计 算 机 学 报
CHINESE JOURNAL OF COMPUTERS

一种基于 FPGA 加速的高性能数据解压方法

刘谱光[1)] 魏子令[1)] 黄成龙[2)] 陈曙晖[1)]

[1)](国防科技大学计算机学院 长沙 410073)

[2)](军事科学院国防科技创新研究院人工智能研究中心 北京 100166)

摘 要 在数据库、深度学习、高效存储等数据读取性能敏感的应用场景中，数据解压性能对上层应用的服务质量有着重要影响。LZ4 无损数据压缩算法具备高速解压特性，因此被广泛应用在高速解压场景中，但其运行需要消耗大量 CPU 资源。为减少 LZ4 数据解压开销，学界和业界提出了基于 FPGA 的 LZ4 数据解压加速方法。但现有方法大多采用逐字节顺序处理的计算模式，导致并行度和吞吐率存在较大不足。因此，设计实现高性能 LZ4 数据解压加速方法成为当前研究亟需解决的关键问题。以 LZ4 解压的高性能加速为目标，本文研究从多层次对 LZ4 解压进行并行加速设计，提出了一种基于 FPGA 加速的高性能 LZ4 数据解压方法。首先，本方法研究对 LZ4 序列解析过程进行并行化改进，设计实现了一个基于多字段并行解析方法的并行化序列解析器，将吞吐率从每周期单字节扩展到每周期多字节。此外，本方法对序列解析器中的高时延长度字段解析逻辑进行优化改进，设计了基于二分法的最大匹配长度快速解析方法，显著减小序列解析器的关键路径时延，使得改进后的设计时钟频率比改进前提高了约 21%。其次，基于并行化序列解析器，本方法设计实现了一个高性能数据解压引擎。该引擎将序列解析与数据还原过程进行解耦设计，对解压输出数据通路进行扩展，解决了解压过程中输入输出吞吐率不匹配的问题。最后，为进一步提高吞吐率性能，本方法提出了可扩展多引擎数据解压加速器设计，并实现了一个基于 CPU-FPGA 架构的异构端到端数据解压加速系统原型。实验分析表明，本方法提出的数据解压引擎的每周期吞吐量是现有研究的 4.1~6.8 倍。该引擎实现了约 1.7 GB/s 的解压吞吐率，达到现有研究的 2.6~6.6 倍。系统原型的端到端测试和资源使用评估结果表明，本方法提出的数据解压加速系统在吞吐率和资源使用方面具备良好的可扩展性。在功耗效率方面，集成 8 引擎的解压加速系统原型的功效比是软件加速方法的 1.6 倍以上。

关键词 数据解压加速；并行化设计；可编程逻辑阵列（FPGA）；LZ4 算法

中图法分类号 TP302

An FPGA-Accelerated High-Performance Data Decompression Method

LIU Pu-Guang[1)] WEI Zi-Ling[1)] HUANG Cheng-Long[2)] CHEN Shu-Hui[1)]

[1)](College of Computer Science and Technology, National University of Defense Technology, Changsha 410073)

[2)](Artificial Intelligence Innovation Center, National Innovation Institute of Defense Technology, Academy of Military Sciences, Beijing 100166)

Abstract In scenarios where data read performance is crucial, such as databases, deep learning, and efficient storage, the performance of data decompression greatly affects the quality of service for higher-level applications. The LZ4 lossless data compression algorithm is known for its high-speed decompression characteristic, making it a popular choice in scenarios that require rapid decompression. However, LZ4 compression places a significant burden on CPU resources. To mitigate the overhead of LZ4 data decompression, academia and industry have proposed FPGA-based acceleration methods for LZ4 decompression. However, the majority of existing methods

本课题得到国家自然科学基金(No. 62202486, 61972412, U22B2005, 12102468)、国防科技大学校科研项目(No. ZK21-02)资助. **刘谱光**，男，博士研究生，中国计算机学会(CCF)学生会员，主要研究领域为大数据处理加速和网络空间安全. E-mail: liupuguang12@nudt.edu.cn. **魏子令**，男，博士，副研究员，中国计算机学会(CCF)会员，主要研究领域为网络空间安全监测和网络优化等. **黄成龙**，男，博士，助理研究员，主要研究领域为存内计算等. **陈曙晖**(**通信作者**)，男，研究员，博士生导师，中国计算机学会(CCF)会员，主要研究领域为网络空间安全监测和高性能网络流量处理. E-mail: shchen@nudt.edu.cn.

utilize a byte-by-byte sequential processing approach, which severely limits parallelism and throughput. Consequently, developing high-performance methods for accelerating LZ4 data decompression is a critical challenge in current research. Our research aims to achieve high-performance LZ4 decompression acceleration. We investigate the parallel acceleration of LZ4 decompression at multiple processing levels and propose an FPGA-accelerated method for high-performance LZ4 data decompression. Initially, we enhance the parallelization of the LZ4 sequence parsing process by developing a multi-field parallel parser that enables processing multiple bytes per cycle instead of one byte per cycle. Moreover, we optimize the long-delay logic for parsing the length field in the sequence parser. We introduce a dichotomous-based parsing method for rapidly determining the maximum match length. This approach substantially decreases the critical path delay of the sequence parser and enhances the design clock frequency by approximately 21%. Subsequently, we develop a high-performance data decompression engine based on the parallel sequence parser. This engine separates the sequence parsing and data recovery processes and expands the decompression output data path to resolve the input-output throughput mismatch during decompression. Finally, to enhance the throughput performance, we introduce a scalable data decompression accelerator with multiple engines. Additionally, we construct a prototype of a heterogeneous end-to-end data decompression acceleration system based on CPU-FPGA architecture. The experimental analysis demonstrates that the decompression engine proposed in this paper outperforms existing methods in terms of throughput, processing parallelism, and memory resource efficiency. The byte-per-cycle throughput of our decompression engine is 4.1 to 6.8 times higher than that reported in previous studies. Our decompression engine achieves a decompression throughput of approximately 1.7 GB/s, representing a 2.6 to 6.6 times improvement compared to previous studies. Our engine achieves remarkable memory efficiency, with a throughput of 114.5 MB/s per BRAM, surpassing the best existing method by a factor of 2.5. The experimental results of the end-to-end throughput and resource usage evaluation of our system prototype demonstrate the excellent scalability of our data decompression acceleration system in terms of throughput performance and resource utilization. The throughput and resource consumption of the system exhibit linear scalability with the number of engines. In addition, due to the faster growth of throughput compared to resource consumption, the resource-performance cost-benefit ratio of the accelerator progressively improves with an increasing number of engines. Regarding power efficiency, the prototype system demonstrates an increase in power efficiency with the number of engines. The decompression acceleration system, integrated with 8 engines, achieves more than a 1.6× increase in power efficiency compared to the software-based acceleration method.

Key words Data decompression acceleration; parallelization design; field-programmable gate array (FPGA); LZ4 algorithm

1 引言

在当前的大数据时代，海量数据中蕴含着巨大价值，推动了深度学习、大数据等数据密集型应用的快速发展，但同时超大数据规模给数据处理带来了巨大挑战。在处理前对数据进行无损压缩可有效降低存储开销、提高处理效率[1-2]。在数据库、深度学习、高效存储等数据高频读取应用中，数据解压性能将显著影响数据读取性能，进而决定上层应用服务质量[3-4]。因此，LZ4 等具备高解压吞吐率的无损压缩算法被广泛应用在高速解压应用场景中，但此类算法也不可避免会产生大量 CPU 开销[5-7]。随着摩尔定律的放缓，CPU 性能成本在快速增加。此外，在移动计算、物联网、星载等资源受限场景中，CPU 资源只能有限使用。因此，研究对无损数据解压进行高效加速以降低计算成本具有重要意义[4]。

为降低 CPU 开销，大量学界和业界研究开始关注基于可编程逻辑阵列（FPGA, Field-Programmable Gate Array）的高效硬件专用加速器（Domain-Specific Accelerator）[3-4,8-9]。一方面，相比于 CPU，FPGA 加速器可以通过大量的并行流

水设计实现更高的处理性能。另一方面，基于硬件设计的可重构性，FPGA 设计比专用集成电路（ASIC, Application Specific Integrated Circuits）具备更好的灵活性和适应性[1, 10]。

现有研究提出了若干基于 FPGA 的无损解压加速方法，主要围绕 LZ4、Deflate 和 Snappy 等基于 LZ77 算法的无损压缩算法进行加速设计[11-14]。其中，LZ4 算法相比其他 LZ77 系列算法实现了更高的压缩和解压吞吐率，然而其数据编码规则并不利于解压加速设计。在应用场景中，LZ4 解压加速方法的输入数据为 LZ4 压缩数据构成的单个解压任务或多个解压任务组成的任务流。LZ4 压缩数据中的基本编码元素称为序列（sequence），每个序列包含多个不同的信息字段[5]。LZ4 序列之间以及各个字段之间都存在数据依赖性，严重限制了解压过程的可并行性，使得 LZ4 解压的高性能加速成为当前研究面临的重要挑战。

Bartik M 等人在 2015 年首次提出了基于 FPGA 的 LZ4 压缩加速方法，其硬件设计思路与 LZ4 软件实现基本一致，但该研究没有实现解压加速[15]。Liu W 等人提出了首个 LZ4 解压加速设计，该研究通过修改序列格式来减少压缩数据输出的时延波动[16]。但该研究没有实现并行化设计，其解压吞吐率仅为 1 字节每时钟周期（简称周期）。Liu P 等人最近提出了应用于网络流量压缩的 LZ4 硬件加速方法，实现了现有 LZ4 解压加速引擎设计中的最高吞吐率和存储效率[11]。产业界也提出了针对 LZ4 的加速方案。Xilinx 提出了基于高层次综合技术的数据压缩解压库，其 LZ4 解压设计可支持部分并行处理[12]。该设计对未匹配数据解析和匹配数据还原过程进行了一定的并行优化，但其序列解析设计仍局限于每周期逐字段解析。现有设计较低的并行化程度无法发挥 FPGA 加速的并行处理优势，因此难以实现高性能解压目标。

针对 LZ4 解压的高性能加速难题，本文围绕高性能解压加速的设计目标，研究从字段解析层面开始自底向上对 LZ4 解压进行多层次并行加速设计（图 1）。在序列解析层面，本文设计了一个基于多字段并行解析方法的并行化序列解析器，并对其中的高时延长度字段解析逻辑进行了优化改进。对于单个解压任务的处理，本文研究将 LZ4 解压流程中的序列解析与数据还原过程解耦，优化解压数据流处理通路，设计了一个高性能数据解压引擎。为实现解压性能扩展性，本文提出了支持多任务并行处理的可扩展多引擎解压加速器，并实现了一个基于 CPU-FPGA 异构架构的端到端解压加速系统原型。

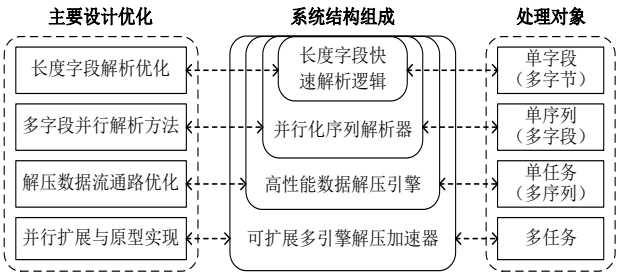


图 1 多层次解压加速优化设计逻辑图

本文主要贡献概括如下：

（1）基于多字段并行解析方法的并行化序列解析器设计。基于多字段并行解析方法，该解析器可实现每周期多字段的序列解析，最快可在一周期内完成单个序列的解析。此外，本文对解析器关键路径中的长度字段解析逻辑进行了优化改进，将整体设计时钟频率提高了约 21%；

（2）高性能数据解压引擎设计。针对数据解压过程中输入输出吞吐率不一致的特性，该解压引擎对序列解析与数据还原过程进行解耦设计，并对解压输出数据通路进行扩展，以支持高吞吐率的数据解压处理；

（3）可扩展的多引擎解压加速器设计。该设计可支持多任务多引擎并行处理，提供可线性扩展的数据解压性能。基于多引擎加速器，本文设计实现了一个基于 CPU-FPGA 架构的异构端到端解压加速系统原型，对本文设计进行了验证测试；

（4）实验评估表明，本文解压引擎实现了当前最高的处理并行度（6.8 字节/周期）和吞吐率（1.7 GB/s），分别达到现有方法的 4.1~6.8 倍和 2.6~6.6 倍。系统原型测试表明多引擎加速器具备良好的性能扩展性，8 引擎加速系统的功效比达到了标准软件的 1.8 倍和软件加速方法的 1.6 倍以上。

本文第 2 节先介绍问题背景，包括 LZ4 算法基本原理和解压加速挑战；第 3 节对核心的并行化序列解析器的设计优化进行详细阐述；第 4 节介绍高性能解压加速引擎的设计和并行扩展，包括数据解压引擎及多引擎解压加速器的设计；第 5 节对解压加速引擎进行多方面对比分析，并基于系统原型评估多引擎解压加速性能；第 6 节概述数据解压加速相关工作；第 7 节对全文进行总结并展望下一步的研究方向。

2 问题背景

2.1 LZ77压缩算法原理及LZ4压缩数据格式

无损数据压缩旨在完整保留数据信息的条件下对原始数据进行重组编码以降低数据冗余、减少数据量[17]。LZ77 算法是一种基于字典的无损压缩算法[18]，其基本原理是：将一定范围的历史输入数据作为查询重复数据的字典，在发现当前待处理数据与字典中数据匹配后，用较小的指向重复数据的字典索引替换当前数据，以实现对数据的压缩。

LZ4 无损压缩算法是 LZ77 算法的一种变体。LZ4 算法中使用哈希表快速查找数据是否存在于字典，每当找到重复内容后，即将此重复内容（可压缩数据）与在此内容前输入的未发现重复的内容（不可压缩数据）作为一组进行编码，编码后的字符串称为序列。如图 2 所示，一个序列由多个字段组成。每个序列中必须包含两个定长字段：令牌（token，简称 T）和匹配位置偏移（offset，简称 O）。序列中可能包含三种不定长字段：未压缩字符长度（literal length，简称 LL）、未压缩字符（literal，简称 L）、匹配字符长度（match length，简称 ML）。多个字段按以令牌为首的固定顺序排列，每个字段都包含数据解压的必要信息。

令牌	未压缩字符长度	未压缩字符	匹配位置偏移	匹配字符长度
1字节	⌈(N-14)/255⌉字节	N字节	2字节	⌈(M-19)/255⌉字节

图 2 LZ4 序列格式

LZ4 序列中，T 字段长度为 1 字节，前 4 比特（记为 TL）和后 4 比特（记为 TM）分别记录了不同信息。TL 和 LL 字段共同用于保存未压缩字符长度信息，当总的未压缩字符长度值（N）小于 15 字节时，仅用 TL 即可保存 N；当 N≥15 时，TL 不足以表示完整的未压缩长度，需要 LL 字段用于保存更多的未压缩长度信息，每 1 字节 LL 可表示 255 长度，LL 总长度为⌈(N-14)/255⌉。TM 和 ML 字段共同用于保存匹配字符长度信息，当实际匹配长度值（M）小于 19 字节时，仅用 TM 即可保存匹配字符长度信息①；当 M≥19 时，需要使用 ML 保存更多的匹配长度信息。L 字段表示原始的未压缩字符，即压缩过程中未发现重复的数据，当 N=0 时序列中不存在 L 字段。O 表示当前匹配数据与字典中匹配数据之间的地址偏移，用于数据解压时索引字典中对应的匹配数据进行数据还原。

2.2 LZ4解压加速挑战

LZ4 解压算法的目的是对输入的 LZ4 压缩数据按照数据编码规则进行解码，输出压缩前的原始数据。一份压缩后的 LZ4 数据（即一个解压任务）一般由多个序列组成。LZ4 解压时按顺序对每个序列进行处理，先对序列进行解析得到未压缩字符和匹配长度及偏移等匹配信息，再根据匹配信息从字典中读取输出原匹配数据。其中，对单个序列的解析是 LZ4 解压算法的核心，其流程如图 3 所示。

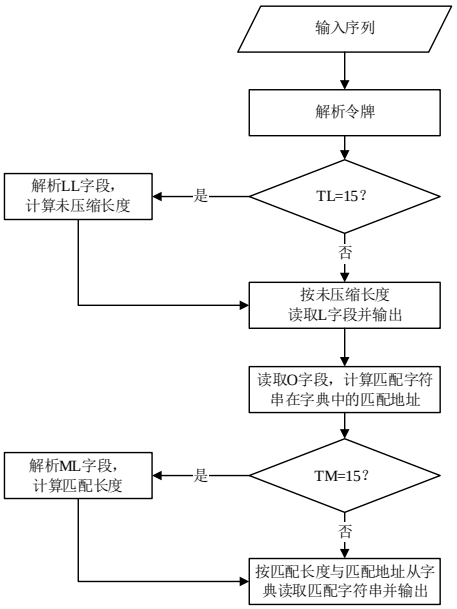


图 3 LZ4 序列解析流程

序列中各字段间的逻辑关联严重影响了序列解析效率。首先，必须在解析 T 后才能确定是否需要解析 LL 和 ML。其次，在解析完 T 和 LL 得到未压缩长度 N 后，才能进一步读取正确长度的 L 字段。同理，O 和 ML 必须在读取完 L 之后进行解析。由于各字段的位置和长度必须在前一字段解析完后才能确定，因此一般按照 T→LL→L→O→ML 的顺序依次进行解析处理以还原正确的原始数据。综上可知，单个序列的字段解析过程存在着严重的数据依赖性。此外，由于序列是一种长度不确定的编码，因此必须在处理完当前序列后才能开始处理下一序列。可见序列间的处理也存在着数据依赖性，所以很难对多个序列进行并行处理。以上两个层次的数据依赖性限制了 LZ4 解压的可并行性，使得 LZ4 解压的高性能加速成为当前研究面临的重要挑战。本文将从多层次克服 LZ4 解压加速面临的并行化处理难题，设计实现高性能的 LZ4 解压加速方法。

① LZ4 以字节为最小单元进行数据处理，压缩要求的最小匹配长度为 4 字节，因此仅需记录实际匹配长度值（M）减去 4 的差值即可。

3 并行化序列解析器

序列解析器是 LZ4 数据解压设计中的核心模块。序列解析器的性能是决定 LZ4 数据解压性能的关键因素，但序列解析流程中的数据依赖性严重限制了解析器性能。本文针对并行序列解析难题，研究设计了并行化序列解析器。该解析器通过多字段并行解析方法实现并行序列解析处理，成倍提高了吞吐率性能。此外，本文对并行化解析器中的高时延长度字段解析逻辑进行设计优化，减小了数据处理时延、提高了设计时钟频率。

3.1 并行序列解析方法设计

3.1.1 传统单字节序列解析方法

现有基于 FPGA 的 LZ4 解压加速研究没有突破解压过程的数据依赖性限制，大多采用与 LZ4 软件相同的处理方法，即按周期逐字节对压缩数据进行处理。传统方法的序列解析过程可以用图 4 所示的状态机描述。该状态机共包含 TKN、LLEN、LIT、OFS、MLEN、R_DICT 等 6 个状态，对应序列中 5 个字段的解析以及匹配数据读取过程。

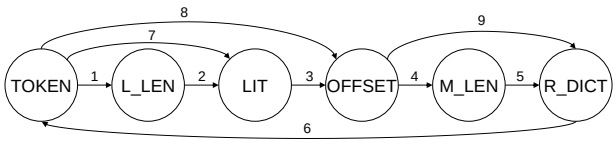


图 4 传统序列解析器状态转换图

该状态机的执行过程以及各个状态的具体含义如下：

TKN：解析 T 字段，读取 TL 和 TM，需要 1 个周期。若 TL=15，则需要继续解析 LL 字段，转到 LLEN 状态（路径 1）；当 0<TL<15 时，下一状态解析 L，转到 LIT 状态（路径 7）；当 TL=0 时，不需要解析 L，转到 OFS 状态（路径 8）。

LLEN：解析 LL 字段，计算完整未压缩字符长度 N，每周期解析 1 字节，共需要⌈(N-14)/255⌉个周期。解析完毕后，转到 LIT 状态（路径 2）。

LIT：解析 L 字段，输出未压缩字符串，每周期解析 1 字节，则需要 N 个周期。解析完毕后，转到 OFS 状态（路径 3）。

OFS：解析 O 字段，计算匹配数据在字典中的匹配地址，需要 2 个周期。当 TM=15 时，需要继续解析 ML，转到 MLEN 状态（路径 4）；当 TM<15 时，不存在 ML，该序列解析完毕，转到 R_DICT 状态（路径 9）。

MLEN：解析 ML 字段，计算完整匹配长度 M，每周期解析 1 字节，共需要⌈(M-19)/255⌉周期。解析完毕后，该序列解析完毕，转到 R_DICT 状态（路径 5）。

R_DICT：根据匹配长度 M 和匹配地址从字典中读取匹配数据并输出，每周期输出 1 字节，则需要 M 周期。匹配数据读取完毕后，转到 TKN 状态开始解析下一个序列（路径 6）。

传统方法中每周期逐字节的处理模式限定了解压设计的吞吐率性能。在输入吞吐率为 1 字节/周期时，输出吞吐率最大只能达到 CR 字节/周期（CR 为压缩率，原始数据量与压缩后数据量之比）。因此传统解压加速方法难以发挥出硬件设计的并行处理优势。

3.1.2 多字段并行解析方法

传统序列解析器的逐字节的处理模式导致其序列解析吞吐率仅为 1 字节/周期。在序列解析器设计中，本文提出多字段并行解析方法，实现每周期对多个字段进行并行处理，最快可在一个周期完成一个序列的解析，成倍提高序列解析器解析效率。

本方法首先将序列解析器每周期的处理对象从单字节扩展为一个包含多字节的处理窗口（窗口大小记为 P 字节），以提高序列解析器的性能上限。之后，本方法根据处理窗口首字节所属字段将解析器处理状态分为 TKN、LLEN、LIT、OFS、MLEN 共 5 种状态。与单字节解析方法不同，该方法在每个状态对多个字段进行并行处理。并行解析的字段数量决定序列解析器序列解析的并行度（每周期可解析的字节数）。由于字段间的依赖性，每个状态下处理窗口内的字段组合并不确定。本方法设计对多种字段组合进行并行处理以实现多字段的并行解析。其中，设计进行并行处理的字段数量将决定字段组合种类的数量。字段组合的数量越多，需要对应实现的并行处理逻辑也会越多，进而消耗更多的硬件逻辑资源。因此，本方法提出有限并行解析原则以避免复杂的并行处理带来过多的资源消耗。具体来说，该原则将并行解析字段的数量限制在 3 个以内，且最多包含 1 个不定长字段。本方法对每个状态下的字段组合及判定条件进行深度分析，确定可解析字段组合、对应的解析判定条件以及状态跳转，设计多字段并行解析状态转移逻辑。

以 TKN 态为例简要分析多字段并行解析设计流程。如图 5 所示，从处理窗口第 1 字节（下文以 B[i]表示处理窗口第 i 字节）对应的 T 字段开始进

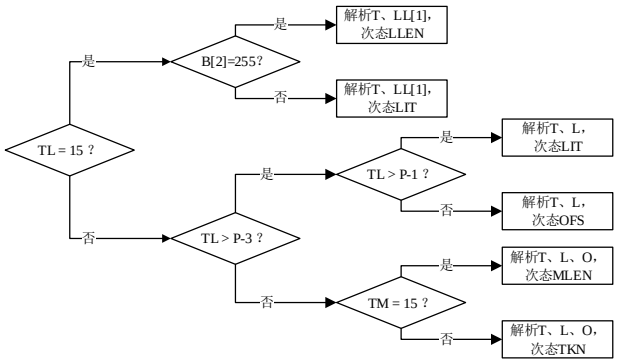


图 5 多字段组合解析判定分析过程（TKN 态）

行分析。当 TL=15 时，则说明 B[2]为 LL 字段，若 B[2]=255，说明后续字节仍为 LL 字段。由于 LL 为不定长字段，根据有限并行解析原则，不再解析后续字节，下一周期跳转到 LLEN 态处理。此情况下并行解析的判定条件为 TL=15 且 B[2]=255，解析的字段包括 T 和 LL 第 1 字节（LL[1]）共 2 字节（表 1 第 3 行）。若 TL<15，则后续字段为 L 和 O。当 TL≤P-3 时，说明 1 字节的 T、TL 字节的 L、2 字节的 O 三个字段的长度之和小于等于 P，即三者可同时在处理窗口内解析，此情况下解析的字段包括 T、L、O 共 TL+3 字节，下一状态取决于 TM 值。当 TM=15 时，需要解析 ML 字段，则下一状态为 MLEN（表 1 第 7 行）。TKN 态下其他多字段组合解析情况分析与以上过程类似。如图 5 所示，TKN 态共包含 6 种解析情况，需要并行处理的解析情况数量随着并行解析的字段数增大而指数增加。

多字段并行解析方法的详细状态转移设计如表 1 所示，其中包括每种解析情况对应的解析判定条件、解析字段组合、解析字节长度、次态，共包含 17 种字段组合解析情况及 15 条状态转移路径。多字段并行解析方法可显著提高解析器的序列解析速度。基于多字段并行解析方法设计的并行化序列解析器对应的状态转移图如图 6 所示。在 TKN 态和 OFS 态下，解析器最短只需要一个周期即可回到相同状态，完成一次序列解析。

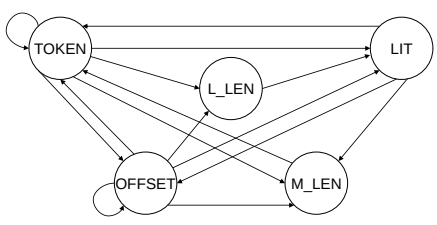


图 6 并行化序列解析器状态转换图

3.2 序列解析器硬件设计优化

3.2.1 多层次并行解析器设计

表 1 多字段并行解析状态转移表

现态	判定条件	解析字段	解析长度	次态
TKN	TL ≤ P-3 & TM < 15	T, L, O	TL+3	TKN
	TL = 15 & B[2] = 255	T, LL[1]	2	LLEN
	TL = 15 & B[2] < 255	T, LL[1]	2	LIT
	P-3 < TL ≤ P-1	T, L	TL+1	OFS
	TL < 15 & TL>P-1	T, L	P	LIT
	TL ≤ P-3 & TM = 15	T, L, O	TL+3	MLEN
LLEN	B[P] < 255	LL	DB	LIT
LIT	N ≤ P-2 & TM < 15	L, O	N+2	TKN
	P-2 < N ≤ P	L	N	OFS
	N ≤ P-2 & TM = 15	L, O	N+2	MLEN
OFS	TM = 15 & B[3] < 255	O, ML[1]	3	TKN
	TM < 15 & TL = 15 & B[4] = 255	O, T, LL[1]	4	LLEN
	TM < 15 & TL = 15 & B[4] < 255	O, T, LL[1]	4	LIT
	TM < 15 & P-3 < TL < 15	O, T, L	P	LIT
	TM < 15 & TL ≤ P-3	O, T, L	TL+3	OFS
	TM = 15 & B[3] = 255	O, ML[1]	3	MLEN
MLEN	B[P] < 255	ML	DB	TKN

并行化序列解析器的硬件设计充分发挥了硬件并行处理优势，实现多状态多字段多层次并行处理。本文设计的并行化序列解析器架构如图 7 上部分所示。并行化解析器主要由多个状态解析处理单元和一个状态控制输出模块组成。解析器中各个状态解析处理单元并行运行进行对应状态的多字段解析处理，生成预选次态及输出信息。状态控制输出模块包括一个状态寄存器和一个输出选择器。输出选择器根据寄存器中记录的现态选择对应状态解析处理单元的预处理信息输出，同时选择次态输入到状态寄存器。每一周期，序列解析器以处理窗口数据作为输入，经过多字段并行解析，输出解析后的未压缩信息和匹配信息。其中，未压缩信息包括未压缩字符片段、未压缩字符片段长度和累计未压缩字符长度，匹配信息包括累计匹配字符长度和匹配位置偏移。状态解析处理单元层面的并行化设计可有效提高解析处理效率，避免增加过多处理时延。

在状态解析处理单元的内部结构中，字段组合解析过程采用多字段多条件并行解析设计，对选择判定条件进行并行处理，同时对多字段组合进行并行信息提取，降低状态解析单元的处理时延。以 TKN 态为例，其对应的状态解析处理单元设计如图 7 下部分所示。首先，该单元根据 TKN 态下多种可能的序列组成结构从处理窗口读取与判定条件和预输出解析信息相关的字段。根据表 1 中给出的 TKN 态状态处理规则，需要读取的判定条件相关字

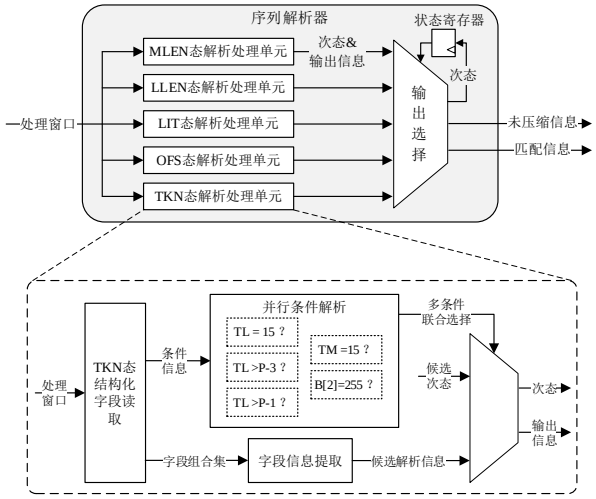


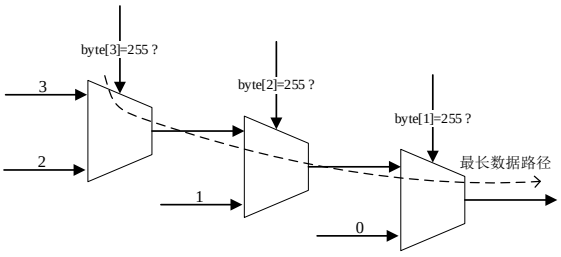
图 7 并行化序列解析器架构及状态解析处理单元结构

段包括 TL、TM、B[2]，输出解析信息相关字段包括 T、L、O、LL[1]。之后，并行条件解析模块对多个状态判定条件进行并行处理，得到一组条件判定结果。同时，字段信息提取模块对多个字段组合进行并行解析，提取对应解析信息，包括未压缩信息和匹配信息。最后，状态解析处理单元将条件判定结果组合作为选择器输入信号，选择器根据选择信号输出对应字段解析信息和次态。与 TKN 态解析处理单元相同，所有状态解析处理单元的多字段解析过程都采用了并行化处理，进一步提高了处理效率，并且不会增加过多的额外处理时延。

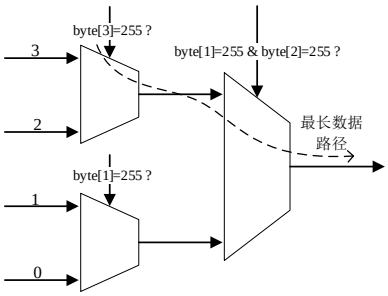
3.2.2 长度字段快速解析方法

多字段并行解析方法可成倍提高序列解析器的吞吐率，但更复杂的逻辑设计也不可避免会增加解析处理时间。在时序电路设计中，寄存器之间的最长数据处理路径（关键路径）对硬件设计的最高时钟频率（简称频率）和吞吐率有着重要影响。在并行化序列解析器中，长度字段解析逻辑由多级逻辑电路组成，其时延在关键路径时延中占比最大，成为限制解析器性能的主要因素。为提高解析器频率性能，本文对长度字段解析逻辑进行优化改进，设计了基于二分选择结构的最大匹配长度快速解析方法，可有效降低关键路径时延。

长度字段解析是 LLEN 态和 MLEN 态解析处理单元的核心功能。如表 1 所示，在 LLEN 态和 MLEN 态中，解析器需要对 LL 和 ML 两个长度字段进行并行化解析，通过长度字段解析功能计算出对应的字段解析长度（decoded bytes, DB）。该功能在状态解析处理单元的字段信息提取模块中实现。具体来说，长度字段解析需要计算给定长度字符串之中从第 1 字节开始字节值连续等于 255 的字节个数，在此称为最大匹配长度（maximum match length，MML）。假设一个 4 字节字符串为“255|255|24|255”，其前 2 个字节值连续为 255，第 3 字节不是，则该字符串的 MML 为 2。常规的长度字段解析方法一般采用从第 1 字节开始逐字节匹配选择的思路来计算 MML，其硬件结构由多个选择器串型连接组成。图 8(a)展示一个输入为 3 字节的常规 MML 计算设计，其中每个选择器根据对应字节的匹配结果选择输出匹配长度值。仅当前 m 个字节匹配而第 m+1 个字节不匹配时，MML 计算单元会输出匹配长度值 m。数据处理时延方面，对于 n 字节的问题规模，传统 MML 计算设计中的最长数据处理路径（从字符串输入到匹配长度输出）包括 n 个选择器，其处理时延随输入字节串长度的增大而直线增加。



(a) 串型逐级选择（最长路径包括 3 个选择器）



(b) 二分选择（最长路径包括 2 个选择器）

图 8 输入为 3 字节的 MML 计算设计结构对比

为降低 MML 计算的数据时延，本文对 MML 计算逻辑进行改进，提出了基于二分选择结构的 MML 快速计算设计。MML 快速计算设计的基本原理是对问题进行分解并对分解后的问题进行并行处理。具体来说，本设计将目标字符串从中间二分为长度相等的两个子串，先计算前半子串是否完全匹配，即所有字节值为 255。若前半子串完全匹配，对应选择器选择输出后半子串的 MML 计算结果；若前半子串不完全匹配，则选择输出前半子串的 MML 计算结果。子串的 MML 计算过程再继续采用二分选择法，反复迭代二分直至单字节匹配计算。上述过程中，每一级迭代中的匹配条件计算与

结果选择过程可并行运行，因此该设计的最长数据处理路径取决于迭代级数。一个输入为 3 字节的 MML 快速计算设计如图 8(b)所示，其中最长数据处理路径仅包括 2 个选择器。对于 n 字节 MML 计算，MML 快速计算设计需要的比较器数量与常规设计相同，但设计中最长数据处理路径包含的比较器数量相比常规设计从 n 个降低到 $log_2(n+1)$个，长度字段解析的时间复杂度可从 $O(n)$ 降低到 $O(log(n))$。因此，本文设计的基于二分选择结构的 MML 快速解析设计可有效降低长度字段解析逻辑的处理时延，提高并行化序列解析器的处理性能。

4 高性能解压引擎设计和并行扩展

并行化序列解析器可进行高速序列解析，提取出序列中包含的未压缩信息和匹配信息。在完成序列解析后，需要根据匹配信息还原匹配数据。本文基于并行化序列解析器设计了一个高性能数据解压引擎，实现数据解压全过程流水化处理。该引擎采用松耦合结构，将序列解析阶段与数据读写阶段解耦，实现高吞吐率流水化数据解压。此外，本文设计了支持多任务并行处理的可扩展多引擎解压加速器，并实现了基于 CPU-FPGA 架构的异构数据解压加速系统原型。

4.1 基于松耦合架构的高性能数据解压引擎

4.1.1 数据解压引擎架构设计

数据解压处理架构对解压性能有着重要影响。一些现有研究采用同步处理模式进行序列解析和匹配数据读取，导致其数据处理效率较低。此外，现有方法中输入输出带宽相同的设计并不适合输出数据量大于输入数据量的实际解压过程，输出数据流的反压将导致实际输入吞吐率低于设计输入带宽。针对以上问题，本文提出基于松耦合架构的高性能数据解压引擎设计，对数据解压各过程和输入输出数据通路进行解耦和改进，主要在数据处理模式、数据通路设计、信息传输通道等方面进行了设计优化。高性能数据解压引擎架构如图 9 灰色区域所示。

在数据处理模式方面，针对同步处理模式效率较低的问题，该引擎将解压数据处理过程解耦，分为序列解析和数据读写两个功能独立的部分，采用异步处理模式进行数据处理。序列解析部分和数据读写部分之间没有直接逻辑依赖，通过数据传输通道进行解压信息的传递，两部分可同时运行以实现处理效率最大化。具体来说，序列解析部分负责对输入压缩数据进行序列解析，输出匹配信息和未压缩信息并写入传输通道，其性能对应解压引擎的输入吞吐率。数据读写部分从信息传输通道中将匹配信息和未压缩信息读出，根据匹配信息将匹配数据从字典中读出，再与未压缩数据合并后输出，其性能对应解压引擎的输出吞吐率。

在数据通路设计方面，解压过程中输出吞吐率大于输入吞吐率的数据流特性决定了解压引擎的吞吐率上限取决于输出带宽大小。因此，本文对解压引擎的数据输出通路（数据读写部分）进行扩展，以更大的输出带宽支持更大的实际输出吞吐率。对于输入通路带宽，本文提出非对称输入输出通路设计，根据输出带宽和数据压缩率特征配置合适的输入带宽值。解压引擎通过信息传输通道协调输入输出数据流。

本文解压引擎中信息传输通道基于异步 FIFO 实现，在调节数据流和硬件高效设计中起到重要作用。在连续的序列解析数据还原过程中，各序列的

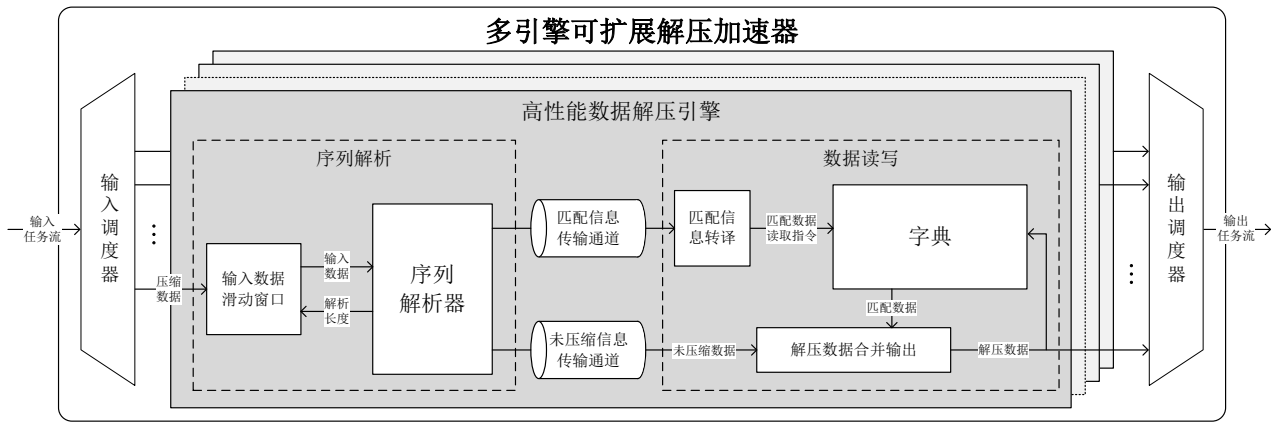


图 9 可扩展多引擎解压加速器和高性能数据解压引擎架构

压缩率并不完全一致，导致输入序列解析过程和数据读写输出过程处理的数据量在不断变化。压缩率更高的序列对应的解压数据长度更大，因此需要更多的数据读写时间。对此，信息传输通道负责将序列信息缓存，以调节和稳定输入输出吞吐率。此外，异步 FIFO 可支持跨时钟域的信息传递。因此，序列解析部分和数据读写部分可根据设计需求分别采用不同的时钟频率实现，优化解压引擎整体资源性能效率。

4.1.2 引擎内部模块设计实现

如图 9 所示，数据解压引擎由多个子模块组成，包括输入数据滑动窗口、序列解析器、两个信息传输通道、匹配信息转译、解压数据合并输出、字典等模块。其中核心模块为序列解析器和字典，分别负责序列解析和历史数据存储。其余子模块主要负责数据流控制和字典读写控制。

序列解析部分由序列解析器和输入数据滑动窗口组成。序列解析器在第 3 节已经详细介绍。在 3.1.2 小节中提到，序列解析器以多个字节组成的处理窗口作为输入，但解析器在每个周期解析的字节长度并不固定。因此，本文设计输入数据滑动窗口模块用于控制外部输入数据与序列解析器之间的数据流。该模块中设置了一个长度为 2*P 的输入数据缓冲区。缓冲区中靠前的 P 字节作为序列解析器的输入处理窗口。根据当前周期序列解析器的解析长度，该模块在下一周期将缓冲区中数据向前移动相应距离。每当缓冲区中的数据长度小于 P 字节时，该模块从外部输入中读取 P 字节补充到缓冲区。

解压引擎中包括两个信息传输通道，未压缩信息传输通道和匹配信息传输通道，都基于异步 FIFO 实现。未压缩信息传输通道用于传输序列解析器每周期输出的未压缩字符片段、未压缩字符片段长度、单个序列内未压缩字符总长度等未压缩信息。匹配信息传输通道用于传输序列解析器对每个序列解析得到的匹配字符长度和匹配位置偏移等匹配信息。

数据读写部分主要由字典、匹配信息转译、解压数据合并输出等模块组成。匹配信息转译模块根据匹配信息计算匹配数据在字典中的地址，通过向字典发送数据读取指令来控制匹配数据读取过程。解压数据合并输出模块接收来自传输通道的未压缩信息和来自字典的匹配数据。一方面，该模块负责将未匹配字符片段进行拼接合并；另一方面，该模块需要将未匹配字符与匹配数据拼接合并。此外由于合并后的数据长度并不固定，因此需要根据输出数据位宽进行整流输出。该模块在输出数据的同时将数据写入字典。

字典是数据读写部分的核心模块，其性能决定解压引擎的最大数据输出带宽。为实现高速的匹配数据还原，字典必须支持对任意位置的多字节串进行快速读取。然而，FPGA 中基于 block RAM 等片上存储单元实现的常规存储仅支持按相同数据粒度进行地址索引和数据读写，无法同时实现细粒度索引和高带宽读写。比如，对于一个存储粒度和读写位宽为 1 字节的常规存储，数据可按字节进行索引，但每周期只能读写 1 字节数据。针对此问题，本文设计实现了一个基于二次选择支持细粒度索引的高带宽字典。如图 10 所示，该字典以两个常规存储块对数据进行交替编址存储，通过一个数据选择逻辑从两存储块输出的粗粒度数据中选择更细粒度的数据输出。假设 8 字节字符串“abcdefgh”按 2 字节粒度存储，“ab”、“cd”、“ef”、“gh”分别位于图 10 中第 i、i+1、i+2、i+3 地址位置。若需要读取第 4、5 字节“de”，则字典先从分别位于两存储块的第 i+1、i+2 地址中读取出“cd”、“ef”，再通过数据选择逻辑从“cdef”中选择输出“de”。以上过程中，存储块的数据读取地址（rd_addr）和选择逻辑的选择信号根据字典读取地址生成。数据写入字典时，按照存储块的数据粒度交替写入两存储块。通过字典写入地址末位生成数据写入信号（wr_en）以选择目标存储块。在解压引擎实现中，字典存储块存储粒度设计与输出数据通路位宽相同。本文设计中存储块存储粒度为 32 字节，因此字典读写数据带宽可达 32 字节/周期。

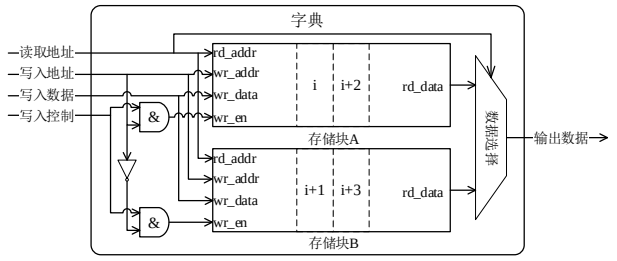


图 10 高带宽字典设计

4.2 多引擎解压加速器设计与原型实现

数据解压引擎内部数据处理并行度的提高可显著提高数据解压性能，但也使得设计逻辑复杂度快速增加。另一方面，解压引擎的实际吞吐率受限于输出带宽，因此难以通过不断增加解压引擎内部并行度提高解压性能。对此，本文提出多引擎扩展

解压加速器设计，在高性能解压引擎的基础上，实现多引擎对多解压任务的并行处理，进一步提升解压性能。此外，本文设计实现了一个基于CPU-FPGA架构的异构解压加速系统原型，研究加速器在应用场景中的部署实现。

4.2.1 可扩展多引擎解压加速器设计

本文设计的可扩展多引擎解压加速器架构如图9所示。加速器组成主要包括多个数据解压引擎、一个输入调度器、一个输出调度器。其中，数据解压引擎是解压系统的核心，其设计在4.1小节中已详细介绍。通过多引擎进行多任务并行处理可扩展提高数据解压吞吐率，解压引擎的数量可根据可用硬件资源和应用场景需要进行配置。输入调度器和输出调度器用于支持实现多引擎多任务并行解压。

输入调度器负责将输入任务流分配到多个解压引擎。调度器根据每个解压任务的数据长度将多个任务分离，采用基于轮询的调度策略对输入任务进行快速分配。针对每一个输入任务，调度器对解压引擎状态进行循环查询，将任务分配到轮询到的空闲引擎。输出调度器负责将各引擎输出的解压数据调度输出，同样采用轮询调度策略。此外，考虑到外部输入输出和解压引擎的数据流位宽和时钟频率可能不匹配，输入调度器和输出调度器中设计了基于异步FIFO缓存的数据流调控机制，用于调节转换数据流位宽和频率。

4.2.2 CPU-FPGA异构解压加速系统原型

本文提出的多引擎解压加速器可通过多种形式进行部署，如近存储计算（智能存储）、近网络计算（智能网卡）、PCIe加速卡等。可根据应用场景需求和硬件条件选择合适的部署形式。其中，PCIe加速卡可利用商用服务器进行增量部署，具备较好的兼容性。因此，本文选择结合CPU主机和PCIe加速卡的异构平台对多引擎解压加速器的部署应用进行原型验证测试，设计实现了一个基于CPU-FPGA架构的异构解压加速系统原型，其架构如图11所示。

系统原型由CPU和FPGA两部分组成，两者之间通过PCIe接口连接，采用DMA模式进行数据传输。FPGA部分主要包括多引擎解压加速器和数据I/O模块。数据I/O模块负责CPU和FPGA之间的数据传输控制，主要由PCIe传输接口逻辑和DMA引擎构成。CPU部分主要实现了一个LZ4数据解压加速调用接口。该接口支持多线程调用和异步任务处理模式，可实现硬件加速性能最大化。具

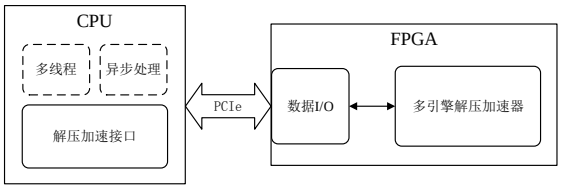


图11 CPU+FPGA异构数据解压加速系统架构

体来说，每个线程通过发送队列以异步处理模式向FPGA端下发解压任务，并通过队列状态判断加速器负载状态。多线程下发的任务以轮询方式进行处理，也可支持优先级处理等其他调度策略。

5 实验评估与分析

本节对本文提出的基于FPGA加速的高性能数据解压方法进行详细评估分析。首先，从多方面对解压加速引擎的性能进行评估，并与现有工作进行对比分析。之后，在系统原型实验基础上对多引擎加速系统的可扩展性和功效等方面进行评估分析。

5.1 实验配置和评估方法

本文从解压加速引擎和多引擎解压加速器两个层面对本文方法进行全面的评估分析。本文在解压加速引擎层面与已有方法进行了详细对比分析。为验证多引擎解压加速器的加速效果及可扩展性，本文通过实现的加速系统原型对加速器性能进行测试评估。实验评估基于Calgary数据集进行[19]。实验评估环境的详细参数配置如表2所示。

5.1.1 解压加速引擎评估方法

在解压加速引擎评估中，为更好地与现有最新工作进行对比，解压加速引擎评估的目标FPGA平台与现有方法[12]保持一致。解压加速引擎的字典带宽设置为32字节/周期，输出数据通路位宽为32字节。根据非对称输入输出通路设计，引擎中序列解析器的处理窗口大小（P）设置为16字节。

对解压加速引擎的评估包括解压性能和硬件设计两方面。解压性能评估的指标包括输出吞吐率和对应的每周期处理字节数，分别用于评估解压引擎的实际解压性能和处理并行度。在评估方法上，使用ModelSim对解压引擎的数据解压过程进行功能正确性验证，并记录解压过程所消耗的准确的时钟周期数。通过总时钟周期数、解压数据大小、设计频率来计算准确的解压吞吐率和每周期处理字节数。硬件设计评估主要针对资源使用效率，通过FPGA开发工具给出的资源使用报告得到引擎的资

表 2 实验评估环境参数配置

名称	规格参数
解压加速引擎评估	
FPGA	Xilinx Alveo U200
FPGA 开发工具	Xilinx Vivado 2019.1
硬件开发语言	Verilog
功能仿真工具	ModelSim PE Student Edition 10.4a
引擎处理窗口大小	16 字节
测试数据集	Calgary 压缩评估数据集
多引擎解压加速器评估	
异构原型主机	Intel Core i5-7500 CPU @ 3.4 GHz, CentOS Linux 7
异构原型 FPGA	Intel Arria 10 (10AX048H2F34E2SG)
FPGA 开发工具	Intel Quartus Prime Professional Edition 18.0
原型开发语言	Verilog (FPGA); C (software)
解压加速器设计频率	125 MHz
软件测试主机	Intel Core i7-6700HQ CPU @ 2.60GHz, Ubuntu 18.04 LTS
软件加速参考方案	Intel Integrated Performance Primitives 2020
测试数据集	Calgary 压缩评估数据集

源消耗数据。此外，为验证 3.2.2 小节提出的长度字段快速解析方法的效果，本文对使用该方法优化前后的设计分别进行评估分析。

5.1.2 多引擎解压加速器评估方法

在多引擎解压加速器评估中，主要通过系统原型的端到端数据解压测试和系统资源使用分析来对设计可扩展性进行评估验证，并对加速系统原型进行功耗效率分析。评估方法上，本文通过对比集成不同数量加速引擎的系统原型的端到端吞吐率、硬件资源使用、功耗效率来进行系统性能可扩展性分析。

在端到端吞吐率测试中，一定数据量的待解压任务持续从主机端向 FPGA 加速卡发送，加速卡处理完数据后返回至主机，以主机发送首个解压任务到收到最后一个已完成任务之间的时间作为总解压时间。通过总任务量和总解压时间计算端到端吞吐率。系统原型的硬件资源使用和功耗数据通过 FPGA 开发工具的报告得到。本文以吞吐率与功耗的比值作为系统的功耗效率的量化指标。本文对 LZ4 软件和基于 Intel Integrated Performance Primitives（IPP）软件加速库[20]的 LZ4 加速方法进行了测试，将其功耗效率作为参考对本设计的加速效果进行评估。

5.2 解压加速引擎评估分析

5.2.1 吞吐率性能分析

图 12 展示了本文解压加速引擎处理 Calgary 数据集中各类数据的每周期输出字节数。本文解压加速引擎实现了最大超过 9.7 字节的每周期吞吐量（geo）。从各类数据测试结果来看，解压吞吐率与压缩率存在一定相关性。数据压缩率越大时，数据解压过程需要还原的匹配字符越多，越能发挥字典的高带宽读取优势，则吞吐率越高。图 12 中的实验结果表明，对于大部分测试数据，其压缩率越大，则解压吞吐率越高。

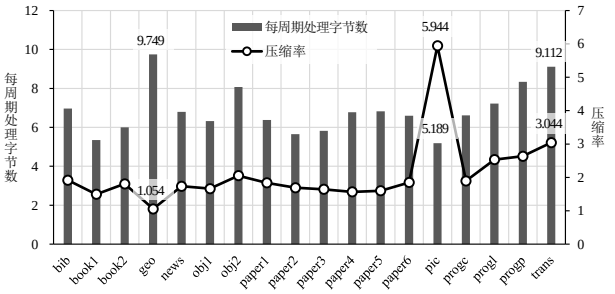


图 12 Calgary 数据集测试中解压加速引擎解压性能

但也存在其他因素导致吞吐率与压缩率呈负相关的情况，大致可分为两种情况。一种情况是压缩率很低的时候吞吐率性能反而较高，对应图 12 中 geo 数据的测试结果。该情况是由于低压缩率数据中存在大量连续的未压缩字符字段，可被序列解析器以高并行度进行解析，因此吞吐率性能反而较高。

另一种情况是压缩率很高的时候吞吐率反而较低，对应测试中 pic 数据的解压性能。产生此现象的主要原因是 pic 数据中存在大量连续重复的字符串。此类数据压缩后，序列的匹配偏移一般较小。在数据解压过程中，当匹配偏移小于输出数据通路位宽时，每周期最大只能还原输出与匹配偏移大小相同长度的数据，导致该情况下输出吞吐率较低。

5.2.2 解压引擎设计对比分析

本文将提出的解压加速引擎与首个 LZ4 解压加速方法（MLZ4D）[16]、已有最新的 LZ4 解压加速方法（APNet-22）[11]、当前业界主流方法（Xilinx 和 Xilinx-Streaming）[12]进行对比分析，各解压加速引擎方案的详细硬件资源使用及解压性能数据如表 3 所示。其中，本设计 1 为未采用长度字段快速解析方法优化的设计，本设计 2 为优化后的设计。与现有 LZ4 解压引擎相比，本文设计在解压性能和存储资源效率上均有有显著提升。

在频率性能方面，由于并行化序列解析器的多字段并行解析设计增加了部分处理时延开销，导致本设计 1 的设计频率（217 MHz）与频率最高的 Xilinx-Streaming 方法（300 MHz）相差约 27%。本

表 3 各解压引擎方案的资源使用及解压性能对比

方案	逻辑资源		存储资源	频率	吞吐率	处理并行度	逻辑资源效率	存储资源效率
	LUT	FF	BRAM	（MHz）	（MB/s）	（字节/周期）	（吞吐率/LUT）	（吞吐率/BRAM）
MLZ4D	342	377	20	260	260	1	0.76	13
APNet-22	-	-	14.44	-	660.76	-	-	45.8
Xilinx	7300	7000	34	262	443	1.69	0.06	13
Xilinx- Streaming	6000	5000	32	300	368	1.23	0.06	11.5
本设计 1	18447	2136	15	217	1491	6.87	0.08	99.4
本设计 2	15266	2191	15	263	**1718**	**6.87**	0.11	**114.5**

注：部分方案（APNet-22、Xilinx、Xilinx-Streaming）的使用了 BRAM 以外的存储资源，本文按照单个 BRAM 的容量（36 Kb）将对应方案的存储资源使用换算为消耗的 BRAM 数量。

文提出的长度字段快速解析方法有效减少了的关键路径时延。因此，本设计 2 相比本设计 1 的设计频率从 217 MHz 提高到 263 MHz，提升了约 21%。

在解压性能方面，并行序列解析方法的高处理并行度显著提升了解压引擎的吞吐率性能。本设计的处理并行度达到了平均 6.87 字节/周期，是现有最好方法（Xilinx）的 4.1 倍。本设计 2 可实现超过 1.7 GB/s 的吞吐率，达到了现有最好方法（APNet-22）的 2.6 倍。在时延敏感场景中，解压引擎吞吐率的提升可降低单个任务的处理时间，加快上层应用的处理响应。

在资源使用方面，由于 MLZ4D 方法采用较为简单的单字节处理设计，因此其逻辑资源消耗相比其他方案而言相对较少。除 MLZ4D 之外，其他方法都对解压引擎进行了一定并行化设计，因此逻辑资源使用量有所增加。本设计 2 实现了并行化方案中最高的逻辑资源效率。由于存储资源在多引擎加速框架的输入输出调度器以及 FPGA 数据 I/O 模块中需要大量使用，因此解压加速引擎较少的存储资源消耗有助于平衡整体设计的资源使用。结合资源使用和吞吐率性能来看，本文设计实现了最高存储资源使用效率，每单位 BRAM 可实现 114.5 MB/s 的吞吐率，是现有最好方法（APNet-22）的 2.5 倍。

总的来说，本设计的解压引擎实现了当前最高的吞吐率、处理并行度、存储资源效率。解压引擎吞吐率的提高可加快单个任务处理效率，存储资源效率的提高有利于多引擎加速器的性能扩展。

5.3 多引擎解压加速器评估分析

5.3.1 系统性能可扩展性分析

在解压吞吐率方面，集成不同数量加速引擎的系统端到端吞吐率性能如图 13 所示。集成 4 加速引擎的系统原型的端到端解压吞吐率最大可达 4 GB/s 以上。对于大部分测试数据，解压加速系统的端到端吞吐率随着加速引擎数量而线性递增，该结果表明解压加速系统具备良好的吞吐率可扩展性。

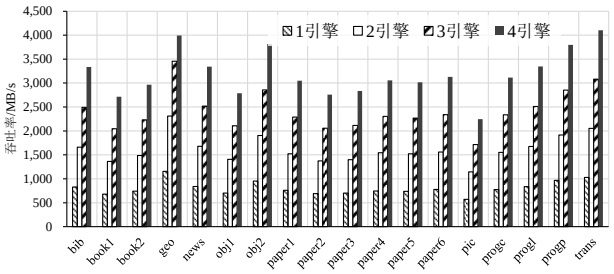


图 13 多引擎解压加速系统原型的端到端解压吞吐率

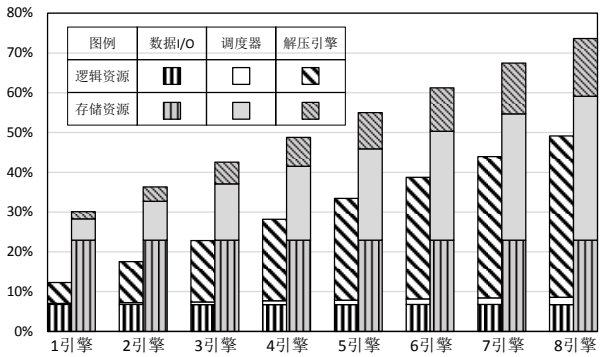


图 14 系统原型各部分消耗资源占 FPGA 总资源的比例

良好的资源使用效率是保证系统性能可扩展性的基础。图 14 展示了集成不同数量解压引擎的系统原型的各部分使用的逻辑资源和存储资源占总可用资源量的比例。其中，数据 I/O 模块用于实现 CPU 和 FPGA 之间的数据传输，其实现与加速器集成的引擎数量没有直接联系，因此该模块的逻辑资源和存储资源使用量基本不变。多引擎解压加速器（输入输出调度器和解压引擎）所使用的资源量随着引擎数量增长，大致呈线性增长趋势。总的来看，系统总资源使用量不会随着集成加速引擎数量增大而快速膨胀。

结合吞吐率性能可扩展性分析可知，随着加速引擎的增加，系统原型的“资源-性能”费效比在逐渐提升，系统整体资源性能效率更高。对于各类资源的使用，存储资源消耗要多于逻辑资源。此外，各模块消耗各类资源的情况有明显差别，数据 I/O 模块和调度器消耗的存储资源比例显著大于逻辑

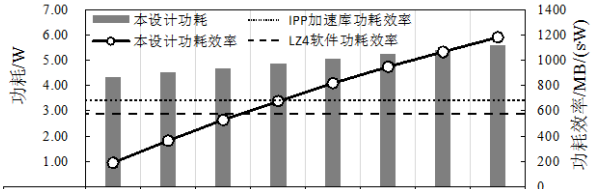
资源比例。在 8 引擎的系统原型中，解压引擎消耗了 40.5%的逻辑资源及 14.5%的存储资源，其他模块消耗了 8.6%的逻辑资源及 59.1%的存储资源。因此，本文解压加速引擎设计的较低存储资源消耗可以平衡整体资源使用，提高整体资源利用效率。

以上的系统原型测试分析表明多引擎解压加速器具备良好的性能可扩展性和资源性能效率。

5.3.2 功耗效率分析

功耗效率是数据中心、移动计算等功耗敏感场景中的重要应用性能指标，也是衡量硬件加速设计的关键因素。本文对集成不同数量解压引擎的加速系统功耗进行评估分析，并结合吞吐率对功耗效率进行评估。

加速系统的功耗性能评估结果如图 15 所示。由于引擎数量的增多带来硬件资源使用的增多，系统的功耗也会随之增大，但增长幅度并不大。功耗增长趋势较慢的原因有两方面。一方面，数据 I/O 模块的资源使用量不会随引擎数量变化，带来的功耗量基本不变。另一方面，FPGA 芯片中的未使用资源也会产生一定量的功耗。因此，总功耗的增加主要是由增加引擎的资源消耗所致。随着集成引擎数量的增大，FPGA 整体资源利用率得到提升，系统需要额外增加的功耗并不多。由于吞吐率增速要显著大于功耗增速，因此系统的功耗性能效率随着引擎数量在不断增大。集成 8 引擎的系统功耗效率可达 1180 MB/(s·W)，是 LZ4 软件和 IPP 加速库的 1.8 倍和 1.6 倍以上。



引擎数量	1	2	3	4	5	6	7	8
功耗	4.33	4.52	4.69	4.88	5.04	5.23	5.41	5.59
功耗效率	190.5	365.0	527.7	676.2	818.5	946.5	1067.5	1180.7

图 15 多引擎解压加速系统原型的功耗及功耗效率

6 相关研究

6.1 LZ4算法加速研究

自 Yann Collet 在 2011 年提出 LZ4 算法后，若干针对 LZ4 算法进行加速的研究被相继提出。大多数研究主要关注对 LZ4 压缩过程进行加速设计。然而，在数据库等读多写少的应用场景中，数据解压性能更为重要，解压加速设计相比压缩加速设计也更具备挑战性[4]。

2015 年，Bartik M 等人首次提出了 LZ4 算法加速设计，主要研究基于 FPGA 对 4K 视频流进行 LZ4 无损压缩[15]。该设计按照与 LZ4 软件类似的模式进行逐字节压缩处理，没有对压缩过程进行并行化设计。由于视频数据包体积小于标准字典大小，该设计减小了字典大小和哈希表大小，降低了存储资源使用，但也使得压缩率性能损失较大。该团队在后续对其设计进行了并行化改进，研究提出了支持并行处理的重复匹配查找单元[21]，并以此为基础设计了更高性能的 LZ4 压缩加速器[22]。但该研究中采用的多端口读写字典方案需要消耗大量的存储资源，其存储资源效率较低。

为提高压缩率性能，顾巍等人将 LZ4 算法与哈夫曼编码结合，实现了一种基于 FPGA 的压缩加速设计[23]。该研究采用哈夫曼编码对 LZ4 压缩后的数据进行二次压缩，进一步提高了压缩率。该研究对压缩设计进行了部分并行化改进，可支持最大 4 字节每周期的处理速度。

针对 LZ4 压缩过程数据输出时延不稳定的问题，Liu W 等人研究提出了基于 LZ4 算法改进的数据压缩 FPGA 加速设计，并同时提出了解压加速设计[16]。该研究通过修改 LZ4 序列格式来减少压缩数据输出的时延波动。然而，该设计的数据格式无法与 LZ4 软件兼容，因此不具备普适性。该研究的压缩和解压设计都采用逐字节处理模式，因此其吞吐率性能也相对较低。

针对压缩加速设计中普遍存在的存储资源消耗较大、FPGA 资源使用不均衡不充分的问题，Liu P 等人研究提出了一个基于 CPU-FPGA 异构加速的高性能无损压缩系统设计及并行化 LZ4 压缩加速核设计[24]。该研究围绕提高资源使用效率对压缩加速核进行了并行化设计，实现了较高的存储性能效率。

在业界，Xilinx 对压缩算法加速进行了较多研究，实现了包括 LZ4 在内的一系列压缩算法加速库[12]。Xilinx 采用高层次综合技术对硬件设计进行实现[25]，该方法具备可敏捷开发应用的特性。其 LZ4 解压设计对未压缩字符字段解析和匹配字符还原过程进行了优化改进，可支持部分并行处理。但该设计仍然局限于逐字段处理，其并行化程度有限。

针对 LZ4 解压的并行化难题，Mahony A 等人提出了一种减少数据依赖的 LZ4 解压硬件设计优化改进方法[26]。该研究面向分布式存储应用场景，

主要提出在序列解析与匹配数据还原之间设立缓存以提高处理效率，并对匹配数据还原过程进行了并行化改进。但该研究没有关注对序列解析的并行化设计。Liu P 等人提出了结合 LZ4 压缩和解压加速的流量压缩系统设计，建立了资源性能模型对压缩解压性能进行优化配置[11]。但该工作没有给出具体的解压加速引擎设计及实验数据。

上述工作对 LZ4 压缩和解压加速进行了广泛探索，但在并行化设计方面缺乏深入研究。大多数解压加速方法没有进行并行化设计，导致其吞吐率较低。一些方法对 LZ4 解压的部分过程进行了并行化改进，但仍然没有充分发挥硬件并行处理优势。此外，现有方法中硬件设计各部分数据通路都按相同带宽进行设计，没有考虑实际数据解压过程中输入输出吞吐率不一致的情况，因此普遍存在输入带宽利用不充分以及输出带宽不足的问题。本文针对以上问题进行了深入研究，从多层次对 LZ4 解压加速设计进行并行优化改进，提出了一种新的基于 FPGA 加速的高性能数据解压方法。

6.2 其他解压加速研究

大量工作对 LZ4 以外的其他经典无损数据解压算法加速开展了广泛研究，主要围绕 Deflate、Snappy、LZW 等常见算法进行加速设计。

Deflate 算法将 LZ77 压缩和哈夫曼编码结合，可实现较高的压缩率，是 Zlib、Gzip 等压缩软件的基础[6]。Lazaro J 等人在 2007 年提出了首个基于 FPGA 的 Deflate 解压加速设计，可实现在 212 MHz 频率下每周期 1 字节的吞吐率[27]。Zaretsky D 等人提出了应用在网络流量压缩解压处理场景中的 Deflate 解压加速方法[28]。该方法通过 8 引擎加速在 120 MHz 频率下可实现 1 GB/s 解压吞吐率，但其解压引擎没有进行并行化设计。Ouyang J 等人针对数据中心存储场景提出了基于 FPGA 的 Gzip 压缩解压加速设计，有效提高了存储利用率并降低了 CPU 开销[29]。该方法对哈夫曼解码过程进行了并行化改进，其 4 解压引擎系统在 132 MHz 频率下实现了最大约 300 MB/s 的解压吞吐率。Ledwon M 等人提出了基于高层次综合技术的 Deflate 压缩解压加速设计，其解压引擎实现了大约 550 MB/s 的平均输出吞吐率[13,30]。

Snappy 算法是 Google 提出的基于 LZ77 算法的一种新型快速无损压缩算法，被广泛应用在大数据处理场景中[7]。2018 年，Qiao Y 等人研究并提出了首个面向 Snappy 解压的 FPGA 加速设计，可在常见数据库测试集中实现最大 15 字节每周期的输出吞吐率[31]。针对 Qiao Y 等人研究中数据还原过程存在访存冲突的问题，Fang J 等人对该研究进行了进一步优化改进，设计实现了一个高性能 Snappy 解压加速器[14,32]。该设计支持对多个 Snappy 基本压缩单元的并行处理以及高效的字典读写，可实现最大 31 字节每周期的输出吞吐率。

LZW 算法是一种基于 LZ78 算法的经典无损压缩算法，是 GIF、TIFF 等图像编码格式的基础。与 LZ77 系列算法不同，其字典并不直接采用历史数据，需要实时计算构造，因此其加速设计难度较大。Lin M 等人较早提出了并行化的 LZW 算法硬件设计[33]。该研究对 LZW 软件采用的递归查询字典进行了改进，提出了一个可并行访问的字典集设计，提高了压缩和解压吞吐率。针对 LZW 解压字典并行化构造难题，Kagawa H 等人对 LZW 解压过程进行了分解，提出了一种吞吐率最优的 LZW 加速设计[34]。该设计可实现约 295 MB/s 的解压吞吐率。

7 总结

高性能无损压缩数据解压在解压时间敏感的应用场景中至关重要，但同时需要大量计算开销。由于并行化程度较低，现有基于 FPGA 的数据解压加速研究难以实现高性能 LZ4 数据解压。本文研究从多层次对 LZ4 解压进行并行化设计，提出了以并行化序列解析器为核心的高性能数据解压引擎以及可扩展多引擎解压加速器。本文解压引擎在吞吐率、并行度、存储资源效率等方面均优于现有方法。系统原型测试表明本文提出的多引擎解压加速器具备良好的性能可扩展性及功耗效率。

展望未来，本文研究可在多领域进行扩展应用。本文设计提出的多字段并行解析、长度字段快速解析、多引擎并行化扩展等方法并不限于 LZ4 解压加速，也可应用于其他解压算法加速设计。考虑到并行解析字段组合对序列解析器的处理并行度和逻辑资源消耗有着重要影响，后续研究将探索对并行解析字段组合进行优化改进。

致 谢 本课题得到国家自然科学基金(No. 62202486, 61972412, U22B2005, 12102468)、国防科技大学校科研项目(No. ZK21-02)等提供相关支持。感谢所有评审人员的建议和帮助!

参 考 文 献

[1] Hu X, Wang F, Li W, et al. QZFS: QAT accelerated compression in file system for application agnostic and cost efficient data storage//Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference. Renton, USA, 2019: 163-176

[2] Jia Y, Shao Z, Chen F. SlimCache: an efficient data compression scheme for flash-based key-value caching. ACM Transactions on Storage, 2020, 16(2): 14

[3] Tu Y, Chen H, Wang H, et al. Research on heterogeneous accelerated columnar storage engine based on co-optimization of software and hardware for GoldenX. Chinese Journal of Computers, 2022, 45(1): 207-223 (in Chinese)
(屠要峰, 陈河堆, 王涵毅等. 面向GoldenX软硬协同优化的异构加速列式存储引擎研究. 计算机学报, 2022, 45(1): 207-223)

[4] Fang J, Mulder Y T B, Hidders J, et al. In-memory database acceleration on FPGAs: a survey. The VLDB Journal, 2020, 29(1): 33-59

[5] Collet Yann, et al. LZ4: extremely fast compression algorithm, https://github.com/lz4/lz4 2023,3,13

[6] Deutsch P, RFC1951 DEFLATE compressed data format specification version 1.3, https://tools.ietf.org/html/rfc1951 2023,3,13

[7] Google, Snappy: a fast compressor/decompressor, https://github.com/google/snappy 2023,3,13

[8] Agostini M, O'Brien F, Abdelrahman T. Balancing graph processing workloads using work stealing on heterogeneous CPU-FPGA systems//49th International Conference on Parallel Processing (ICPP). Edmonton, Canada, 2020: 50

[9] Samardzic N, Qiao W, Aggarwal V, et al. Bonsai: high-performance adaptive merge tree sorting//Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). 2020: 282-294

[10] Abali B, Blaner B, Reilly J, et al. Data compression accelerator on IBM POWER9 and z15 processors//Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). 2020: 1-14

[11] Liu P, Chen S. Reducing Network Traffic Storage Overhead: A Hardware-Accelerated Lossless Data Compression System, https://conferences.sigcomm.org/events/apnet2022/program.html 2022,7,1

[12] Xilinx. Xilinx LZ4 compression and decompression, https://xilinx.github.io/Vitis_Libraries/data_compression/2022.1/source/L2/lz4.html 2023,3,13

[13] Ledwon M, Cockburn B F, Han J. High-throughput FPGA-based hardware accelerators for Deflate compression and decompression using high-level synthesis. IEEE Access, 2020, 8: 62207-62217

[14] Fang J, Chen J, Lee J, et al. An efficient high-throughput LZ77-based decompressor in reconfigurable logic. Journal of Signal Processing Systems. 2020, 92(9): 931-947

[15] Bartik M, Ubik S, Kubalik P. LZ4 compression algorithm on FPGA//2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS). Cairo, Egypt, 2015: 179-182

[16] Liu W, Mei F, Wang C, et al. Data compression device based on modified LZ4 algorithm. IEEE Transactions on Consumer Electronics, 2018, 64(1): 110-117

[17] Sayood K. Introduction to Data Compression. Fifth edition. USA: Elsevier, 2018

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

[19] Bell T, Witten I, Cleary J. Modeling for text compression. ACM Computing Surveys, 1989, 21(4): 557-591

[20] Intel. Accelerating LZ4 with Intel Integrated Performance Primitives, https://www.intel.com/content/www/us/en/developer/articles/technical/accelerating-lz4-with-integrated-performance-primitives.html 2023,3,13

[21] Bartik M, Benes T, Kubalik P. Design of a high-throughput match search unit for lossless compression algorithms//2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC). Las Vegas, USA, 2019: 732-738

[22] Benes T, Bartik M, Kubalik P. High throughput and low latency LZ4 compressor on FPGA//2019 International Conference on ReConFigurable Computing and FPGAs. Cancun, Mexico, 2019: 1-5

[23] Gu W. The optimization of LZ4 lossless compression algorithm based on FPGA [master's thesis]. Southeast University, Nanjing, China, 2017 (in Chinese)
(顾巍. 基于FPGA的LZ4 无损压缩算法优化设计[硕士学位论文]. 东南大学, 南京, 2017)

[24] Liu P, Wei Z, Yu C, Chen S. HybriDC: a resource-efficient CPU-FPGA heterogeneous acceleration system for lossless data compression. Micromachines, 2022, 13(11): 2029

[25] Xilinx. Vitis high-level synthesis user guide, https://docs.xilinx.com/r/en-US/ug1399-vitis-hls 2023,3,13

[26] Mahony A, Tringale A, Duquette J, et al. Reduction of execution stalls of LZ4 decompression via parallelization, USA, 2018,5,15

[27] Lazaro J, Arias J, Astarloa A, et al. Decompression dual core for SoPC applications in high speed FPGA//The 33rd Annual Conference of the IEEE Industrial Electronics Society (IECON). Taipei, China, 2007: 738-743

[28] Zaretsky D, Mittal G, Banerjee P. Streaming implementation of the ZLIB decoder algorithm on an FPGA//2009 IEEE International Symposium on Circuits and Systems (ISCAS). Taipei, China, 2009: 2329-2332

[29] Ouyang J, Luo H, Wang Z, Tian J, Liu C, Sheng K. FPGA implementation of GZIP compression and decompression for IDC services//2010 International Conference on Field-Programmable Technology (FPT). Beijing, China, 2010: 265-268

[30] Ledwon M, Cockburn B F, Han J. Design and evaluation of an FPGA-based hardware accelerator for Deflate data decompression// 2019 IEEE Canadian Conference of Electrical and Computer

Engineering (CCECE). Edmonton, Canada, 2019: 1-6

[31] Qiao, Y. An FPGA-based Snappy decompressor-filter [master's thesis]. Delft University of Technology, Delft, The Netherlands, 2018

[32] Fang J, Chen J, Lee J, et al. Refine and recycle: a method to increase decompression parallelism//2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP). New York, USA. 2019: 272-280

[33] Lin, M. A hardware architecture for the LZW compression and decompression algorithms based on parallel dictionaries. The Journal of VLSI Signal Processing-Systems for Signal, Image, and Video Technology. 2000, 26: 369–381

[34] Kagawa H, Ito Y, Nakano K. Throughput-optimal hardware implementation of LZW decompression on the FPGA//2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW). Nagasaki, Japan, 2019: 78-83

LIU Pu-Guang, Ph.D. candidate. His research interests include big data processing acceleration and cyberspace security.

WEI Zi-Ling, Ph.D., associate professor. His main research interests are network forensics and network optimization.

HUANG Cheng-Long, Ph.D., research associate. His main research interest is in-memory computation.

CHEN Shu-Hui, Ph.D. supervisor, professor. His research interests focus on network forensics and high-performance network traffic processing.

Background

In the current era of big data, the large data scale has brought huge challenges to data processing. Lossless data compression can improve the efficiency of storage and processing. In read-first applications such as databases, data decompression performance will significantly affect the quality of services. The LZ4 lossless compression algorithm with high decompression throughput is widely used in these application scenarios. However, it inevitably incurs significant CPU overhead. To reduce CPU costs, many academic and industrial studies have focused on accelerating lossless data decompression by field-programmable gate arrays (FPGAs).

The data dependency of LZ4 decompression severely limits the parallelizability of LZ4 decompression, which make the acceleration of LZ4 decompression a huge challenge. Most existing studies are deficient in parallelization and cannot take advantage of FPGAs. To address the challenge of high-performance LZ4 decompression acceleration, this paper designs the parallel acceleration of LZ4 decompression from multiple levels and proposes an FPGA-accelerated high-performance LZ4 data decompression method. This method improves the parallelization of the LZ4 sequence parsing process and designs a parallel sequence parser based on multi-field parallel parsing. The parser extends the throughput from one byte per cycle to multiple bytes per cycle. In addition, .is optimized and a dichotomous-based fast maximum match length parsing method is proposed. It significantly reduces the critical path delay of the sequence parser and improves the design clock frequency by about 21%. Moreover, a high-performance data decompression engine is designed based on the parallel sequence parser. The engine decouples the sequence parsing and data recovery processes and extends the decompression output data path to solve the input-output throughput mismatch. To further improve the throughput performance, this method proposes a scalable multiple-engine acceleration system framework for high-performance data decompression and implements a heterogeneous end-to-end data decompression acceleration system prototype on the CPU-FPGA architecture. Experimental analysis shows that the byte-per-cycle throughput of the decompression engine is 4.1–6.8 times higher than that of existing studies. The engine achieves a decompression throughput of about 1.7 GB/s, which is a 2.6–6.6 times improvement compared to existing studies. The end-to-end throughput experiment and resource usage evaluation results of the system prototype show that the proposed data decompression acceleration system has good scalability in terms of throughput and resource usage. In addition, the power efficiency of the decompression acceleration system with 8 engines is more than 1.6 times that of the software-based acceleration method.

This work has been supported in part by the National Natural Science Foundation of China (No. 62202486, 61972412, U22B2005, and 12102468), and the National University of Defense Technology Foundation (No. ZK21-02).