Journal of the Faculty of Engineering and Architecture of Gazi University 39:3 (2024) 1933-1943

Mühendislik Mimarlık Fakültesi Dergisi
Journal of The Faculty of Engineering and Architecture of Gazi University
Elektronik / Online ISSN : 1304 - 4915
Basılı / Printed ISSN : 1300 - 1884

CComp: A parallel compression algorithm for compressed word search

Emir Öztürk*, Altan Mesut
Department of Computer Engineering, Faculty of Engineering, Trakya University, 22030, Edirne, Türkiye

Highlights:

- Parallel compression algorithm for compressed search
- Faster decompression than most algorithms
- 6 times faster compressed search than the best algorithm which is Zstd

Keywords:

- Compressed Search
- Data Compression
- Parallel Programming
- Text Compression

Article Info:

Research Article
Received: 05.11.2022
Accepted: 27.11.2023

DOI:

10.17341/gazimmfd.1199811

Acknowledgement:

Correspondence:

Author: Emir Öztürk
e-mail: emirozturk@trakya.edu.tr
phone: +90 535 237 1982

Graphical/Tabular Abstract

In this study, a novel text compression algorithm named CComp is presented. CComp mainly aims to reduce compressed search times on compressed streams. To achieve this goal, CComp uses parallel encoding and removes some reducing steps of map-reduce operations in compression stage. Compressed search results of different algorithms and comparison of these results with CComp are given in Table A.

Table A. Compressed search times of different algorithms in seconds

	Pigz Gzip	Pigz Zlib	Pbzip2	Zstd	Xz64	CComp 4 thrd	CComp 8 thrd	CComp 16 thrd	CComp 32 thrd	CComp 64 thrd	Grep
english50mb	0,293	0,248	1,652	0,091	0,536	0,035	0,026	0,019	0,016	0,015	0,115
enwik8 (95 Mb)	0,559	0,450	3,114	0,172	1,37	0,063	0,045	0,036	0,028	0,025	0,190
english100mb	0,580	0,495	3,277	0,173	1,162	0,064	0,045	0,032	0,025	0,022	0,229
english200mb	1,147	0,988	6,559	0,340	2,494	0,120	0,089	0,065	0,047	0,043	0,458
enwik9 (953 Mb)	5,178	5,069	29,196	1,517	12,089	0,566	0,475	0,327	0,311	0,276	1,806
english1024mb	5,837	5,987	33,555	1,801	13,480	0,554	0,451	0,320	0,281	0,253	1,712
turkish50mb	0,285	0,285	1,480	0,109	0,625	0,059	0,042	0,029	0,023	0,023	0,059
turkish100mb	0,555	0,555	2,928	0,196	1,219	0,102	0,081	0,055	0,043	0,041	0,102
turkish200mb	1,092	1,092	5,843	0,369	2,417	0,175	0,142	0,111	0,078	0,071	0,175
turkish1024mb	5,500	5,500	29,845	1,761	12,285	0,815	0,652	0,483	0,418	0,382	0,815
TOPLAM	21,026	20,669	117,449	6,529	47,677	2,553	2,048	1,477	1,270	1,151	5,661

Purpose:
The aim of this study is to offer a much faster compressed search process than previously developed algorithms while providing the compression time and ratio balance as these algorithms.

Theory and Methods:
To obtain the time results on the selected datasets, each method was run 10 times for each file and the time values were averaged. CComp is run with 5 different thread parameters as 4, 8 and 16, 32 and 64. For the search process, 100 random words with different frequencies in the dataset are selected and the averages of the search times of these words are obtained for each algorithm.

Results:
CComp is compared to parallel gzip, pbzip2, parallel zlib, zstd and xz. Obtained results show that CComp's compression ratios give nearly same results with other word-based algorithms. The compression time results are close to other algorithms. While the decompression algorithm shows better results than all algorithms except Zstd in terms of time, it gives better results than all algorithms as the file size grows. In the compressed search process, which is the main focus of this study, search results are obtained approximately 6 times faster than the Zstd algorithm, which gives the best results after CComp.

Conclusion:
CComp has been able to increase the performance of compression, search and decompression operations by allowing them to be performed in parallel. In addition, it has an additional advantage over other algorithms in the compressed search phase, thanks to the selection of symbols as words, different from the symbols selected for other algorithms. Therefore, CComp can be used as a better alternative to existing algorithms that support compressed search.

Journal of the Faculty of Engineering and Architecture of Gazi University 39:3 (2024) 1933-1943

Mühendislik Mimarlık Fakültesi Dergisi
Journal of The Faculty of Engineering and Architecture of Gazi University

Elektronik / Online ISSN : 1304 - 4915
Basılı / Printed ISSN : 1300 - 1884

CComp: Sıkıştırılmış kelime arama için paralel bir sıkıştırma algoritması

Emir Öztürk*, Altan Mesut
Trakya Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, 22030, Edirne, Türkiye

ÖNEÇIKANLAR

- Sıkıştırılmış arama için parallel sıkıştırma algoritması
- Birçok algoritmadan daha hızlı açma hızı
- En iyi algoritma olan Zstd'den 8 kat daha hızlı sıkıştırılmış arama

Makale Bilgileri
Araştırma Makalesi
Geliş: 05.11.2022
Kabul: 27.11.2023

DOI:
10.17341/gazimmfd.1199811

Anahtar Kelimeler:
Sıkıştırılmış arama,
veri sıkıştırma,
paralel programlama,
metin sıkıştırma

ÖZ

Üretilen verilerin saklamasında alandan tasarruf etmek önemlidir. Sıkıştırma algoritmaları bu tasarrufu sağlamak amacıyla kullanılmaktadır. Saklanmak istenen veri bir kere sıkıştırılmakta fakat üzerinde arama yapmak amacıyla defalarca erişilmektedir. Bu sebeple sıkıştırılmış verinin en büyük dezavantajı bu verinin kullanılmak istendiğinde açılması gerekliliğidir. Hızlı bir açma algoritması ile veya açma işlemine ihtiyaç duymayan bir sıkıştırılmış arama yönteminin kullanılması ile bu dezavantajlı durum giderebilir. Sıkıştırılmış arama hem arama uzayının küçük olması hem de açma yapmaması sayesinde aç-ve-ara yöntemlere göre daha hızlı sonuçlar elde edebilmektedir. Bu makalede sıkıştırılmış arama desteği sunan paralel yarı statik kelime tabanlı bir sıkıştırma algoritması olan CComp sunulmuştur. CComp'un amacı diğer paralel sıkıştırma algoritmalarının hızında sıkıştırma-açma ve daha hızlı sıkıştırılmış arama yapmaktır. CComp sıkıştırma, açma ve arama işlemlerini paralel olarak gerçekleştirmektedir. CComp diğer paralel yöntemler ile karşılaştırılmıştır. Sonuçlarda gösterildiği gibi CComp'un sıkıştırma oranları diğer kelime tabanlı algoritmalarla paralel sonuçlar vermektedir. Sıkıştırılmış arama işleminde ise daha önce en iyi sonucu veren Zstd algoritmasına göre yaklaşık 7 kat daha hızlı arama sonuçları elde edilmiştir. Bu sonuçlar ile CComp sıkıştırılmış arama desteği sunan algoritmalara daha iyi bir alternatif olarak gösterilebilmektedir.

CComp: A parallel compression algorithm for compressed word search

HIGHLIGHTS

- Parallel compression algorithm for compressed search
- Faster decompression than most algorithms
- 8 times faster compressed search than the best algorithm which is Zstd

Article Info
Research Article
Received: 05.11.2022
Accepted: 27.11.2023

DOI:
10.17341/gazimmfd.1199811

Keywords:

ABSTRACT

It is important to save space storing the generated data. To achieve this, compression algorithms are used. Stored data is compressed once but accessed many times to search on it. For this reason, the biggest disadvantage of compressed data is that it needs to be decompressed when it will be used. This disadvantage can be eliminated by using a fast decompression algorithm or a compressed search method that does not require decompression. Compressed search can achieve faster results than open-and-search methods, thanks to its small search space and not using decompression. In this article, CComp, a parallel semi-static word-based compression algorithm that supports compressed search, is presented. The purpose of CComp is to obtain faster search results while compressing-decompressing at the speed of other parallel compression algorithms. CComp performs these operations in parallel. CComp has been compared to other parallel methods. As shown in the results, the compression ratios of CComp give results in parallel with other word-based algorithms. In the compressed search process, results were obtained approximately 7 times faster than the Zstd algorithm, which gave the best results before. With these results, CComp can be shown as a better alternative to algorithms that support compressed search.

*Sorumlu Yazar/Yazarlar / Corresponding Author/Authors : *emirozturk@trakya.edu.tr, altanmesut@trakya.edu.tr / Tel: +90 535 237 1982

1. Giriş (Introduction)

Üretilen verideki dramatik artışla birlikte veri sıkıştırmanın önemi artmaktadır [1]. Veri sıkıştırma, ses, görüntü, metin ve video gibi çeşitli veri türleri için kullanılabilmektedir [2]. Bu yöntemlerden bazıları kayıplı sıkıştırma gerçekleştirirken, kayıpsız sıkıştırma da yaygın olarak kullanılmaktadır [3]. Kayıplı sıkıştırma genellikle görüntü, video veya ses üzerindeki gözle veya kulakla algılanamayan ayrıntıların atılmasıyla gerçekleştirilir. Öte yandan, literatürde yapılmış birçok çalışmada metin üzerinde bilgi kaybını önlemek için genellikle kayıpsız sıkıştırma yöntemleri tercih edilir [4–7]. Metin verisi İngilizce doğal dilde olmak zorunda değildir. İngilizceden farklı diğer doğal dillerde veya DNA dizilimi gibi farklı alfabeye sahip verilerde metin verisi bulunması mümkündür. Bu sebeple kayıpsız algoritmaların performansının arttırılabilmesi için farklı dillerde özelleştirilmiş yöntemler bulunmaktadır [8, 9]. Bundan farklı olarak DNA dizilimleri için yapılan çalışmalar da mevcuttur [10-12]. Metin verilerinin üretim hacmi her yıl bir önceki yıla göre daha da artmaktadır. Büyük veri kavramının yaygınlaşmasıyla birlikte GB'larca veri dakikalar içinde üretilmekte ve bu veriyi depolamanın maliyeti artmaktadır [13]. Bu maliyeti azaltmak için veri sıkıştırma yöntemleri ile verilerin ihtiyaç duyduğu alan azaltılabilir. Depolama maliyetleri geçmiş yıllara göre düşse de üretilen veri miktarının fazla olması ve bu miktarın her geçen gün artması veri sıkıştırmaya hala kritik bir ihtiyaç olduğunu göstermektedir.

Sıkıştırma yöntemleri, geçmiş yıllarda sıkıştırma sürelerinden bağımsız olarak sıkıştırma oranını maksimize etmeyi amaçlamıştır. Ancak günümüzde depolama kapasitesi maliyetlerinin geçmişe göre daha düşük olması ve üretilen verinin hızının artması nedeniyle sıkıştırma oranı ve sıkıştırma süresini dengelemek sadece sıkıştırma oranına odaklanmaktan daha önemli hale gelmiştir [14]. Bu nedenle son yıllarda geliştirilen yöntemler, sıkıştırma oranını maksimize etmekten çok, mümkün olan en kısa sürede kabul edilebilir oranlarda sıkıştırma elde etmeyi amaçlamaktadır [14, 15]. Sıkıştırma ve açma hızlarını artırmanın bir yolu, bu işlemleri paralel olarak gerçekleştirmektir [16-19]. Bu amaçla mevcut yöntemlerin paralel versiyonları da geliştirilmiştir [20, 21]. Metin verileri sıkıştırıldıktan sonra arama yapmanın üç yolu vardır. Bunlar aç-ve-ara, kısmi açma ve arama ve son olarak sıkıştırılmış aramadır. Aç-ve-ara yönteminde veri üzerinde arama yapmak için sıkıştırılmış verinin öncelikle tamamen açılması gerekmektedir. Bu yöntemin en büyük dezavantajı verinin içerisinde yapılacak her sorgu veya aramada verinin tamamının belleğe ya da diske açılması gerekliliğidir. Açma yönteminin hızlı olduğu varsayılsa bile her sorguda dosyanın tekrar açılması işleminin getireceği işlem ve zaman yükü oldukça fazla olacaktır.

Bu yöntemden daha iyi performans veren bir alternatif ise dosyanın yalnızca aranacak kısmını açmak ve arama yapmaktır. Fakat bu yöntemi her sıkıştırma algoritması desteklememektedir. Özellikle sıkıştırma esnasında dinamik işlem gerçekleştiren ve sembollere karşılık gelen kodların sıkıştırma esnasında değiştiği durumlara sahip algoritmalarda verinin herhangi bir noktadan itibaren açılması mümkün olmamaktadır. Arama işleminde büyük bir dosya içerisinde hangi kısmın aranacağının bilinememesi de yine bu yöntemin başka bir problemidir. Yapısal olarak bloklara ayrılmış bir veride arama alanlarının boyutları belirlenebilse de işlenmemiş olarak tutulan metin verisi üzerinde aranan kelimenin verinin hangi kısmından itibaren aranması gerektiği sorusu da cevaplanamamaktadır. Ayrıca dosyanın tamamını açma işlemine göre çok daha az bir açma yüküne sahip olmasına rağmen yine de bir açma işlemine ihtiyaç duyması bu yöntemin dezavantajı olarak kabul edilebilir. Arama işlemi için bahsedilen iki yöntemi kullanabilen yöntemler için odak noktası açma hızını maksimize etmek olmalıdır. Arama yöntemlerinin sonuncusu sıkıştırılmış veri üzerinde yapılan aramadır [22, 23]. Sıkıştırılmış arama adı verilen bu yöntem, temel olarak sıkıştırılmış veri üzerinde belirli uzunluktaki bir örüntüyü arama işlemi gerçekleştiren dizge eşleme yöntemidir [22]. Dizge eşleme ise bir metin üzerinde tekil bir parçanın veya çoklu parçaların birebir aramaya göre daha performanslı bir şekilde aranması işlemidir [24]. Sıkıştırılmış arama aç-ve-ara veya kısmi açma gerçekleştiren yöntemlere göre çok daha hızlıdır. Bunun iki sebebi bulunmaktadır. Birincisi yöntemde verilerin açılmasına gerek olmaması ve açmanın zaman maliyetinin ortadan kalkmasıdır. İkinci olarak sıkıştırılmış verilerin boyutu açık veriye göre küçük olacaktır ve aranacak sembol de sıkıştırılarak bu veri üzerinde aranacaktır. Böylece içerisinde arama yapılan verinin boyutu açık veriden küçük olduğu için arama hızı açık metin üzerinde yapılan arama işlemine göre bile daha hızlı olacaktır.

Doğal dilde saklanan verilerde genellikle kelimeler üzerinde arama işlemleri yapılacağı kabul edilebilir. N-gram tabanlı arama yöntemlerinde verilen bir kelimenin öncelikle sıkıştırılmış arama destekleyen yöntemin saklama formatına uygun bir şekilde kodlanmalı ve daha sonra oluşan kodların dosya içerisinde aranması işlemi gerçekleştirilmelidir. Bu nedenle kelimelerin sıkıştırma yönteminde sembol olarak seçilmesinin kelime arama performansına etkisi n-gram tabanlı yöntemlere göre çok daha fazla olacaktır. Arama işleminde bazı yöntemler sembolleri birebir eşlemeye çalışırken, kelimeleri arayan yöntemler de bulunmaktadır [25, 26].

Dizge eşleme ya da sıkıştırılmış arama yöntemlerinde bazı yöntemler birebir arama sonucu verirken bazı yöntemler ise yaklaşık aramayı desteklemektedir [27, 28]. Bu yöntemler yavaş bir şekilde birebir arama sonucu vermektense gerçek sonuca olabildiğince yakın fakat hızlı çalışan arama algoritmaları kullanırlar. CComp birebir eşleme ve arama gerçekleştirmektedir.

Arama işleminin performansını artırmak için indeksleme yöntemlerini kullanmak da mümkündür [29,3 0]. Ancak bu durumda indeks depolama maliyeti de dikkate alınmalıdır. Özellikle parçalı verilerin saklandığı durumda indeks veri boyutundan daha büyük olabilmektedir. Yer ihtiyacının kritik olduğu sistemlerde hızlı erişim ve arama için sıkıştırılmış arama, indekslerin kullanılmasından daha uygun olacaktır.

Bu çalışmada, paralel sıkıştırma ve açma işlemlerini gerçekleştiren ve sıkıştırılmış arama desteği sağlayan kelime tabanlı bir yöntem olan CComp önerilmiştir. CComp, paralel yöntemlerin sıkıştırma-açma hızına ulaşmayı ve diğer yöntemlere göre daha hızlı sıkıştırılmış arama gerçekleştirmeyi amaçlamaktadır.

İkinci bölümde deneysel sonuçlar için CComp ile karşılaştırılan literatürde halihazırda mevcut olan sıkıştırma yöntemleri anlatılmıştır. Üçüncü bölümde CComp sıkıştırma, açma ve sıkıştırılmış arama işlemleri detaylandırılmıştır. Dördüncü bölümde deneysel sonuçlar incelenmiş ve son bölümde sonuçlar verilmiştir.

2. Önceki Çalışmalar (Related Work)

Bu bölümde, CComp ile karşılaştırılan ve deneysel sonuçların elde edilmesinde kullanılan literatürdeki yöntemler anlatılmaktadır. Pigz, Pbzip2, Xz ve Zstd'nin seçilmesinin nedeni bu algoritmaların CComp gibi paralel olması ve ayrıca verilen tüm algoritmaların sıkıştırılmış arama desteği de sunmasıdır.

2.1. Pigz (Parallel Gzip & Zlib)

Gzip kayıpsız bir sıkıştırma yöntemi ve dosya formatıdır [31]. Unix Compress'e alternatif olarak Jean-Ioup Gailly ve Mark Adler

tarafından geliştirilmiştir. LZ77 ve Huffman kodlamasına dayalı DEFLATE algoritması üzerine kurulmuştur [32, 33]. Huffman kodlama daha sık geçen sembollerin daha küçük uzunlukta kodlar ile kodlanması için geliştirilmiş bir algoritmadır. Huffman algoritmasının halen geliştirilen varyasyonları bulunmaktadır [34].

Gzip'in paralel versiyonu, Mark Adler [20] tarafından geliştirilen Pigz (gzip'in paralel implementasyonu) olarak adlandırılır. Pigz sıkıştırma için Zlib ve paralel işlemler için pthread kitaplıklarını kullanır. Bu nedenle, gzip algoritması paralel olarak çalışabilir ve klasik gzip'ten daha iyi sıkıştırma-açma süreleri sağlar. Algoritma, giriş akışını varsayılan olarak 128 Kb'lık parçalara böler ve her bir parçayı paralel olarak sıkıştırır. Daha sonra bu parçalar sırayla tek bir iş parçacığında birleştirilir. Benzer şekilde, her parçanın kontrol değeri paralel olarak hesaplanır. Son olarak gerekli başlık oluşturulduktan sonra oluşturulan sıkıştırılmış veri dosyaya yazılır. Pigz ile sıkıştırılan akışlar, klasik gzip algoritması ile açılabilir. Benzer şekilde, klasik gzip algoritması ile sıkıştırılmış bir akışın açılması pigz ile gerçekleştirilebilmektedir.

Gzip gibi Zlib de Jean-Ioup Gailly ve Mark Adler [35] tarafından geliştirilmiştir. Gzip'e benzer şekilde, Zlib de DEFLATE tabanlıdır. DEFLATE yöntemi ile sıkıştırılan veriler üzerine gerekli başlıkların eklenmesi işlemini gerçekleştirir. Bu nedenle, Zlib uygulaması Gzip akışlarını okuyabilir veya oluşturabilir. Zlib'in başlığı, Gzip ile oluşturulan başlıklardan daha az yer kaplar. Ayrıca Gzip tek dosya sıkıştırma işlemi gerçekleştirebilirken Zlib'de böyle bir kısıtlama yoktur. Yöntemin uygulanması temel olarak Gzip ile aynı olduğundan, Pigz ile bir Zlib akışı da oluşturulabilmektedir.

2.2. Pbzip2

Bzip2, girdi verilerini bloklar halinde alan ve veriler üzerinde BWT (Burrows Wheeler Transform) ile dönüştürme işlemi gerçekleştiren bir yöntemdir [36]. BWT 3 ana süreçten oluşur [37]. İlk olarak, giriş verileri bir blok olarak alınır ve sıralanır. Sıralama işleminden sonra MTF (Move To Front) dönüşümü ile benzer veriler sıkıştırma oranını arttırmak amacı ile yan yana getirilir. Bu iki dönüşüm aşamasından sonra veriler Huffman kodlaması ile sıkıştırılır.

Pbzip2, Bzip2'nin paralel bir uygulamasıdır [21]. Bzip2 dosya formatı ile uyumludur. Sıkıştırılmış dosya, klasik Bzip2 ile açılabilir. pthread kütüphanesi ile geliştirildiği için gcc gibi bu kütüphaneye destek sunan bir C++ derleyicisinin olduğu ortamlarda çalıştırılabilir. Pbzip2, bloklar için BWT'nin adımlarını paralelleştirir. Her blok paralel olarak sıkıştırılır ve oluşturulan çıktı akışına iletilir.

2.3. Zstd

Zstd (Zstandard), 2016 yılında Yann Collet tarafından geliştirilen kayıpsız bir sıkıştırma yöntemidir [38]. DEFLATE'e oldukça yakın sıkıştırma oranlarına çok daha yüksek sıkıştırma hızları ile ulaşmayı amaçlar. Özellikle açma hızının yüksek olması hedeflenmiştir. Zstd'nin en yüksek sıkıştırma seviyelerinde LZMA'ya yakın sıkıştırma oranları sağlanabilmektedir. Zstd, LZ77'yi temel alır ve LZ77'ye göre daha geniş pencere boyutları kullanmaktadır. Entropi kodlama aşamasında, Asimetrik sayı sistemleri'nin (Assymmetric numeral systems - ANS) hızlı tablolu versiyonu olan tANS'ı, Sonlu Durum Entropisini (Finite State Entropy) [39] ve Huffman kodlamasını birlikte kullanır. Zstd ayrıca paralel bir versiyon da içermektedir.

2.4. Xz

Xz Utils, Linux ve Windows'ta bulunan bir yazılım paketidir [40]. Paket, LZMA ve Xz sıkıştırma algoritmalarını içerir. Xz, Gzip ve Bzip2'den daha iyi sıkıştırma oranları elde eder, ancak sıkıştırma hızı diğer algoritmalardan çok daha yavaştır. Xz'nin sıkıştırma ve açma işlemleri Bzip2'den daha hızlıdır, ancak Gzip'ten daha yavaştır. Varsayılan olarak kullandığı algoritma LZMA2'dir. Xz algoritması üzerinde BCJ (Branch / Call / Jump) filtreleri uygulanarak sıkıştırma oranı arttırılabilmektedir.

3. Önerilen Yöntem **(Proposed Method)**

CComp, sıkıştırılmış arama desteğine sahip yarı statik, kelime tabanlı bir sıkıştırma yöntemidir. Yöntem, diğer yarı statik yöntemler gibi ilk aşamada sembolleri ve bu sembollerin frekanslarını elde eder ve bu sembollerin sözlüklerini oluşturur. Sözlüklerin oluşturulmasının amacı, sık kullanılan sembolleri daha kısa olan kod sözcükleri ile kodlamaktır. Kelime tabanlı bir yöntem olduğu için CComp sembol olarak kelimeleri kullanmaktadır. CComp, sıkıştırma, açma ve sıkıştırılmış arama işlemlerini paralel olarak gerçekleştirir. Yöntemin sıkıştırma, açma ve sıkıştırılmış arama aşamaları alt bölümlerde verilmiştir.

3.1. Sıkıştırma (Compression)

CComp yarı statik bir yöntem olduğu için sıkıştırma işlemi iki aşamada gerçekleştirilir. İlk aşamada sıkıştırılacak olan veriden kelimeler elde edilir. Verilen metindeki bu kelimelere göre sözlükler oluşturulur ve bu sözlüklerdeki kelimelere birer kod atanır. İkinci aşamada ise bu sözlükler kullanılarak kelimelerden oluşturulan kod sözcükleri çıktı akışına yazılır.

Kodlama için bir veya iki bayt büyüklüğünde kod sözcükleri kullanılmaktadır. Her bir kod sözcüğünün ilk dört biti sözlük numarasını (indeksini) içerir. Ardından bir bit kod sözcüğünün uzunluğunu belirlemek için bayrak olarak kullanılır. Bu bayrak değerine göre son 3 veya 11 bit, sözlük indeksi veya sözlüklerde o kelime yoksa kelimenin uzunluğu için kullanılır. 3 bit kullanıldığında 8, 11 bit kullanıldığında ise sözlüklerde 2048 adet kelimeye kod atanabilmektedir.

Sıkıştırma işleminde 15 farklı sözlük kullanılmaktadır. 15 adet sözlüğün seçilmesinin nedeni, kod sözcüklerinde sözlük numarasına karşılık gelen değeri temsil etmek için 4 bitin ayrılması ve sözlüklerde olmayan kelimeleri tanımlamak için 16. değerin (binary:1111, decimal:15) kullanılmasıdır. 15 sözlükten 14'ü kelimelerin ilk harflerine göre ayrılmıştır. Son sözlük ise noktalama karakterlerinden oluşan gruplar ve İngilizce olmayan harflerle başlayan kelimeler için kullanılır. Sözlük numarasını temsil etmek için 4 bit yerine 3 bit kullanılırsa 7 farklı sözlüğün her birinde 16 & 4096, 5 bit kullanılırsa 31 farklı sözlüğün her birinde 4 & 1024 adet kelime saklanabilir. Yapılan testler sonucunda sıkıştırma oranı, sıkıştırma hızı ve açma hızı açısından en verimli seçeneğin 4 bit olduğu görülmesi nedeniyle algoritmada bu değer kullanılmıştır.

İngilizcede 26 harf vardır ve bu harflerle başlayan kelimeler 14 sözlüğe dağıtılmalıdır. Bu durumda bir sözlükte ortalama iki adet farklı harfle başlayan kelime grubu saklaması gerektiği anlamına gelir. Bu da hangi sözlüklerde hangi harfle başlayan kelimelerin saklanacağının belirlenmesi anlamına gelmektedir. Sözlüklere eklenecek kelimeler sıkıştırma esnasında girdi dosyasından çıkartılmaktadır. Her farklı sıkıştırma işlemi için ise bu kelimeler değişecektir. Bu sebeple harflerin her işlemde belirlenmesi için bir frekans sıralaması aşaması gerekmektedir. Bu da algoritmanın yavaşlamasına sebep olacaktır. Bunun yerine İngilizce metinlerden çıkartılan istatistiksel veriler kullanılarak hangi harflerin sözlükleri nasıl paylaşacağı önceden belirlenmiş ve harfler sözlüklere dağıtılmıştır. 26 harfin 14 sözlüğe dağıtım işlemini gerçekleştirmek için, İngilizce metinlerden oluşturulan bir veri seti kullanılarak

kelimeler ilk harflerine göre (büyük harfler de küçük harf gibi kabul edilerek) gruplandırılmış ve toplam kelime sayısına göre yüzdeleri (frekansları) elde edilmiştir. En yüksek frekansa sahip ilk iki harf (t ve a) 0 ve 1 numaralı sözlüklere atanmış, kalan 24 harf ise 2-13 numaraları arasındaki 12 farklı sözlüğe ikişer ikişer atanmıştır. Harfler ikili olarak birleştirilirken "frekansı en yüksek harf (h) en düşük harf (z) ile, frekansı en yüksek ikinci harf (s) en düşük ikinci harf (x) ile, …" sırası gözetilmiştir. İngilizce dili için kelimelerin ilk harflerinin frekansları ve bu frekanslar gözetilerek atandıkları sözlük numaraları Tablo 1'de verilmiştir.

Zipf yasasına göre, sıklığa göre sıralanan her kelime, kendisinden önceki kelimenin frekansının yarısına sahip olacaktır [41]. Böylece kelimelerin frekans sıralaması yapıldığında düşük frekansa sahip kelimelerle yüksek frekansa sahip kelimeler arasındaki frekans farkının çok fazla açılacağı ön görülebilir. Sıkıştırma aşamasında çok sık geçen bir kelimenin kodlanması sıkıştırma oranını arttırmak adına önemlidir. Kelime frekansları çıkarılan tabloda elde edilen değerler ile sıkıştırılacak veri için farklı olacak olsa da dilin yapısı ve kelimelerin istatistiksel olarak daha sık geçme olasılıklarının elde edilmesi ile oranın yüksek tutulması amaçlanmıştır.

Bu amaç doğrultusunda Tablo 1'de gösterildiği gibi aynı sözlüğü paylaşacak olan iki harften birinin yüksek frekans, diğerinin düşük frekansa sahip olması tercih edilmiştir. Çünkü iki harf de yüksek frekansa sahip olursa, her sözlükte yalnızca 2048 kelime saklandığı için yüksek frekanslı bazı kelimeler sözlüğe dahil olamayacaktır. Böyle bir durumda düşük frekanslara sahip kelimeler 14 sözlüğün herhangi birinde bulunurken yüksek frekanstaki sözlüklerin dışarıda kalması olasılığı bulunmaktadır. Birbirinden farklı sözlükler olsalar da 14 farklı sözlüğün içerisinde olabildiğince yüksek frekansa sahip olma ihtimali olan kelimelerin bulunması önemlidir.

Tablo 1. İngilizce metinlerdeki kelimelerin ilk harflerinin frekansları ve atanan sözlük numaraları
(Frequencies of the first letters of words for English texts, and assigned dictionary numbers)

Harf	Frekans (%)	Sözlük numarası	Harf	Frekans (%)	Sözlük numarası
t	17,48	0	l	2,45	13
a	12,14	1	n	2,19	13
h	7,79	2	r	2,10	12
s	7,51	3	e	1,88	11
w	7,43	4	g	1,70	10
o	6,75	5	u	1,11	9
i	4,86	6	y	0,98	8
b	4,47	7	v	0,66	7
m	4,13	8	k	0,60	6
f	4,02	9	j	0,25	5
c	3,60	10	q	0,23	4
d	2,97	11	x	0,02	3
p	2,67	12	z	0,01	2

Örneğin, yüksek frekanslı 'a' ve 't' ile başlayan kelimelerin aynı sözlükte birleştirildiğini varsayarsak, frekanslara göre sıralama yapıldığında her iki harf ile başlayan kelimeler de yüksek frekansa sahip olduğundan sözlüğe her harften ortalama 1024 kelime dahil edilecektir. 'a' ve 't' harflerinin 1024'üncü kelimeleri 18 ve 11 frekanslarına sahiptir. Düşük frekanslı kelimelere sahip 'v' ve 'w' harfleri için ise bu değerler 4'tür. Bu durumda 'a' ve 't' harfleri Sözlük 1'de ve 'v' ve 'w' harfleri Sözlük 2'de birleştirildiğinde Sözlük 1'de 11 frekansındaki bir kelime elenirken sözlük 2'deki 4 frekansına sahip bir kelime ise kodlama sözlüğünde yer alacaktır. Bu da karşılaşma olasılığı daha yüksek olan bir kelimenin elenmesi sebebiyle sıkıştırma oranının düşmesi olasılığının artması anlamına gelmektedir.

Türkçe metinlerin sıkıştırılmasında kullanılmak amacıyla benzer bir çalışma Türkçe metinler içeren bir veri seti ile de yapılmıştır. Bu istatistiksel çalışma sonucunda elde edilen kelimelerin ilk harflerine göre frekans sıralaması ve bu doğrultuda belirlenen sözlük numaraları Tablo 2'de verilmiştir.

Tablo 2. Türkçe metinlerdeki kelimelerin ilk harflerinin frekansları ve atanan sözlük numaraları
(Frequencies of the first letters of words for Turkish texts, and assigned dictionary numbers)

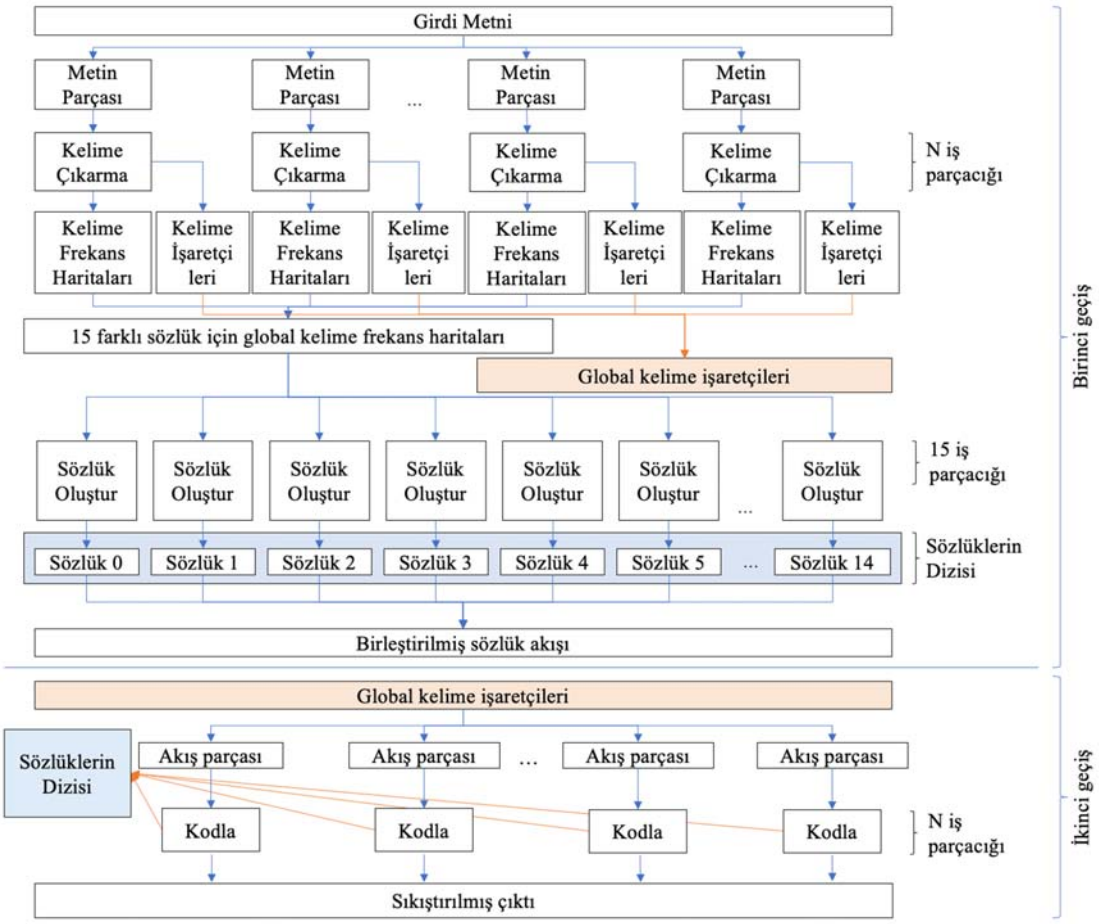
Harf	Frekans (%)	Sözlük numarası	Harf	Frekans (%)	Sözlük numarası
b	10,69	0	ç	2,40	13
k	8,14	1	ö	1,88	12
a	7,67	2	p	1,87	11
d	7,55	3	ş	1,34	10
s	6,96	4	f	1,20	9
y	6,80	5	ü	1,19	8
i	6,24	6	c	1,17	7
g	5,87	7	n	1,15	6
t	4,94	8	u	0,90	5
o	4,47	9	r	0,85	4
e	4,14	10	z	0,67	3
v	3,83	11	l	0,52	2
m	3,74	12	ı	0,17	1
h	3,54	13	j	0,10	0

CComp'un ilk aşamasında sözlüklerin nasıl oluşturulduğu ve ikinci aşamasında bu sözlükler ile nasıl sıkıştırma işleminin yapıldığı Şekil 1'de verilmiştir.

Şekil 1'de görüldüğü gibi yöntem, giriş metnini parametre olarak verilen iş parçacığı sayısına göre böler. Her iş parçacığı metnin kendisine iletilen kısmından kelimeleri çıkarır ve bu kelimeler üzerinde frekans sayma işlemi yapar. Bu aşamada her iş parçacığının elinde lokal olarak frekans değerleri bulunmaktadır. 15 farklı sözlük için lokal olan kelime frekansları daha sonra ortak tek bir kelime frekans tablosunda toplanır. Başlangıçta tek bir tablo üzerine birden fazla iş parçacığı ile yazma işlemi gerçekleştirilememekte ve frekans verisinin bozulma ihtimali bulunmaktadır. Bunu engellemenin yolu kilit (lock) mekanizmasının kullanılması olsa da bu durumda her iş parçacığı sözlükleri beklediği için performans açısından olumsuz bir sonuç oluşacaktır. Tüm lokal haritaların tek bir noktada birleştirme işleminin maliyeti bu bekleme maliyetinden düşük olmaktadır.

Kelimeleri elde etmek için boşluksuz kelime modeli (Spaceless Word Model) kullanılmıştır. Bu model sembolleri kelimeler ve noktalama işaretleri olmak üzere ikiye ayırmaktadır. Noktalama işaretlerini tek tek işlemek yerine ardışık olan noktalama işaretlerini de gruplar ve bu grubu bir sembol olarak belirler. Spaceless Word Model noktalama işaretlerinin başlangıç ve bitiş noktalarını belirleyerek dinamik uzunlukta bir grup oluşturup sembol haline getirmektedir. Grupların sabit bir boyutu bulunmamaktadır. Aynı kelimenin büyük harfli hali ile küçük harfli hali aynı sözlük içerisinde farklı kelimeler olarak saklanmıştır. Örneğin 'the', 'The' ve 'THE' kelimeleri 3 farklı kelime olarak Tablo 1'den elde edileceği gibi 0 numaralı sözlükte saklanabilecektir. Herhangi bir kelime küçültme büyütme durumunda kayıpsız sıkıştırma gerçekleştirmek için bu kelimenin değiştirildiğine dair bir işaret tutulması gerektiğinden bu işlem gerçekleştirilmemiştir.

Kelime çıkarma aşamasında bu kelimeler ilk harflerine göre 15 farklı sözlüğe kaydedildiği ve bu sözlükler üzerinde paralel arama işlemleri yapılabildiği için performans kazancı sağlanmaktadır. Arama işlemleri veriyi değiştirmediği için eşzamanlı olarak erişim mümkündür. Aynı zamanda sözlüklerin eleman sayısı arttıkça sözlük



Şekil 1. Sıkıştırmanın birinci ve ikinci aşamaları (First and second stages of compression)

üzerinde arama maliyetinin de arttığı bilinmektedir. Bu sebeple sözlüklerin 15 küçük parçaya bölünmesi ile bu maliyet de düşürülmüştür.

Kelimeler ilk aşamada elde edilirken, sıkıştırmanın ikinci aşamasında tekrar bu elde etme işlemini tekrarlamamak ve sıkıştırma performansını arttırmak amacıyla kelimelerin başlangıç indekslerini ve uzunluklarını saklayan bir kelime işaretçi vektörü de çıkarılır. Böylece ikinci aşamada kelime çıkarma maliyeti ortadan kalkar.

Bu aşama tamamlandıktan sonra her sözlük için en sık geçen 2048 kelime seçilerek 15 farklı sözlüğe (Sözlükler dizisi) yazılır. Burada 2048 kelime seçilmesinin nedeni, kelimelerin ikinci aşamada maksimum 11 bit ile kodlanabilmesidir. Son olarak, ilk aşamanın sonunda oluşturulan sözlükler açma aşamasında ihtiyaç duyulacağı için tek bir akışta birleştirilir. Sözlükler oluşturulduktan sonra ikinci aşamaya geçilir.

İkinci aşamada, ilk aşamada oluşturulan kelime-işaretçi vektörü ve sözlük dizisi girdi olarak kullanılır. Kelime-işaretçi vektörü, bu aşamada belirtilen sayıda iş parçacığına bölünür. Her iş parçacığı sözlüklerde bu kelimelerin ilk harflerine göre ayıklanmış kelimeleri arar. Bu aşamada her kelime için 15*2048=30720 kelimelik tek bir sözlüğe sorgu atmak yerine ilk harf kontrolü yapılarak 2048 elemanlı bir sözlükte arama gerçekleştirilir. Kelimenin sözlüklerde bulunup bulunmamasına bağlı olarak belirli kurallar dahilinde bir kod sözcüğü oluşturulur ve bu kod sözcüğü çıktıya yazılır. Bu kuralların adımları Şekil 2'de gösterilen sözde kodda verilmiştir.

```
Girdi akışından bir kelime (W) oku
W[0]'dan sözlük numarasını bul (ID)
Eğer Sözlük[ID] W'yi içeriyorsa
   KodSözcüğü = Sözlük[ID][W]
   Eğer KodSözcüğü < 8 ise
      Çıktıya 1 byte yaz
      {ID(4 bit) + "0" (1 bit) + KodSözcüğü (3 bit)}
   Değilse
      Çıktıya 2 byte yaz
      {ID(4 bit) + "1" (1 bit) + KodSözcüğü (11 bit)}
Değilse
   Eğer Uzunluk(W) < 8 ise
      Çıktıya 1 byte yaz
      {15(4 bit) + "0" (1 bit) + Uzunluk(W) (3 bit)}
      Çıktıya W kelimesini kodlamadan yaz
   Değilse
      Çıktıya 2 byte yaz
      {15(4 bit) + "1" (1 bit) + Uzunluk(W) (11 bit)}
      Çıktıya W kelimesini kodlamadan yaz
```

Şekil 2. Sıkıştırma algoritması için sözde kod
(Pseudocode for compression algorithm)

Sözlük numarası 4 bit ile kodlanır. 15 değeri, sözlükte bir kelimenin bulunmadığını açma işlemine belirtmek için kullanılır. Bu nedenle bölüm başında belirtildiği gibi sözlük sayısı 15 adet (0-14 arası) olarak seçilmiştir. Yöntem, kelimeleri 1 bayt veya 2 bayt olarak kodlayabilir.

Bu nedenle, açma yönteminin akıştan 1 bayt mı yoksa 2 bayt mı okuyacağını tespit edebilmesi için 4 bitlik sözlük numarasından sonra 1 bit ayrılmıştır. "0" biti, açma yönteminin 1 bayt okuyacağı anlamına gelirken "1" biti, yöntemin kod sözcüğünü doğru bir şekilde tespit etmek için akıştan bir bayt daha okuması gerektiğini belirtir. Kodlama 1 ve 2 bayt olarak yapıldığından, yüksek frekans değerlerine sahip kelimeler 1 bayt ile kodlanabilirken, daha düşük frekansa sahip kelimeler 2 bayt ile kodlanır. Burada amaç en sık geçen kelimelerin olabildiğince en kısa şekilde ifade edilebilmesidir. Ayrıca sözlüklerde bulunmayan kelimeleri ifade etmek için kullanılan kaçış karakteri de bu kelimelerin uzunluk değerlerine göre 1 veya 2 bayt olabilir.

Desteklenen en uzun kelime uzunluğu 11 bit olarak kodlanabilir. Bu, 2^11-1 = 2047 uzunluğunda bir kelimenin saklanmasına izin verir. Bu uzunlukta bir kelimeyle karşılaşmak pek olası olmasa da yöntem ilk aşamada kelimeleri elde eder ve 2047 değerine sahip bir kelime bulunursa kelime çıkarma işlemi esnasında kelimeyi bu kısımdan böler.
Sıkıştırma örneği için "Onlar her şeyi affederler" cümlesindeki kelimelerin sözlük indeksleri ve kelime indeksleri Tablo 3'te verilmiştir.

Tablo 3. Örnek kelime indeksler (Example words and indexes)

Kelime	Sözlük Numarası	İndeks değeri	Uzunluk
Onlar	9	1	5
her	13	10	3
şeyi	-	-	4
affederler	-	-	10

Tabloda verilen değerler tüm farklı sıkıştırma adımlarını gösterecek şekilde seçilmiş ve örnek Türkçe olduğu için sözlük numaraları Tablo 2'den elde edilmiştir. Sıkıştırma algoritması, kodlama aşamasında "Onlar" kelimesi ile karşılaştığında, kelime "O" harfi sözlüğünde olduğundan ve kelimenin bu sözlükteki sırası (indeks değeri) 8'den küçük olduğundan, ilk 4 bit sözlük numarası "9" olarak kodlanır. Sonraki 1 bit "0" olur ve son 3 bit "1" indeks değeri olarak kodlanır. Oluşturulan kod, 1 bayt olarak "1001|0|001 (145)" değerini alır.

İkinci adımda, "her" kelimesi ile karşılaşıldığında, "13" sözlük numarası 4 bit olarak kodlanır. Bu kelimenin indeks değeri 8'den büyük olduğu için bir sonraki bit değeri "1" olarak ayarlanır. Ardından kalan 11 bite kelime indeksi olan "10" değeri atanarak 2 baytlık bir çıktı oluşturulur. Elde edilen 2 bayt "1101|1|00000001010 (55306)" değerini alır.

"şeyi" kelimesi sözlükte bulunamadığı için kodlanmadan çıktı akışına yazılacaktır. Kelime uzunluğu 4 olduğu için 3 bit ile kodlanabilir. Böylece 4 bitlik "15" değeri, 1 bitlik "0" değeri ve 3 bit uzunluğundaki "4" değeri kodlanmış olacaktır. Ortaya çıkan bayt "1111|0|100" (244) değerini alır ve ardından bu değere "şeyi" kelimesi eklenir. Buradaki "15" değeri, açma yöntemine kodlanmamış bir kelimeyi belirtir.

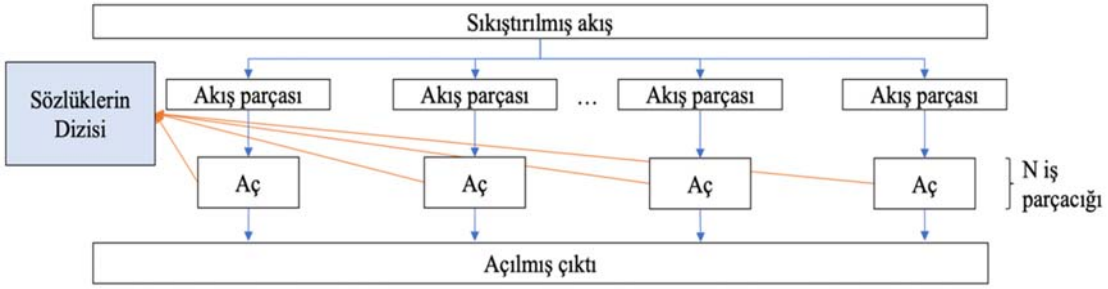
Son olarak, "affederler" kelimesinin de sözlükte olmaması nedeniyle önce "15" değeri 4 bit olarak kodlanır. Kelime uzunluğu 8'den büyük olduğundan, 1 bitlik bir "1" değeri ve ardından 11 bitlik bir uzunluk değeri (10) kodlanır. Ortaya çıkan bayt değeri "1111|1|00000001010 (63498)" olacaktır. Bu baytlardan sonra, kelimenin kendisi çıktı akışına kodlanır. Böylece, nihai çıktı akışı [145 55306 244 şeyi 63498 affederler] olacaktır.

İkinci aşama paralel olarak gerçekleştirildikten sonra iş parçacıklarından elde edilen sıkıştırılmış akışlar dosyaya yazılır. C++'ın paralel kütüphanesi vektörler üzerinde parçacıklar arasında bir engelleme olmadan (non-blocking) çalışabildiğinden, çıktı vektörü önceden oluşturulur ve bu vektörün farklı bölümlerine yazma işlemi veri bütünlüğünü etkilemeden paralel olarak yapılabilir. Her iş parçacığına vektörün belirli bir bölümü tahsis edilir ve iş parçacıkları birbirlerini engellemeden aynı vektör üzerine yazma işlemi gerçekleştirirler. Bu işlem ikinci geçiş sırasında paralel akışların çıktılarını birleştirme (reduce) maliyetini ortadan kaldırmayı amaçlamaktadır.

3.2. Açma (Decompression)

Açma sırasında sıkıştırma aşamasında oluşturulan sözlük dosyası tek parça olarak okunur ve 15 farklı sözlüğe bölünerek belleğe çıkarılır. Daha sonra kodlanmış akış okunur ve açma için parametre olarak verilen iş parçacığı sayısına bölünür. Her parça, kod çözme için iş parçacıklarına verilir. Kodlama aşamasında, bu parçaların sıkıştırılmamış boyutları da sıkıştırılmış dosyaya tamsayı olarak yazılır. Bu sayede açılmış dosyanın boyutu başlangıçta bilinir ve kodlama aşamasında olduğu gibi tek bir çıktı vektörü birden fazla iş parçacığının aynı anda kullanabilmesi için oluşturulur. Bu vektörün farklı bölümlerinin indeksleri, parametre olarak iş parçacıklarına iletilir. Her iş parçacığı kendi parçasındaki kodları okur ve bu vektöre kodu çözülmüş metni yazar. Böylece, bölünmüş vektörleri birleştirme maliyeti ortadan kalkar. Sözlük arama işlemleri verileri değiştirmediği için sözlüklerde okuma işlemleri iş parçacıkları arasında bir kilit mekanizması (lock) kullanılmadan yapılabilir. Açma işlemi adımları Şekil 3'te verilmiştir.

Açma aşamasında, okunan bir baytın ilk 4 biti kontrol edilir. Bu bitlerin değeri 15'ten küçük ise bu değerin bir sözlük numarası olduğu söylenebilir. Daha sonra 5. bite bakılarak kod sözcüğünün kalan 3 bit veya 11 bit ile temsil edileceği belirlenir. Buna göre, kod sözcüğü elde edildikten sonra, bu kod sözcüğüne karşılık gelen kelime sözlükten okunur ve çıktı akışına yazılır. İlk dört bitin değeri 15 ise, bu kelimenin sözlüklerde bulunmadığı ve akışa kodlanmamış bir biçimde yazıldığı anlamına gelir. Yine 5. bite bakılarak 3 veya 11 bit okunur ve kelime uzunluğu elde edilir. Daha sonra bu uzunlukta verilen kadar bayt değeri akıştan okunur ve çıktı akışına yazılır. Tüm iş parçacıkları işlemlerini bitirdiğinde çıktı akışı dosyaya yazılır. Örneğin, bir önceki bölümde verilen sıkıştırılmış akışı [145 55306 244 şeyi 63498 affederler] açmak için, yöntem önce bir bayt okur ve ikili (1001|0|001)



Şekil 3. Açma aşaması (Decompression stage)

olarak “145” değerini alır. İlk dört bitten kelimenin indeksi “9” olan sözlükten okunacağı belirlenir. Bir sonraki bitin “0” değerine sahip olması ile kelime indeksinin son 3 bitte olduğu belirlenir. Bu 3 bitten “1” değeri elde edilir ve “Onlar” kelimesi “0” numaralı sözlükten “1” indeksi ile okunur ve çıktı akışına yazılır.

55306 değeri için başlangıçta değerin ilk baytı (1101|1|000) olarak okunur. İlk 4 bitin değerine bakılarak kelimenin 13 numaralı sözlükten okunacağı belirlenir. Sonraki 1 bit ile sözlükte ilgili kelimenin indeks değerinin 11 bit olarak okunacağı belirlenir ve akıştan bir bayt daha okunur ve ilk baytın son 3 biti ile birleştirilir (00000001010). 10 değeri elde edildiği için 13 numaralı sözlüğün 10. kelimesi “her” olarak sözlükten okunur ve çıktı akışına yazılır.

55306'dan sonra karşılaşılan 244 (1111|0|100) değeri için ilk 4 bit “15” olarak okunur ve kod çözücü kelimenin sözlükte bulunmadığını algılar. Bir sonraki bit, kelime uzunluğunu belirlemek için okunur. 0 ile karşılaşıldığından sonraki 3 bit ile kelime uzunluğu “4” olarak elde edilir ve sıkıştırılmış akıştan 4 bayt daha okunarak “şeyi” kelimesi çıktı akışına yazılır.

Son olarak, 63498'in ilk baytı okunur (1111|1|000). İlk 4 bit “15” ve sonraki bit 1 olduğundan, bir bayt daha okunur ve kalan 3 sıfır bit ile birleştirilir (000|00001010). Elde edilen “10” değeri ile sıkıştırılmış akıştan 10 karakter okunur ve çıktı akışına yazılır.

3.3. Sıkıştırılmış arama (Compressed Search)

Sıkıştırılmış arama yapabilmek için verilen kelimenin de sıkıştırılması gerekmektedir. Daha sonra kelimenin kodlanmış halinin sıkıştırılmış metin üzerinde aranması işlemi gerçekleştirilir. Arama adımları Şekil 4'te verilmiştir.

Kelime öncelikle ilk harfine göre 15 sözlükten birinde aranır. Kelime bulunursa, kod sözcüğü elde edilir. Bulunamazsa, sıkıştırma aşamasında belirlenen kurallara göre bir veya iki baytlık kod sözcüğü oluşturulur ve ardından kelime ile birleştirilir. Böylece sıkıştırılmış dosyada aranacak bayt akışı elde edilmiş olur.

İkinci aşamada, sıkıştırılmış akış parametre olarak verilen iş parçacığı sayısı kadar parçalara bölünür ve oluşturulan bayt akışı her bölümde aranır. Sıkıştırılmış semboller önceki sembollere bağlı olmadığından ve farklı yerlerde farklı değerler göstermediğinden her parça içinde paralel arama işlemi yapılabilmektedir. Arama aşamasında, açma aşamasında olduğu gibi, ilk 4 bit ve uzunluk değerleri eşleşmezse, arama verimliliğini artırmak için atlama işlemleri yapılır. Arama birebir byte karşılaştırma ile yapılamamaktadır. Kodlanan kelime ile aynı baytlara sahip bir konuma gelindiğinde bu baytların başka bir kelimenin bir parçası veya uzunluk gibi bir değer olma ihtimali bulunmaktadır. Bu durumda arama yöntemi doğru sonuç vermeyecektir.

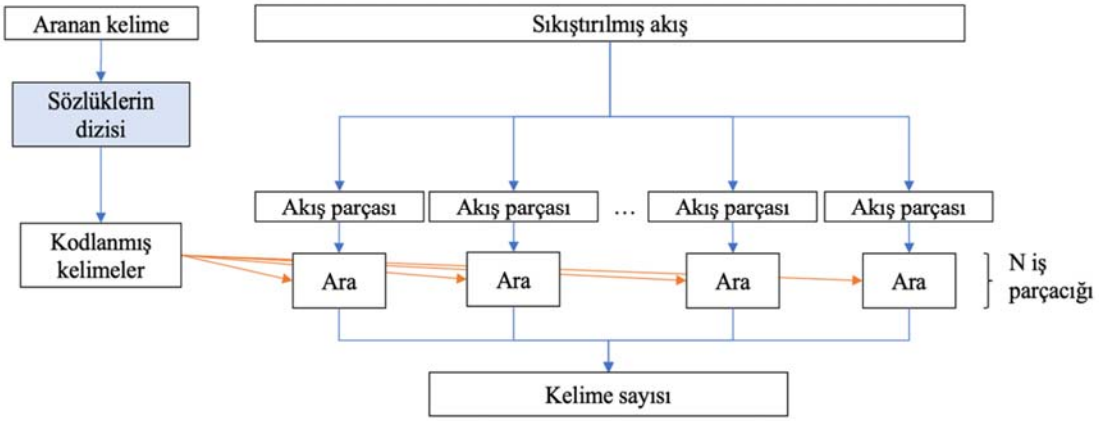
Örneğin “her” kelimesi aranmak istendiğinde, aranacak olan kodlanmış hali önceki bölümde bahsettiğimiz gibi “1101|1|00000001010 (55306)” olarak oluşturulacaktır. Bu değer byte dizisi olarak [216,10] ikilisi şeklinde gösterilebilir. Bu değerlerin aranması sırasında metin içerisinde 0 numaralı sözlüğün 472 ve 552. kelimelerinin art arda geçiyor olduğu bir kısma gelindiğinde bu kelimelerin kodları olan 2520 ve 2600 değerleri byte dizisi olarak (9,216,10,40) olarak saklanacaktır. Böyle bir durumda birebir karşılaştırma yapıldığında iki kelimenin arasındaki baytlar olan (216,10) değerine rastlandığı için her kelimesinin bulunduğu söylenecek fakat aslında bulunmamış olacaktır.

Bunun gibi yanlış bir tespit işlemini ortadan kaldırmak adına arama sırasında öncelikle ilk 4 bit kontrol edilir ve sözlük numarasının eşleşip eşleşmediği kontrol edilir. Eğer bu değerler uyuşmuyorsa 5. bit okunarak kalan 3 veya 11 bitin atlanması gerektiği tespit edilir. Eğer ilk okunan 4 bit kaçış olarak tespit edilirse ve eşleşme bulunmuyorsa yine aynı işlem adımları tekrarlanarak kelime uzunluğu bulunur ve uzunluk kadar byte atlanarak yanlış bulma işleminin önüne geçilmiş olunur.

4. Deneysel Sonuçlar (Experimental Results)

Sonuçların elde edilmesinde AMD Ryzen 2700x işlemcili ve 32 GB RAM'li Ubuntu 20.10 işletim sistemli bir sistem kullanılmıştır. CComp, -O3 parametresi kullanılarak g++ ile derlenmiştir. Uygulamanın kaynak koduna https://github.com/emirozturk/CComp adresinden ulaşılabilmektedir. Pigz, Pbzip2 ve Zstd için ubuntu apt-get ile indirilen paketler sırasıyla 2.6, 1.1.13, 1.4.8 sürümleriyle kullanılmıştır. Deneyler için kullanılan Xz, ubuntu'da 5.2.5 sürümüyle halihazırda bulunmaktadır. Tüm yöntemler süre sonuçlarını etkilememesi adına varsayılan sıkıştırma parametreleri ile kullanılmıştır.

Sıkıştırma testleri için Pizza&Chili derleminden [42] dört dosya (english50mb, english100mb, english200mb, english1024mb) ve Wikipedia makalelerini içeren enwik8 ve enwik9 dosyaları [43] ile İngilizce veri seti oluşturulmuştur. Türkçe veri seti ise SuDer derlemine [44] ait JSON dosyalarındaki metinler birleştirilerek elde edilen 1024 MB boyutundaki turkish1024mb dosyası ve bu dosyanın ilk 50/100/200 MB boyutundaki verisini içeren üç ayrı dosya ile oluşturulmuştur. SuDer derlemi Cumhuriyet ve Sabah gazetelerinden



Şekil 4. Sıkıştırılmış arama aşaması (Compressed search stage)

elde edilen Türkçe haber metinlerini içermektedir. İngilizce metin dosyaları sıkıştırılırken Tablo 1'de verilen sözlük numaraları, Türkçe metin dosyaları sıkıştırılırken ise Tablo 2'de verilen sözlük numaraları kullanılmıştır. Sıkıştırma oranı sonuçları Tablo 4'te karakter başına bit (bit per character - bpc) cinsinden verilmiştir.

Tablo 4. Bpc cinsinden sıkıştırma oranları (Compression Ratios in bpc)

Dosyalar	Pigz	Pbzip2	Zstd	Xz	CComp	MWCA*
english50mb	3,02	2,27	2,39	1,67	3,22	2,86
english100mb	3,03	2,24	2,46	1,82	3,26	2,85
english200mb	3,03	2,25	2,63	1,99	3,17	2,82
english1024mb	3,04	2,27	2,79	2,13	3,09	2,83
enwik8 (95 Mb)	2,93	2,32	2,85	2,13	3,75	3,32
enwik9 (953 Mb)	2,59	2,03	2,51	1,87	3,78	3,39
turkish50mb	2,96	2,18	2,80	2,00	3,84	3,55
turkish100mb	2,96	2,17	2,80	1,99	3,85	3,53
turkish200mb	2,96	2,17	2,80	2,00	3,85	3,53
turkish1024mb	2,96	2,18	2,81	2,01	3,90	3,57

Zlib ve Gzip üzerinde sıkıştırma yapan kütüphane Pigz olduğundan ve her ikisi için farklı süreler içerisinde aynı sonuçları verdiğinden dolayı Zlib ve Gzip üzerinde yapılan arama sonuçları Pigz sütunu altında verilmiştir. Tabloda görüldüğü gibi, CComp diğer sıkıştırma algoritmalarına göre daha düşük sıkıştırma oranlarına sahiptir. Bunun nedeni kelimelerin sembol olarak seçilmesidir. Diğer bir kelime tabanlı sıkıştırma algoritması olan MWCA'nın [26] sıkıştırma oranları da kelime tabanlı sıkıştırma algoritmalarının sıkıştırma oranlarına örnek olarak Tablo 4'te verilmiştir. Buna göre CComp kelime tabanlı sıkıştırma algoritmalarının oranlarına yaklaşmaktadır. CComp İngilizce metinlerin boyutunu ortalama %58, Türkçe metinlerin boyutunu ise ortalama %52 oranında azaltmıştır.

Sıkıştırma süreleri Tablo 5'te ve açma süreleri Tablo 6'da saniye cinsinden verilmiştir. Süre sonuçları için, her yöntem her dosya için 10 kez çalıştırılmış ve süre değerlerinin ortalaması alınmıştır. CComp 4, 8, 16, 32 ve 64 olmak üzere 5 farklı iş parçacığı parametresi ile çalıştırılmıştır. Xz için süre sonuçları, en iyi sonuçları veren 64 iş parçacığı parametresi ile verilmiştir. Tablo 5'te görüldüğü gibi, CComp'un en iyi süre sonuçları, küçük dosyalarda 8, büyük dosyalarda ise 16 iş parçacığı ile elde edilmiştir. Toplam sürelere bakıldığında CComp 16'nın en hızlı yöntem olduğu ve CComp 8'in Pigz Gzip/Zlib ile yakın sonuç ürettiği görülmektedir.

Tablo 6'da görüldüğü gibi açma sürelerinde en iyi hızlı yöntem Zstd olmuş, CComp 32 ve CComp 16 ona yakın sonuçlar vermiştir. CComp tüm iş parçacıklarında Zstd hariç diğer tüm yöntemlerden hızlı olmuş, CComp 32 en büyük dosyada Zstd'den de daha hızlı bir sonuç elde etmiştir. Pigz yöntemleri yaklaşık olarak sıkıştırma yaptıkları hızlarda açma yaparken, CComp iş parçacığı sayısına göre değişmekle birlikte, sıkıştırma hızından 2 ile 4 kat arası daha hızlı açma yapmaktadır. Pbzip2 sıkıştırma hızından yaklaşık 2 kat, Zstd 3 kat, Xz64 ise 10 kat daha hızlı açma yapabilmektedir.

Sıkıştırılmış arama sürelerini elde etmek için her dosyada veri setinden rastgele seçilen 100 kelime aranmış ve seçilen bu 100 kelimenin arama sürelerinin ortalaması elde edilmiştir. Tablo 7'de tüm yöntemlerin sıkıştırılmış arama sonuçları ile sıkıştırılmamış metin üzerinde yapılmış Grep araması sonuçları verilmiştir. Tablodan da görüleceği gibi, sıkıştırma ve açmadan farklı olarak, arama yapılırken CComp yöntemi 64 iş parçacığında en hızlı sonuçları vermiştir. Toplam süreler incelendiğinde CComp 64 ile arama yapmanın diğer yöntemler arasında en iyi süre sonuçlarını elde eden Zstd ile arama yapmaktan yaklaşık 6 kat, sıkıştırılmamış metinler üzerinde Grep ile yapılan aramadan da yaklaşık 5 kat daha hızlı olduğu görülmektedir.

Tablo 5. Saniye cinsinden sıkıştırma süreleri (Compression times in seconds)

	Pigz Gzip	Pigz Zlib	Pbzip2	Zstd	Xz64	CComp 4 iş par.	CComp 8 iş par.	CComp 16 iş par.	CComp 32 iş par.	CComp 64 iş par.
english50mb	0,277	0,277	0,600	0,272	12,723	0,422	0,387	0,425	0,624	1,061
enwik8 (95 Mb)	0,411	0,445	1,107	0,554	14,812	0,730	0,600	0,610	1,085	1,501
english100mb	0,549	0,548	1,143	0,531	15,523	0,710	0,600	0,630	0,902	1,346
english200mb	1,081	1,136	2,234	1,123	23,275	1,370	1,120	1,180	1,422	1,847
enwik9 (953 Mb)	3,662	3,792	10,635	4,796	83,542	5,990	4,580	4,140	6,706	7,178
english1024mb	5,625	5,688	11,160	5,959	122,239	6,070	4,600	4,280	5,623	5,859
turkish50mb	0,241	0,244	0,632	0,316	15,204	0,604	0,525	0,541	0,770	1,172
turkish100mb	0,464	0,456	1,210	0,642	18,692	0,970	0,786	0,792	0,945	1,351
turkish200mb	0,925	0,915	2,355	1,217	25,529	1,781	1,454	1,520	1,866	2,685
turkish1024mb	4,823	4,655	11,308	6,293	136,300	8,188	6,397	5,834	6,296	7,386
TOPLAM	18,058	18,156	42,384	21,703	467,839	26,835	21,049	19,952	26,239	31,386

Tablo 6. Saniye cinsinden açma süreleri (Decompression times in seconds)

	Pigz Gzip	Pigz Zlib	Pbzip2	Zstd	Xz64	CComp 4 iş par.	CComp 8 iş par.	CComp 16 iş par.	CComp 32 iş par.	CComp 64 iş par.
english50mb	0,201	0,221	0,377	0,087	0,539	0,175	0,228	0,204	0,216	0,157
enwik8 (95 Mb)	0,440	0,473	0,590	0,183	1,350	0,262	0,242	0,295	0,254	0,333
english100mb	0,435	0,502	0,691	0,180	1,160	0,335	0,317	0,261	0,311	0,253
english200mb	0,905	1,020	1,351	0,360	2,443	0,841	0,579	0,451	0,432	0,812
enwik9 (953 Mb)	4,007	4,672	5,358	1,551	11,744	2,681	2,339	1,697	1,683	2,656
english1024mb	4,735	5,259	6,455	1,812	12,996	3,543	2,557	1,956	1,494	2,588
turkish50mb	0,195	0,201	0,381	0,104	0,648	0,205	0,195	0,142	0,129	0,123
turkish100mb	0,377	0,381	0,704	0,186	1,255	0,394	0,387	0,260	0,227	0,217
turkish200mb	0,753	0,759	1,335	0,363	2,472	0,729	0,720	0,546	0,409	0,406
turkish1024mb	3,718	3,732	6,567	1,753	12,617	3,450	3,343	2,742	2,131	1,956
TOPLAM	15,766	17,22	23,809	6,579	47,224	12,615	10,907	8,554	7,286	9,501

Tablo 7. Saniye cinsinden arama süreleri (Search times in seconds)

	Pigz Gzip	Pigz Zlib	Pbzip2	Zstd	Xz64	CComp 4 iş par.	CComp 8 iş par.	CComp 16 iş par.	CComp 32 iş par.	CComp 64 iş par.	Grep
english50mb	0,293	0,248	1,652	0,091	0,536	0,035	0,026	0,019	0,016	0,015	0,115
enwik8 (95 Mb)	0,559	0,450	3,114	0,172	1,37	0,063	0,045	0,036	0,028	0,025	0,190
english100mb	0,580	0,495	3,277	0,173	1,162	0,064	0,045	0,032	0,025	0,022	0,229
english200mb	1,147	0,988	6,559	0,340	2,494	0,120	0,089	0,065	0,047	0,043	0,458
enwik9 (953 Mb)	5,178	5,069	29,196	1,517	12,089	0,566	0,475	0,327	0,311	0,276	1,806
english1024mb	5,837	5,987	33,555	1,801	13,480	0,554	0,451	0,320	0,281	0,253	1,712
turkish50mb	0,285	0,285	1,480	0,109	0,625	0,059	0,042	0,029	0,023	0,023	0,059
turkish100mb	0,555	0,555	2,928	0,196	1,219	0,102	0,081	0,055	0,043	0,041	0,102
turkish200mb	1,092	1,092	5,843	0,369	2,417	0,175	0,142	0,111	0,078	0,071	0,175
turkish1024mb	5,500	5,500	29,845	1,761	12,285	0,815	0,652	0,483	0,418	0,382	0,815
TOPLAM	21,026	20,669	117,449	6,529	47,677	2,553	2,048	1,477	1,270	1,151	5,661

5. Sonuçlar (Conclusions)

Depolama maliyetlerinin azalması ile yüksek sıkıştırma oranlarına olan ihtiyaç azalmıştır. Büyük verilerin hızlı depolanmasının önemi nedeniyle, hızlı sıkıştırma ve açma ihtiyacı artmıştır. Modern çok çekirdekli CPU'larla paralel işlemenin gözle görülür bir avantajı bulunmaktadır ve zamandan tasarruf etmek için sıkıştırma yöntemleri de paralel olarak çalıştırılabilmektedir.

Metin verilerinin üretimi geçmiş yıllara göre önemli ölçüde artmıştır. Bu verilerin büyük bir kısmı doğal dilden oluşmaktadır. Metin verileri sıkıştırılmış bir biçimde depolandığında, üzerlerinde bir arama yapılmak istendiğinde sıkıştırılmış verinin açılmasının bir maliyeti vardır. Bu maliyeti ortadan kaldırmak için sıkıştırılmış arama işlemi kullanılmaktadır. Metin üzerinde arama yapmanın bir yolu, metin indekslerini kullanmaktır. Ancak bu durumda indeksin oluşturulma süresi, güncellemelerin yavaşlaması ve ek depolama maliyeti dikkate alınmalıdır.

Kelime tabanlı sıkıştırma yöntemleri yüksek sıkıştırma oranları sunmasa da en önemli avantajı birçoğunun sıkıştırılmış arama desteğine sahip olmasıdır. Kodlama için en küçük birim kelime olduğunda, sıkıştırılmış metinde kelime aramak, sıkıştırılmamış metinde arama yapmaktan daha hızlı olmaktadır.

Bu çalışmada, CComp adlı paralel kelime tabanlı bir sıkıştırma yöntemi sunulmaktadır. CComp'un amacı, diğer paralel sıkıştırma algoritmalarıyla aynı hız sonuçlarına ulaşırken daha iyi sıkıştırılmış arama süreleri elde etmektir. Bu amaca ulaşmak için hem kelime sözlükleri hem de giriş akışı bölünür ve paralel olarak işlenir. Sıkıştırılmış arama yardımı ile sıkıştırılmış metinler, açma yapmadan aranabilir. Sıkıştırılmış arama, açma maliyetini ortadan kaldırarak ve arama alanını azaltarak düz metin aramaya göre büyük bir avantaj sağlamaktadır. CComp, yüksek sıkıştırma oranlarından ziyade, son yıllarda geliştirilen popüler sıkıştırma yöntemlerin de benimsediği sıkıştırılmış akışlar üzerinde hızlı sıkıştırılmış arama ve ayrıca kabul edilebilir sıkıştırma-açma hızları elde etme amacını taşımaktadır.

Elde edilen sonuçlar incelendiğinde sıkıştırma ve açma hızları, CComp'un performansının mevcut paralel algoritmalara benzer olduğunu göstermektedir. CComp'un en belirgin özelliği, sıkıştırılmış veriler üzerinde arama yapma başarısıdır. CComp ile gerçekleştirilen sıkıştırılmış arama, diğer yöntemlere göre çok daha hızlı bir biçimde gerçekleştirmektedir.

Kaynaklar (References)

1. Özköse H, Arı ES, Gencer C., Yesterday, Today and Tomorrow of Big Data. Procedia - Social and Behavioral Sciences, 195, 1042–1050, 2015.
2. Lawnik M, Pelka A, Kapczyński A., A New Way to Store Simple Text Files. Algorithms, 13, 101 2020.
3. Gupta A, Nigam S., A Review on Different Types of Lossless Data Compression Techniques. 2021.
4. Pandey M, Shrivastava S, Pandey S, Shridevi S., An Enhanced Data Compression Algorithm, 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), Vellore-Hindistan, 1–4, 24-25 Şubat 2020.
5. Ghuge S., Map and Trie based Compression Algorithm for Data Transmission, 2nd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), Bangalore-Hindistan, 137–141, 24-25 Şubat 2020.
6. Nguyen VH, Nguyen HT, Duong HN, Snasel V., n-Gram-based text compression. Computational intelligence and neuroscience, 2016, 1-12, 2016.
7. Rădescu R., Concordance Techniques in Lossless Data Compression of Text Files, 2021 12th International Symposium on Advanced Topics in Electrical Engineering (ATEE), Bükreş-Romanya, 1–4, 23-25 Mart 2021.
8. Abliz W, Wu H, Maimaiti M, Wushouer J, Abiderexiti K, Yibulayin T, Wumaier A., A Syllable-Based Technique for Uyghur Text Compression. Information, 11, 172, 2020.
9. Hilal TA, Hilal HA., Turkish Text Compression via Characters Encoding. Procedia Computer Science, 175, 286–91 2020.
10. Suneetha D, Kishore DR, Babu PN., A Compression Algorithm for DNA Palindrome Compression Technique, ITM Web of Conferences, Mumbai-Hindistan, 1-5, 27-28 Haziran 2020.
11. Murugesan G., Codon Based Compression Algorithm for DNA Sequences with Two Bit Encoding. European Journal of Molecular & Clinical Medicine, 7, 33-41, 2020.
12. Silva M, Pratas D, Pinho AJ., Efficient DNA sequence compression with neural networks. GigaScience, 9 (11), 1-15, 2020.
13. Demchenko Y, De Laat C, Membrey P., Defining architecture components of the Big Data Ecosystem, 2014 International conference on collaboration technologies and systems (CTS), Minneapolis-Minnesota-USA, 104–112, 19-23 Mayıs 2014.
14. Rattanaopas K, Kaewkeeree S., Improving Hadoop MapReduce performance with data compression: A study using wordcount job, 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), Phuket-Tayland, 564–567, 27-30 Haziran 2017.
15. Bartík M, Ubik S, Kubalik P., LZ4 compression algorithm on FPGA, IEEE International Conference on Electronics, Circuits, and Systems (ICECS), Kahire-Mısır, 179–182, 6 Aralık 2015.
16. Guerra A, Lotero J, Isaza S., Performance comparison of sequential and parallel compression applications for DNA raw data. The Journal of Supercomputing, 72, 4696–717, 2016.
17. Sun Y, Gong X, Yang Y., Data compression and parallel computation model research under big data environment, International Conference on Computer Communication and Informatics (ICCCI), Lefkoşa-Kıbrıs, 1–4, 27-29 Eylül 2017.
18. Kumar VS, Nanjundiah R, Thazhuthaveetil MJ, Govindarajan R., Impact of message compression on the scalability of an atmospheric modeling application on clusters. Parallel Computing, 34, 1–16, 2008.
19. Ahmad I, He Y, Liou ML., Video compression with parallel processing. Parallel Computing, 28, 1039–78, 2002.

20. Adler M., pigz: A parallel implementation of gzip for modern multi-processor, multi-core machines. Jet Propulsion Laboratory, 2015.
21. Gilchrist J., Parallel data compression with bzip2, Proceedings of the 16th IASTED international conference on parallel and distributed computing and systems, Dallas-USA, 559–564, 14-16 Aralık 2004.
22. Bell T, Adjeroh D, Mukherjee A., Pattern matching in compressed texts and images. 2001.
23. Mishra SP, Singh CG, Prasad R., A review on compressed pattern matching. Perspectives in Science, 8, 727–9, 2016.
24. Karcıoğlu AA, Bulut H., DNA sekansları için q-gram hash karşılaştırmasına dayalı çoklu kesin dizi eşleştirme algoritması. Gazi Üniversitesi Mühendislik Mimarlık Fakültesi Dergisi, 38, 875–88, 2022.
25. Buluş HN, Carus A, Mesut A., A new word-based compression model allowing compressed pattern matching. Turkish Journal of Electrical Engineering & Computer Sciences, 25, 3607–22, 2017.
26. Öztürk E, Mesut A, Diri B., Multi-stream word-based compression algorithm for compressed text search. Arabian Journal for Science and Engineering, 43, 8209–21, 2018.
27. Srivastav S, Singh PK, Yadav D., A Method to Improve Exact Matching Results in Compressed Text using Parallel Wavelet Tree. Scalable Computing: Practice and Experience, 22, 387–400, 2021.
28. Russo LMS, Navarro G, Oliveira AL, Morales P., Approximate String Matching with Compressed Indexes. Algorithms, 2, 1105–36, 2009.
29. Melink S, Raghavan S, Yang B, Garcia-Molina H., Building a distributed full-text index for the web. ACM Transactions on Information Systems (TOIS), 19, 217–41, 2001.
30. Bast H, Buchhold B., An index for efficient semantic full-text search, Proceedings of the 22nd ACM international conference on Information & Knowledge Management, California-USA, 369–78, 27 Ekim - 1 Kasım 2013.
31. Deutsch P, others., GZIP file format specification version 4.3. 1996.
32. Deutsch P., Rfc1951: Deflate compressed data format specification version 1.3, RFC Editor, 1996.
33. Oswal S, Singh A, Kumari K., Deflate compression algorithm. International Journal of Engineering Research and General Science, 4, 430–6, 2016.
34. Aşşık MM, Oral M., Determination of canonical Huffman codeword lengths by evolution strategies. Journal of the Faculty of Engineering and Architecture of Gazi University, 38 (2), 771-780, 2023.
35. Deutsch P, Gailly J-L., Zlib compressed data format specification version 3.3, RFC 1950, Mayıs, 1996.
36. Burrows M, Wheeler D., A block-sorting lossless data compression algorithm, Digital SRC Research Report, 1994.
37. Manzini G., An analysis of the Burrows—Wheeler transform. Journal of the ACM (JACM), 48, 407–30, 2001.
38. Collet Y, Kucherawy M., Zstandard Compression and the application/zstd Media Type. RFC 8478, 2018.
39. Duda J, Tahboub K, Gadgil NJ, Delp EJ., The use of asymmetric numeral systems as an accurate replacement for Huffman coding, 2015 Picture Coding Symposium (PCS), Cairns-Avustralya, 65–69, 31 Mayıs - 3 Haziran 2015.
40. Belkov R, Kirilenko I., Compressing Embedded GNU/Linux and Windows 10 IoT Images Using XZ Utilities, 1st Scientific and Practical Conference "Software Engineering and Information Organization", SEIM-2016, St Petersburg-Rusya, 41, 22 Nisan 2016.
41. Kirby G., Zipf's law. UK Journal of Naval Science, 10, 180–5, 1985.
42. Ferragina P, Navarro G., Pizza&Chili Corpus—Compressed indexes and their testbeds. September, 2005.
43. Mahoney M., Large text compression benchmark, http://www.mattmahoney.net/dc/text.html, Yayın tarihi Eylül 15, 2022. Erişim tarihi Mayıs 19, 2022.
44. Şen, M.U., Yanıkoğlu, B., Document classification of SuDer Turkish News Corpora, In Proceedings of the 26th Signal Processing and Communications Applications Conference (SIU), İzmir-Türkiye, 1-4, 2-5 Mayıs, 2018.