4) 采用 7 级流水线设计对 FSE 压缩硬件加速架构进行设计。

2.1 FSE 压缩表大小设计

由 FSE 算法的原理可知, 压缩表大小的增大,

会带来建表时间的增加, 同时可以提高压缩精度。因此, 在设计时压缩表的大小不能小于原压缩表大小, 否则会使得精度不满足要求, 从而导致压缩失败。按照上述原则, 对 FSE 中三元组序列 ml, ll, of 的压缩表的大小进行了优化设计, 即将 ml, ll, of 的大小分别确定为 9, 9, 8。用 tableLog 表示压缩表的大小, 即表中元素的个数。

2.2 FSE 压缩运行时间占比统计

三元组序列各自的 tableLog 确定后, 按图 2 中的步骤分别对其进行运行时间统计。测试平台的操作系统为 Windows 10, 处理器为 i5-5200U CPU @2.20 GHz, 测试软件采用 Visual Studio, 测试数据集采用 Silesia 数据集^[26]中的 xml 子数据集。按图 2 中 FSE 压缩的 5 个步骤, 分别进行运行时间统计, 统计结果表明序列映射、符号次数统计、次数归一化、建压缩表、序列压缩各步骤运行时间比约为 160:66:1:4:1 915。可以看出, 序列压缩占总运行时间的 89%, 远大于其他 4 个操作步骤。此外对其他 4 个操作步骤进行硬件加速效果并不明显。因此, 本设计中采用软硬件结合的方式, 序列压缩采

用硬件加速实现, 其余步骤采用软件实现, 从而得到最优的 FSE 压缩方案。

2.3 FSE 压缩的整体架构

按照软硬件结合的方式进行设计, FSE 压缩的整体架构如图 3 所示, 包括 FSE 压缩软件和压缩硬件 2 个模块。其中压缩软件模块包括序列映射(软件部分)、符号次数统计、次数归一化和建 FSE 压缩表 4 个部分。压缩硬件模块包括序列映射(硬件部分, 用于加速)、序列压缩和输入、输出 SRAM。软硬件模块间数据传输采用高速串行计算机扩展总线标准(peripheral component interconnect express, PCIe)接口。数据压缩时, 首先, PC 将待压缩的数据通过 PCIe 传输到输入 SRAM; 其次, 将输入 SRAM 中的数据写入输入缓存, 并根据压缩标准进行压缩; 然后, 将压缩后的数据通过输出缓存写入输出 SRAM 中, 经比特拼接后输入到输出先入先出队列(first input first output, FIFO)中, 设计中 FIFO 的位宽采用 64 bit; 最后, 通过 PCIe 将输出 SRAM 的数据传入 PC, 用于后续的压缩文件打包。

图3 FSE 压缩的整体架构

在进行 FSE 压缩时, 输入的数据包括三元组序列和映射三元组序列。其中, 映射三元组序列是由三元组序列通过简单计算得到。设计中可以采用 2 种方式实现: 一种是增加一个序列映射(硬件)用于存储映射三元组序列; 另一种是采用重新计算三元组序列得到相应的映射三元组序列的方式, 这样将会减少存储空间并提高传输速度。考虑 FSE 之前的操作 LZ77 的输出为三元组序列而非映射三元组序列, 以及 Zstd 整体硬件加速实现的需要, 本设计采用第一种实现方式。可以避免使用额外的存储空间对软件计算得到的映射三元组序列进行存储。

根据 LZ77 的压缩原理以及 Zstd 实际需求, 对三元组各序列的位宽进行设计, 其中 ml, ll 序列的位宽确定为 16 bit, of 的位宽确定为 18 bit。三元组序列的存储最大为 32 kbit。为了减少输出时 SRAM 存取的次数, 输出 SRAM 的位宽确定为 64 bit, 即以 4 bit 为单位进行存储, 而非 8 bit。

2.4 序列压缩模块的 7 级流水设计

序列压缩时三元组序列之间数据没有相互依赖性, 可通过流水设计充分发挥硬件并行处理的优势, 最终采用如图 4 所示的 7 级流水线设计, 在 Zstd 中三元组序列按照逆序方式进行压缩。因此, 在第一个时钟周期 T 时, 三元组序列中最后一组数

据 nbseq 进入 Task1; 在 $2T$ 时, 三元组数据 nbseq 进入 Task2, 三元组序列中倒数第 2 组数据 nbseq-1 进入 Task1, 其中 T 为 1 个时钟周期, 其余以此类推. 每个 Task 的主要任务说明如下.

(1) Task1. 计算当前应处理三元组数据在序列中所处位置, 即当前处理三元组数据在序列中的排序位置. 由于压缩顺序为逆序, 所以首先从三元组序列个数减 1 位置开始, 每个周期依次递减 1, 直到处理完所有三元组序列.

(2) Task2. 按照得到的位置信息从 SRAM 中读取三元组序列数据值, 得到相应的三元组值; 重新计算该值得到相应的映射三元组, 即序列映射(硬件)部分.

(3) Task3. 根据映射三元组的信息读取 SRAM 相应的压缩表数据.

(4) Task4. 根据压缩表和相关数据信息, 计算映射三元组序列的新状态值、FSE 压缩后的最低有效位以及三元组序列映射时所需最小有效偏移位, 并将得到的压缩数据存储在变量 flush1 中. 在存储的过程中, 应判断是否为最后一组压缩的三元组.

(5) Task5. 调整 flush1 中数据的字节顺序, 并保存到变量 bitContainer 中, 计算该变量的有效位数和映射三元组对应 FSE 压缩的最低有效位.

(6) Task6. 当 bitContainer 的位数大于或等于 64 时, 将 SRAM 指针增加一位, 同时计算当前 bitStream 的总长度.

(7) Task7. 判断是否所有的数据都压缩完成, 若压缩完, 则进行结束流操作, 并在压缩流中添加用于解压时确定压缩流的实际位置的标识位.

图 4 序列压缩模块的 7 级流水示意图

3 FSE 解压加速方案设计

FSE 解压加速方案设计步骤与压缩类似. 在对 FSE 解压的各步骤进行时间占比统计的基础上, 分析统计结果, 依据统计结果进行方案设计, 主要包括以下几部分.

(1) 统计分析 FSE 解压的各步骤的进行时间, 并根据统计结果确定最优硬件加速的方案.

(2) 依据硬件资源的开销的原则, 确定 FSE 解压硬件模块的输入/输出的位宽, 例如确定 SRAM 的位宽.

(3) 按照控制要求设计 FSE 解压控制状态机. 由于被压缩数据之间存在依赖性, 无法采用流水结构, 需根据解压控制逻辑, 设计用于 FSE 解压的控制器.

3.1 FSE 解压运行时间占比统计分析

对 FSE 解压步骤的运行时间进行统计分析, 统计使用的测试环境与 FSE 压缩运行时间统计类似. 解压的数据是压缩时生成的压缩文件. 统计结果表明, 建立解压表的时间与序列解压的时间比约为 1:98, 可以看出序列解压占总运行时间约为 99%. 因此, 本设计中采用软硬件结合的方式, 序列解压采用硬件加速实现, 建立解压表采用软件实现, 从而得到最优的 FSE 解压方案.

3.2 FSE 解压的整体架构设计

FSE 解压的整体架构如图 5 所示. 建立 FSE 解压表在 PC 上采用软件实现, 序列解压采用硬件实现. 首先, 序列解压数据由 PC 通过 PCIe 传输到输入 SRAM; 其次, 将输入 SRAM 中的压缩流 bitStream 写入输入缓存进行数据解压, 解压后的数据由输出缓存写入输出 SRAM. 数据经比特拼接后输出到 FIFO 中, 考虑三元组序列的位宽总数, 设计中 FIFO 的位宽为 50 bit. 其中, [15: 0] 为 ll, [33: 16] 为 of, [49: 34] 为 ml. 最后, 通过 PCIe 将解压后数据传入 PC, 得到三元组序列以用于后续数据解压. 压缩流 bitStream 的 SRAM 的位宽设定与压缩模块中相同, 为 64 bit. 压缩流 bitStream 是以字节形式存储, 其总位数是 8 的倍数, 有可能不能整除 64. 考虑 bitStream 是逆序读取, 当 SRAM 中 bitStream 的总长度不是 64 的倍数时, 在原 bitStream 的末尾的低位需要补 $8 \times (8 - \text{bitStream 的字节数} \% 8)$ bit 的 0, 用以保证数据读取的正确性.

3.3 序列解压模块的状态机设计

序列解压模块的状态机示意图如图 6 所示. 该状态机包括 11 个状态: IDLE, INIT, INIT_STM,

图5 FSE解压的整体架构

INIT_STA, RBSTRM, RBSTRM0, RBSTRM1, LDBITS, CMPTLL, STATCHG 和 FINISH.

(1) IDLE空闲状态. 当fse_decode_enable信号为高电平时, 序列解压开始. 该状态主要是进行初始化计算, 结束后跳转到INIT状态. 例如, 计算变量bitD_bitConsumed的值, 该变量表示缓冲数据变量bitD_bContainer(用于缓冲压缩流bitStream的数据, 即待解压的数据流, 以低字节优先方式存储)多少已被解压.

(2) INIT读取压缩流bitStream地址信号. 该状态下, 计算bitD_bContainer, bit_pos(当前缓冲区已解压字节数, 即缓冲区中无效数据的字节数)和bitD_ptr(当前读取bitStream的位置值).

(3) INIT_STM状态用于计算去掉压缩时增加的标识位的bitStream. 该状态下对bitStream进行了修改, 因此需要对相应的变量重新计算, 然后跳转到INIT_STA状态.

(4) INIT_STA状态用于计算三元组各序列的解压状态, 然后跳转到RBSTRM状态.

(5) RBSTRM状态用于判断是否更新bitD_bContainer的值, 然后跳转到RBSTRM0状态.

(6) RBSTRM0状态下更新bitD_bContainer, bitD_bitConsumed和bit_pos, 然后跳转到RBSTRM1状态.

(7) RBSTRM1状态根据计算的三元组的解压状态读取SRAM中的数据, 用loadResult表示读取的结果. loadResult的值为0时表示未读取完; 值为1时表示缓冲流结束, 但未读取完; 值为2时表示读取完成; 值为3时表示读取溢出. ml和of可以利用缓冲区的数据进行解压, 但ll可能无法利用缓冲区进行解压. 因此, 需要通过loadll信号对是否继续读取bitStream进行判断. 综上所述, 当loadResult ≤ 2 且loadll信号为低电平时, 跳转到LDBITS状态; loadll信号为高电平时, 跳转到CMPTLL状态; 当loadResult > 2 时, 跳转到FINISH状态.

(8) LDBITS状态用于读取解压表的数据. 该状态下, 计算loadll的值, 并更新三元组中of和ml和bitD_bitConsumed的值. loadll的值为低电平时跳转到CMPTLL状态, 其值为高电平时跳转到RBSTRM状态.

(9) CMPTLL状态用于计算三元组中ll的值, 并更新bitD_bitConsumed等值; 然后跳转到STATCHG状态.

(10) STATCHG状态用于存储解压的三元组数据, 并更新三元组的状态. 每解压完一组三元组数据, SRAM指针一次增加1位. 当seq_val > 0 时, 表示未解压完, 则跳转到RBSTRM状态; seq_val=0时, 跳转到FINISH状态.

(11) FINISH表示解压结束.

图6 序列解压模块的状态机

4 性能分析

采用Visual Studio 2017和Modelsim的联合验证平台对所设计的系统进行功能验证和性能分析. 实验结果表明, 该方案能够满足特定应用领域对压缩速度的需求.

4.1 测试数据集及指标

测试集对测试结果有较大的影响. 因此选择一个具有代表性、普遍性和公开性的数据集十分重要. 传统的数据集(Calgary Corpus^[27], Canterbury Corpus^[28])的数据库类型和文件大小不能较好地满足当前的应用环境. Silesia数据集涵盖多种常用的文件类型, 符合当前的绝大多数应用领域的实际情况. 该数据集包括4类文件类型, 即文本、图像、数据库和可执行文件, 包括12种数据子集, 即dickens, mozilla, mr, nci, office, osdb, reymont, samba, sao, webster, xml, x-ray.

选取压缩速度和解压速度作为系统的测试指标. 压缩速度是指平均每秒能够压缩的文件大小,

解压速度则是指平均每秒能够解压还原的文件大小, 它们的单位为 MB/s. 常见的压缩速度和解压速度的计算式分别为

$$S_c = \frac{F}{T_c} \times 100 \quad (1)$$

$$S_d = \frac{F}{T_d} \times 100 \quad (2)$$

其中, F 为源文件大小; T_c 为压缩时间; S_c 为压缩速度; T_d 为解压时间; S_d 为解压速度.

4.2 测试平台与环境

为确保验证平台的正确性和公平性, 同时兼顾实验数据比较分析的高效性, 本文性能比较是在相同的数据源下, 通过自动化的方式进行 txt 文件对比. 采用的验证平台示意图如图 7 所示. 全部数据测试是在 Windows 10 操作系统、i5-5 200U CPU @2.20 GHz 处理器下, 使用 Visual Studio 完成的. Zstd 的版本是 Zstd v1.3.7, 优化选取的相关参数, 对所设计硬件架构与软件实现进行压缩率分析, 根据式(1)(2)分别计算压缩和解压速度.

图 7 验证平台示意图

4.3 功能仿真

图 8 和图 9 所示分别为序列压缩、序列解压的功能仿真结果. 图 8 中的 saddr_u16_offndsta 和 sdout_u16_mlfindsta 分别表示 of 序列和 ml 序列的 deltaFindState 在 SRAM 中的地址总线和存储的数据; swdout 表示写入 SRAM 序列压缩输出的暂存数据. 图 9 中的 myseq_addr 表示输出三元组序列写入 SRAM 的地址总线信号; myseq_data 表示输出三元组序列写入 SRAM 中存储的数据.

图 8 序列压缩的部分波形

图 9 序列解压的部分波形

从图 8 和图 9 可以看出, 本文设计的序列压缩和序列解压的结果与直接采用软件实现的 Zstd 结果一致, 功能正确.

4.4 压缩、解压性能分析

Zstd 压缩时首先将文件切片, 每片最大 128 KB. 测试时选取 Silesia 数据集中各子数据集的中间 128 KB 的文件进行性能测试. 在 TSMC 55 nm 的工艺下, 所设计的序列压缩模块和序列解压模块的最高频率均可以达到 750 MHz, 而文献[23]方法的压缩频率最高为 289 MHz, 文献[24]的解压频率最高为 125 MHz. 从硬件实现最高频率角度看, 本文设计的硬件架构优于文献[23]和文献[24]的架构.

本文设计的方案与软件实现方案对比结果如

表 1 所示. 其中, 软件实现的压缩速度 S_{c_soft} 和解压速度 S_{d_soft} 均为 Zstd v1.3.7 版本下测试生成. 在测试过程中, 对 Zstd 使用的参数进行最优化选取, 具体参数为压缩等级选取 1; 窗口大小 windowLog 选取 17; 输出的哈希值位宽 hashLog 选取 14; 进行哈希变换的字节数 searchLength 选取 4; ml, ll 和 of 的 tableLog 分别选取 9, 9, 8. S_{c_hard} 和 S_{d_hard} 分别表示本文采用硬件加速实现方案的压缩速度和解压速度. 由于 PCIe 上所消耗的通信时间与其他时间相比占比较小, 在 S_{c_hard} 和 S_{d_hard} 中均未计入. 此外, S_{c_hard} 为在上述软件实现的基础上将软件序列压缩时间改为硬件压缩时间, S_{d_hard} 和 S_{c_hard} 计算类似. 从表 1 可以看出, 本文所设计的方案与软件实现相

表 1 本文设计的方案与软件实现方案对比

编号	文件名	文件类型	$S_{c_soft}/(\text{MB} \cdot \text{s}^{-1})$	$S_{c_hard}/(\text{MB} \cdot \text{s}^{-1})$	$\frac{S_{c_hard}}{S_{c_soft}}/\%$	$S_{d_soft}/(\text{MB} \cdot \text{s}^{-1})$	$S_{d_hard}/(\text{MB} \cdot \text{s}^{-1})$	$\frac{S_{d_hard}}{S_{d_soft}}/\%$
1	dickens	文本	6.91	62.81	9.09	4.99	498.87	99.97
2	mozilla	可执行文件	2.69	24.45	9.09	14.58	1 458.37	100.03
3	mr	图像	3.82	34.75	9.10	3.80	379.89	99.97
4	nci	数据库	3.14	28.53	9.09	4.05	404.78	99.95
5	ooffice	可执行文件	5.44	49.43	9.09	5.48	547.54	99.92
6	osdb	数据库	7.37	67.00	9.09	6.60	659.92	99.99
7	reymont	文本	4.59	41.75	9.10	3.94	393.56	99.89
8	samba	可执行文件	5.30	48.20	9.09	3.44	343.67	99.90
9	sao	数据库	2.46	22.40	9.11	4.31	430.78	99.95
10	webster	文本	4.29	38.96	9.08	4.16	415.88	99.97
11	xml	文本	2.33	21.21	9.10	4.11	410.71	99.93
12	x-ray	图像	3.53	32.09	9.09	3.44	343.93	99.98

比, 对于所有类型的测试数据整体压缩速度提高了 9 倍以上, 整体解压速度提高了约 100 倍. 综上所述, 所设计的实现方案性能得到大幅度提高, 可以满足特定应用领域对压缩海量数据的要求.

5 结 语

本文针对 Zstd 中的 FSE 提出了适合硬件实现压缩和解压的架构设计方案, 并采用软硬件协同设计, 实现处理数据的最优化. 在 FSE 压缩方案中的序列映射、符号次数统计、次数归一化和建压缩表由软件实现, 序列压缩由硬件实现. 在 FSE 解压方案中, 建解压表由软件实现, 序列解压由硬件实现. 由于序列解压的不同步骤间使用的数据存在依赖性, 需要通过状态机实现. 除此之外, 通过对相关算法原理进行分析, 计算得到相应的位宽. 序列压缩和序列解压的输入/输出数据均采用 SRAM 进行存储. 在实际应用中, 可以将经过训练得到的固定压缩表和解压表存储在 SRAM 中, 在一定周期内对它们进行更新维护, 以得到更好的压缩率. 所设计的硬件架构, 在 TSMC 55nm 的工艺下, 压缩和解压的最高频率均可达到 750MHz, 远高于现有硬件实现架构. 所设计的软硬件协同实现方案与软件实现方案相比, 整体压缩速度提高 9 倍以上; 整体解压速度提高约 100 倍. 综上所述, 所设计的硬件架构和软硬件协调实现方案各方面性能优越, 可以满足特定应用领域对性能的要求, 对 Zstd 整体硬件加速方案实现具有重要的指导意义和借鉴作用.

在后续的研究工作中, 将进一步探讨面向特点应用需求的最优化设计方案. 例如, 利用人工智

能技术训练得到特定领域的 FSE 最优定制化压缩表和解压表, 通过固化压缩表和解压表进一步提升 FSE 的性能. 对 Zstd 的其他组成部分 LZ77, Huffman 编码进行硬件加速设计, 最终实现 Zstd 算法的整体硬件加速实现方案.

参考文献(References):

- [1] Chattopadhyay S, Chattopadhyay G. Conjugate gradient descent learned ANN for Indian summer monsoon rainfall and efficiency assessment through Shannon-Fano coding[J]. *Journal of Atmospheric and Solar-Terrestrial Physics*, 2018, 179: 202-205
- [2] Reddy M R, Akshaya K, Infanta Seles R A I, *et al.* Image compression using Shannon-Fano-Elias coding and run length encoding[C]//*Proceedings of the 2nd International Conference on Inventive Communication and Computational Technologies*. Los Alamitos: IEEE Computer Society Press, 2018: Article No.8473120
- [3] Wang W, Zhang W. Huffman coding-based adaptive spatial modulation[J]. *IEEE Transactions on Wireless Communications*, 2017, 16(8): 5090-5101
- [4] Saarinen M J O. Arithmetic coding and blinding countermeasures for lattice signatures[J]. *Journal of Cryptographic Engineering*, 2018, 8(1): 71-84
- [5] Li B J, Zhang C G, Li B, *et al.* A hardware-efficient parallel architecture for real-time blob analysis based on run-length code[J]. *Journal of Real-Time Image Processing*, 2018, 15(3): 657-672
- [6] Collet Y. Finite state entropy[OL]. [2020-08-13]. <https://github.com/Cyan4973/FiniteStateEntropy>
- [7] Policriti A, Prezza N. LZ77 computation based on the run-length encoded BWT[J]. *Algorithmica*, 2018, 80(7): 1986-2011
- [8] Badkobeh G, Gagie T, Inenaga S, *et al.* On two LZ78-style grammars: compression bounds and compressed-space compu-

- tation[C] //Proceedings of the 24th International Symposium on String Processing and Information Retrieval. Heidelberg: Springer, 2017: 51-67
- [9] Facebook. Zstandard [OL]. [2020-08-13]. <https://github.com/facebook/zstd/releases/tag/v1.4.0>
- [10] Xue Y, Tan Y A, Liang C, *et al.* An optimized data hiding scheme for Deflate codes[J]. *Soft Computing*, 2018, 22(13): 4445-4455
- [11] Qiao W K, Fang Z M, Chang M C F, *et al.* An FPGA-based BWT accelerator for Bzip2 data compression[C] //Proceedings of the 27th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines. Los Alamitos: IEEE Computer Society Press, 2019: 96-99
- [12] Google. Brotli[OL]. [2020-08-13]. <https://github.com/google/brotli>
- [13] Li Bing, Wang Chaofan, Gu Wei, *et al.* Hardware-accelerated circuit design for Gzip compression[J]. *Acta Electronica Sinica*, 2017, 45(3): 540-545(in Chinese)
(李冰, 王超凡, 顾巍, 等. Gzip 压缩的硬件加速电路设计[J]. *电子学报*, 2017, 45(3): 540-545)
- [14] Xu Jinfu, Liu Lu, Li Wei, *et al.* A new lossless compression algorithm based on array configuration speedup model[J]. *Journal of Electronics & Information Technology*, 2018, 40(6): 1492-1498(in Chinese)
(徐金甫, 刘露, 李伟, 等. 一种基于阵列配置加速比模型的无损压缩算法[J]. *电子与信息学报*, 2018, 40(6): 1492-1498)
- [15] Xiao Jian, Hong Cong, Zhang Liang, *et al.* A high speed LZW compression algorithm hardware implementation based on FPGA[J]. *Microelectronics*, 2018, 40(6): 1492-1498(in Chinese)
(肖建, 洪聪, 张粮, 等. 一种高速 LZW 压缩算法 FPGA 硬件实现[J]. *微电子学*, 2019, 49(06): 824-828)
- [16] Zhu Jia, Liu Hongxia. Lossless high compression ratio circuit design[J]. *Journal of Xidian University: Natural Science*, 2019, 46(2): 35-40(in Chinese)
(朱嘉, 刘红侠. 无损高压压缩率电路设计[J]. *西安电子科技大学学报: 自然科学版*, 2019, 46(2): 35-40)
- [17] Huang Zhiyong, Wu Haihua, Yu Zhi, *et al.* Method of network compression and hardware acceleration based on Tiny-yolo[J]. *Journal of South China University of Technology: Natural Science Edition*, 2019, 47(6): 51-56(in Chinese)
(黄智勇, 吴海华, 虞智, 等. 基于 Tiny-yolo 的网络压缩与硬件加速方法[J]. *华南理工大学学报: 自然科学版*, 2019, 47(6): 51-56)
- [18] Dixit M M, Vijaya C. Modelling and hardware implementation of quantization levels of digital cameras in DCT based image compression[J]. *Engineering Science and Technology, an International Journal*, 2019, 22(3): 840-853
- [19] Duda J. Asymmetric numeral systems as close to capacity low state entropy coders[OL]. [2020-08-13]. https://www.researchgate.net/publication/258374370_Asymmetric_numeral_systems_as_close_to_capacity_low_state_entropy_coders
- [20] Duda J, Tahboub K, Gadgil N J, *et al.* The use of asymmetric numeral systems as an accurate replacement for Huffman coding[C] //Proceedings of the 31st Picture Coding Symposium. Los Alamitos: IEEE Computer Society Press, 2015: 65-69
- [21] Yokoo H. On the stationary distribution of asymmetric binary systems[C] //Proceedings of the IEEE International Symposium on Information Theory. Los Alamitos: IEEE Computer Society Press, 2016: 11-15
- [22] Duda J, Niemiec M. Lightweight compression with encryption based on asymmetric numeral systems[OL]. [2020-08-13]. <https://arxiv.org/abs/1612.04662>
- [23] Najmabadi S M, Wang Z, Baroud Y, *et al.* High throughput hardware architectures for asymmetric numeral systems entropy coding[C] //Proceedings of the 9th International Symposium on Image and Signal Processing and Analysis. Los Alamitos: IEEE Computer Society Press, 2015: 256-259
- [24] Najmabadi S M, Tungal H S, Tran T H, *et al.* Hardware-based architecture for asymmetric numeral systems entropy decoder[OL] [2020-08-13]. <https://ieeexplore.ieee.org/document/8122109>
- [25] Najmabadi S M, Tran T H, Eissa S, *et al.* An architecture for asymmetric numeral systems entropy decoder-a comparison with a canonical Huffman decoder[J]. *Journal of Signal Processing Systems*, 2019, 91(7): 805-817
- [26] Deorowicz S. Silesia corpus[OL]. [2020-08-13]. <http://sun.aei.polsl.pl/~sdeor/index.php?page=silesia>
- [27] Bell T C, Witten I H, Cleary J G. Modeling for text compression[J]. *ACM Computing Surveys*, 1989, 21(4): 557-591
- [28] Arnold R, Bell T. A corpus for the evaluation of lossless compression algorithms[C] //Proceedings of Conference on Data Compression. Los Alamitos: IEEE Computer Society Press, 1997: 201-210