LITERATURE REVIEW

In 2020, Manak Pandey conducted a study. The paper proposes an enhanced data compression technique that is lossless and format conservative. It is built using existing compression algorithms either directly or through modifications. The proposed technique is more efficient than pre-existing algorithms for both dictionary-based and non-dictionary-based databases. The working is that the words are given indexes. Then words-indexes are sorted lexicographically. In other words, first modified RLE is applied then incremental prefix encoding is applied. For the indexes, concatenate the consecutive indexes with the same number of digits and store them in a tuple.

The researchers experimented with combinations of various lossless compression algorithms and found that while many combinations produced good compression efficiency, they had a high risk of failure for certain data types. The proposed algorithm emerged as the most efficient one, especially for big data with high logical redundancy. The effectiveness of the proposed technique is greater for large datasets that contain redundant data. Specifically, for datasets containing 100,000 words or more, the technique achieves a compression rate of over 35%. Overall, the paper highlights the potential of seeking better and more efficient compression algorithms from already existing ones.

In the year 2020, Md. Atiqur Rahman introduced a novel lossless text compression algorithm that integrates several established techniques to achieve efficient compression. It is a mix of Burrow Wheeler Transformation (BWT), modified Run-Length Encoding (RLE) and Huffman Encoding. The proposed method splits larger files into smaller ones with each file having the same number of characters. Then apply BWT. After applying BWT, only replace characters which appear more than 3 times in a row. The character is replaced by a formula (counts+96) Then apply RLE on it. This will help for effective compression. For the character length 3, characters are replaced by key by formula character and its count. For the character length 2, no compression code needed. After all these 2-key codes we apply Huffman encoding.

The results of this study indicate that the proposed compression technique outperforms various well-known compression algorithms. On average, it demonstrates a compression ratio that is 12.79% higher than PAQ8n, 21.13% higher than Deflate, 24.73% higher than Bzip2, 20.17% higher than Gzip, 31.63% higher than LZMA, 31.7% higher than LZW, and 11.52% higher than Brotli. However, it is important to note that in terms of both encoding and decoding time, this proposed method is faster than PAQ8n and LZMA but relatively slower compared to the other compression techniques mentioned in the article.

In 2018, Farooq Sunar Mahammad performed a study that focuses on a dual-core architecture and leveraging the OpenMP tool. The investigation revolves around evaluating the effectiveness of this parallel approach in terms of execution time and overall performance across various data compression algorithms, including arithmetic, LZ77, and K-RLE.

The study's findings reveal compelling outcomes, with notable enhancements in algorithm performance achieved through parallelization. Among the compression algorithms scrutinized, arithmetic compression emerges as the frontrunner, showcasing a substantial 46% improvement in data compression when compared to its counterparts, LZ77 and K-RLE, which exhibit commendable enhancements of 44% and 37%, respectively. These outcomes underscore the potency of parallel processing in optimizing data compression algorithms. This study provides valuable insights that pave the way for future advancements in the realm of data compression algorithms, with a focus on parallelization as a catalyst for improvement.

In 2017, Adam Gleave conducted a study that explored the use of the Unicode encoding scheme UTF-8 on the internet, which maps non-ASCII characters to sequences of two to four bytes. As websites have shifted to include more languages other than English, text compression methods that operate on individual bytes have not adapted. The Unicode Consortium has defined the Standard Compression Scheme for Unicode (SCSU) and Binary Ordered Compression for Unicode (BOCU-1) to address this issue. However, these compressors are not as effective on longer texts as gzip and bzip2. The paper investigates an alternative approach that modifies existing compression algorithms such as PPM and LZW to operate directly over Unicode characters rather than bytes, using an adaptive character model based on Polya trees. The approach gives an average improvement in compression effectiveness for LZW of 12.2% and for PPM of 6.1% on a corpus of UTF-8 files. The paper also discusses a token representation for UTF-8, which involves mapping between sequences of bytes and sequences of tokens, including UnicodeChar (c) representing a Unicode character with code point c, ErrorByte (b) representing a byte b in an ill-formed sequence, and EOF as an end-of-file marker. The paper introduces three contextless models over tokens, two of which are non-adaptive and one that is adaptive, and are used as components in the Unicode variants of LZW and PPM.

In a 2017 study by C. Oswald, a novel text compression algorithm was introduced. This algorithm, referred to as the Frequent Pattern-based Huffman (FPH) algorithm, combines frequent pattern mining with conventional Huffman encoding to efficiently compress text data. This article presents a new and efficient text compression algorithm that uses frequent pattern mining in combination with conventional Huffman encoding. The proposed algorithm is designed to address the problem of mining a large and optimal set of patterns by using an efficient frequent pattern mining technique and a hash structure. The proposed algorithm reduces the time required to compress text and improves the compression ratio. Apriori algorithm is used for frequent data mining. Encoding of patterns are derive from Huffman. These codes are stored in a tree. Lengthier patterns are given preference in the tree. It is because they accommodate many characters in a code. The results of the proposed FPH algorithm show that it has a high compression ratio and encoded size, with a compression efficiency ranging from 18-751% on both sparse and dense benchmark datasets. The optimizations in the algorithm have been proven to effectively obtain high compression ratios for dense sequence corpora, overcoming the limitations of text compression algorithms in terms of storage space and time overhead.

In 2017, Robbi Rahim conducted a study that explored the synergy of two key algorithms: the Blowfish algorithm for data security and the Lempel-Ziv-Welch (LZW) algorithm for data compression. The Blowfish algorithm is used to encrypt the data, which produces ciphertext that is then compressed using the LZW algorithm. This approach leads to faster data transmission as the compressed data requires less space and resources. Blowfish encodes the data thus, the achieved compression ratio of 12.5% may suggest that the compression ratios are relatively modest in size. An identified drawback of the LZW algorithm pertains to its ability to handle shorter strings that are present in the dictionary, leading to less efficient compression in such cases.

In 2016, Aníbal Guerra conducted the study that evaluates compression tools for DNA raw data in FASTQ files. General-purpose tools are user-friendly but less specialized, suitable when compression ratios, memory, and multi-threading matter less. Domain-specific tools offer better compression ratios, especially in single-threaded scenarios, with DSRC-fast performing exceptionally well. Multi-threading boosts performance but increases memory usage. Results showed that domain-specific tools increased the compression ratios up to 70%, while reducing the runtime of general-purpose tools up to 7× during compression and up to 3× during decompression. Parallelism scaled performance up to 13× when using 20 threads.

Tool choice depends on specific needs. For max compression, use SCALCE or FASTQZ; QUIP balances speed and effectiveness. PIGZ and random-access tools suit quick access, while PIGZ and PBZIP2 are valuable for common formats and high-performance computing. P7ZIP and SCALCE are ideal with ample memory and non-critical runtime.

The paper highlights installation challenges for domain-specific tools but notes their user-friendliness. Successful tools employ hybrid approaches, combining traditional and modern techniques for better compression and speed.

In this article by Dr. S. Pannirselvam, four different types of lossless compression techniques were explored and their performance on text data compression and decompression was evaluated. The algorithms were comprehensively described, including their working mechanisms and limitations. The assessment considered key metrics such as compression ratio, compression time, and compression speed across various input sizes.

Among the algorithms assessed, the Huffman technique emerged as particularly noteworthy for its superior compression ratio in comparison to the other techniques. Furthermore, it exhibited commendable performance in terms of compression time and speed. This observation underscores the efficacy of Huffman compression for text data.

However, it is worth noting that the pursuit of future research endeavors remains essential. There is a continued need for the development of a highly efficient lossless text compression algorithm that can be effectively applied in real-time applications where text data compression is a vital requirement.

Zhongya Wang conducts a study that addresses the scalability challenges faced by collaborative filtering (CF) algorithms in large-scale real-world applications. To tackle this issue, the authors propose a CUDA-based CF algorithm designed for multi-GPU platforms. Through complexity analysis, they identify the similarity matrix calculation as the computational core of the algorithm.

To optimize performance, the research leverages CUDA kernel functions on GPUs, implementing a tile scheme and various optimizations. Additionally, a workload partitioning scheme is introduced to balance the computational load across different GPUs.

The study conducts experiments using real-world datasets, demonstrating the remarkable efficiency of the proposed CUDA-based CF algorithm on multi-GPU platforms, achieving an impressive speedup of up to x3,691. The algorithm proves to be scalable when varying the number of users, items, and GPUs. The results also reveal that higher throughput on GPUs can be attained by allocating more computation