# AYRIK DALGACIK DÖNÜŞÜMÜ TABANLI PARALEL GÖRÜNTÜ SIKIŞTIRMA SİSTEMİ TASARIMI

Yusuf Coşkun
131402103

**YÜKSEK LİSANS TEZİ**
Bilgisayar Mühendisliği Anabilim Dalı
Bilgisayar Mühendisliği Yüksek Lisans Programı
Danışman: Dr. Öğr. Üyesi Ali Akman

İstanbul
T.C. Maltepe Üniversitesi
Fen Bilimleri Enstitüsü
Ağustos, 2019

## JÜRİ VE ENSTİTÜ ONAYI

YUSUF COŞKUN'ın "Ayrık Dalgacık Dönüşümü Tabanlı Paralel Görüntü Sıkıştırma Sistemi Tasarımı" başlıklı tezi 18.09.2019 tarihinde aşağıdaki jüri tarafından değerlendirilerek "Maltepe Üniversitesi Lisansüstü Eğitim ve Öğretim Yönetmeliği" nin ilgili maddeleri uyarınca Bilgisayar Mühendisliği Anabilim Dalı Yüksek Lisans/Doktora tezi oy birliğiyle/oy çokluğuyla, başarılı/başarısız olarak kabul edilmiştir.

| | Unvanı. Adı ve Soyadı | İmza |
|---|---|---|
| Üye (Tez Danışmanı) | Dr. Öğr. Üyesi Ali AKMAN | ..... |
| Üye | Doç. Dr. Serap ÇEKLİ | ..... |
| Üye | Doç. Dr. Serkan TOPALOĞLU | ..... |

Prof. Dr. İlter BÜYÜKDIĞAN
Enstitü Müdürü V.

| | ŞEKİL ONAY SAYFASI | Doküman No | FR-105 |
|---|---|---|---|
| | | İlk Yayın Tarihi | 20.12.2017 |
| | | Revizyon Tarihi | 10.12.2018 |
| | | Revizyon No | 01 |
| | | Sayfa | 1/2 |

# ŞEKİL ONAY SAYFASI

02./10./2019

FEN BİLİMLERİ ENSTİTÜSÜ MÜDÜRLÜĞÜNE,

Aşağıda bilgileri bulunan lisansüstü öğrencinin tezi şekil yönünden tarafımca incelenmiş ve Enstitüye teslim edilmesi uygun bulunmuştur.

Anabilim Dalı Başkanı V.
Prof. Dr. A. Mesut Kayhan
Adı Soyadı

İmza

| ÖĞRENCİ BİLGİLERİ | |
|---|---|
| ADI SOYADI | Yusuf Coşkun |
| ÖĞRENCİ NUMARASI | 131402103 |
| ANABİLİM DALI | Bilgisayar Mühendisliği |
| PROGRAMI | ( X ) YÜKSEK LİSANS<br>( ) DOKTORA<br>( ) SANATTA YETERLİK |
| DANIŞMANI | Dr. Öğretim Üyesi Ali Akman |
| TEZ BAŞLIĞI | Ayrık Dalgacık Dönüşümü Tabanlı Paralel Görüntü Sıkıştırma Sistemi Tasarımı |
| SAVUNMA TARİHİ | 18/09/2019 |
| e-posta | yusufcoskun23@gmail.com |

| | |
|---|---|
| İç Kapak | ☑ Var ☐ Yok |
| Jüri Onay Sayfası | ☑ Var ☐ Yok |
| Etik İlke ve Kurallara Uyum Beyanı | ☑ Var ☐ Yok |
| İntihal Raporu | ☑ Var ☐ Yok |
| Teşekkür Sayfası | ☑ Var ☐ Yok |
| Öz (Başlık-Öz-Anahtar Sözcükler) | ☑ Var ☐ Yok |
| Abstract (Title-Abstract-Key Words) | ☑ Var ☐ Yok |
| İçindekiler | ☑ Var ☐ Yok |
| Çizelgeler Listesi | ☐ Var ☑ Yok |
| Şekiller Listesi (varsa) | ☐ Şekil yok ☑ Uygundur ☐ Uygun Değildir |
| Kısaltmalar Listesi | ☑ Var ☐ Yok |
| Tablolar Listesi (varsa) | ☐ Tablo yok ☑ Uygundur ☐ Uygun Değildir |
| Ekler Listesi (varsa) | ☑ Ek yok ☐ Uygundur ☐ Uygun Değildir |
| Özgeçmiş | ☑ Var ☐ Yok |
| Sayfa Genişliği | ☑ Uygundur ☐ Uygun Değildir |
| Yazı Tipi | ☑ Uygundur ☐ Uygun Değildir |

| Hazırlayan<br>İlgili Birim | Kalite Koordinatörü<br>Dr. Öğr. Üyesi Şafak GÜNDÜZ | Kurumsal Yetkili<br>Prof. Dr. Belma AKŞİT |
|---|---|---|

*(Doküman No: FR-105; Yayın Tarihi 20.12.2017; Revizyon Tarihi: ; Revizyon No:00)*

| | ŞEKİL ONAY SAYFASI | Doküman No | FR-105 |
|---|---|---|---|
| | | İlk Yayın Tarihi | 20.12.2017 |
| | | Revizyon Tarihi | 10.12.2018 |
| | | Revizyon No | 01 |
| | | Sayfa | 2/2 |

| | |
|---|---|
| Referans Kullanımı | ☑ Uygundur ☐ Uygun Değildir |
| Kaynakça Yazımı | ☑ Uygundur ☐ Uygun Değildir |
| Ekler (varsa) | ☑ Ek yok ☐ Uygundur ☐ Uygun Değildir |

Hazar Akgül
İmza

| Hazırlayan | Kalite Koordinatörü | Kurumsal Yetkili |
|---|---|---|
| İlgili Birim | Dr. Öğr. Üyesi Şafak GÜNDÜZ | Prof. Dr. Belma AKŞİT |

*(Doküman No: FR-105; Yayın Tarihi 20.12.2017; Revizyon Tarihi: ; Revizyon No:00)*

**Revizyon Takip Tablosu**

| REVİZYON NO | TARİH | AÇIKLAMA |
|---|---|---|
| 00 | 01.03.2018 | İlk yayın. |
| | | |

## ETİK İLKE VE KURALLARA UYUM BEYANI

18/09/2019

Bu tezin bana ait, özgün bir çalışma olduğunu; çalışmamın hazırlık, veri toplama, analiz ve bilgilerin sunumu olmak üzere tüm aşamalarından bilimsel etik ilke ve kurallara uygun davrandığımı; bu çalışma kapsamında elde edilmeyen tüm veri ve bilgiler için kaynak gösterdiğimi ve bu kaynaklara kaynakçada yer verdiğimi; çalışmanın Maltepe Üniversitesinde kullanılan "bilimsel intihal tespit programı" ile tarandığını ve öngörülen standartları karşıladığını beyan ederim.

Herhangi bir zamanda, çalışmamla ilgili yaptığım bu beyana aykırı bir durumun saptanması durumunda, ortaya çıkacak tüm ahlaki ve hukuki sonuçlara razı olduğumu bildiririm.

Yusuf Coşkun

| Hazırlayan | Kalite Koordinatörü | Kurumsal Yetkili |
|---|---|---|
| İlgili Birim | Dr. Öğr. Üyesi Şafak GÜNDÜZ | Prof. Dr. Belma AKŞİT |

Yüksek Lisans Tezi - Ayrık Dalgacık Dönüşümü Tabanlı Paralel Görüntü Sıkıştırma Sistemi Tasarımı

ORIJINALLIK RAPORU

| %8 | %5 | %1 | %7 |
|---|---|---|---|
| BENZERLIK ENDEKSI | İNTERNET KAYNAKLARI | YAYINLAR | ÖĞRENCI ÖDEVLERI |

BIRINCIL KAYNAKLAR

| | | |
|---|---|---|
| 1 | Submitted to The Scientific & Technological Research Council of Turkey (TUBITAK)<br>Öğrenci Ödevi | %4 |
| 2 | Submitted to Eskisehir Osmangazi University<br>Öğrenci Ödevi | %1 |
| 3 | www.hho.edu.tr<br>İnternet Kaynağı | %1 |
| 4 | bilgisayarkavramlari.sadievrenseker.com<br>İnternet Kaynağı | <%1 |
| 5 | Submitted to Anadolu University<br>Öğrenci Ödevi | <%1 |
| 6 | Submitted to Çankırı Karatekin University<br>Öğrenci Ödevi | <%1 |
| 7 | Submitted to Ondokuz Mayıs Universitesi<br>Öğrenci Ödevi | <%1 |
| 8 | thesaurus.babylon.com<br>İnternet Kaynağı | <%1 |

Dr. Öğr. Üy. Ali Akkaya

# TEŞEKKÜR

Bu tez çalışmasının seçiminde, yürütülmesinde, her sorumu itina ile cevaplayan ve sonuçlandırılmasında destek ve yardımlarını esirgemeyen danışman hocam Sayın Dr. Öğr. Ali Akman'a teşekkür ederim.

Yusuf Coşkun

Ağustos 2019

# ÖZ

## AYRIK DALGACIK DÖNÜŞÜMÜ TABANLI PARALEL GÖRÜNTÜ SIKIŞTIRMA SİSTEMİ TASARIMI

Yusuf Coşkun
Yüksek Lisans Tezi
Bilgisayar Mühendisliği Anabilim Dalı
Bilgisayar Mühendisliği Yüksek Lisans Programı
Danışman: Dr. Öğr. Üyesi Ali Akman
Maltepe Üniversitesi Fen Bilimleri Enstitüsü, 2019

Günümüzde dijital görüntülerin çözünürlükleri giderek artmakta ve bunun sonucunda depolama birimlerinde kapladıkları alanlar da büyümektedir. Dijital görüntülerin boyutlarının artması başka bir noktaya iletilmesini zorlaştırmaktır. Geçmişten günümüze kablosuz ağlar, internet gibi veri iletişim bantları görüntü teknolojisi kadar hızlı gelişemediğinde dijital görüntülerin sıkıştırılmasına ihtiyaç doğurmuştur.

Günümüzde teknolojinin ilerlemesi ile düşük güç tüketimine ve düşük işlemci birimine sahip birçok tek çekirdekli bilgisayarlar bulunmaktadır. Bu tarz ürünlerin tercih edilmesinin başlıca tercih sebepleri arasında düşük enerji tüketimi, taşınabilir olması ve mali açıdan uygun olması öne çıkmaktadır. Bu gibi sistemler, üzerinde görüntü işlemek için tercih edilmeye başlanmıştır. Ayrıca çözünürlüğün artması ile dijital görüntünün işlenmesi için daha yüksek işlem birimlerine ihtiyaç duyulmaktadır. İşlem birimlerin günümüzde ulaştığı hızlar bu problemi çözmekte zorlanmaktadır. Görüntü işlemlerini birden fazla işlem biriminde paralel bir şekilde işlemek bu probleme çözüm olabilir.

Bu tez çalışmasında görüntü sıkıştırma için kullanılan Haar ayrık dalgacık dönüşümü kullanılarak, dağıtık bellek sistemine uygun bir şekilde paralel çalışan bir algoritma ile gerçeklenmiştir. Sistem olarak RaspberryPi 3 tercih edilmiş ve görüntü sıkıştırmada kuantalama ve exp-golomb yöntemleri de kullanılmıştır.

**Anahtar Sözcükler:** Görüntü Sıkıştırma, Ayrık dalgacık dönüşümü, Paralel programlama, Dağıtık bellek mimarisi, Raspberry Pi, MPI, Kuantalama, Exp-golomb, Kodlama.

# ABSTRACT

## DESIGN OF PARALLEL IMAGE COMPRESSION SYSTEM BASED ON DISCRETE WAVELET TRANSFORM

Yusuf Coşkun
Master Thesis
Department of Computer Engineering
Computer Engineering Programme
Advisor: Assist. Prof. Ali Akman
Maltepe University Science and Engineering Graduate School, 2019

Nowadays, the resolution of digital images is increasing and as a result the areas they occupy in storage units are growing. Increasing the size of digital images makes it difficult to transmit to another point. From the past to the present, the need to compress digital images have emerged since data communication bands such as wireless networks and the Internet could not develop as fast as image technology.

Nowadays, with the advancement of technology, there are many single core computers with low power consumption and low processor unit. Low energy consumption, portability and financial feasibility are among the main reasons for choosing such products. Such systems have begun to be preferred for image processing. In addition, higher processing units are needed for digital image processing with increased resolution. The speed that the processing units reach today has difficulty to solve this problem. Processing images in parallel using multiple processing units may be the solution to this problem.

In this thesis, Haar discrete wavelet transform used for image compression is implemented with an algorithm that works in parallel with the distributed memory system. RaspberryPi 3 was preferred as the system and quantization and exp-golomb methods were used in image compression

**Keywords:** Image compression, Discrete wavelet transform, Parallel programing, Distributed memory architecture, Raspberry Pi, MPI, Quantization, Exp-golomb, Encoding.

# İÇİNDEKİLER

# TABLOLAR LİSTESİ

# ŞEKİLLER LİSTESİ

# KISALTMALAR

| | |
|---|---|
| WSN | : Kablosuz Sensör Ağı |
| WMSN | : Kablosuz Multimedya Sensör Ağı |
| SPIHT | : Set Partitioning in Hierarchical Trees |
| ADD | : Ayrık Dalgacık Dönüşümü |
| EZW | : Embedded Zerotrees of Wavelet |
| EPCOT | : Embedded Block Coding with Optimal Truncation |
| HEVC | : High Efficiency Video Coding |
| FPS | : Frame Per Second |
| ADPD | : Ayrık Dalgacık Paket Dönüşümü |
| FT | : Fourier Transform |
| FFT | : Fast Fourier Transform |

# ÖZGEÇMİŞ

**Yusuf Coşkun**

**Bilgisayar Mühendisliği Anabilim Dalı**

**Eğitim**

| *Derece* | *Yıl* | *Üniversite, Enstitü, Anabilim/Anasanat Dalı* |
|---|---|---|
| Y.Ls. | … | Maltepe Üniversitesi, Fen Bilimler Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı |
| Ls. | 2012 | İstanbul Ticaret Üniversitesi, Fen Edebiyat Fakültesi Matematik Bölümü |
| Lise | 2006 | Pendik Lisesi |

**İş/İstihdam**

| *Yıl* | *Görev* |
|---|---|
| 2015 - 2019 | Matematik Öğretmeni, Özel Final Lisesi |

**Kişisel Bilgiler**

| | |
|---|---|
| Doğum yeri ve yılı | : İstanbul, 1988 Cinsiyet: E |
| Yabancı diller | : İngilice |
| GSM / e-posta | : +90 (507) 355 33 23 / yusufcoskun23@gmail.com |

# BÖLÜM 1. GİRİŞ

Günümüzde dijital görüntü sistemlerinin bir çok alanda yoğun bir şekilde kullanılıyor. Bu alanların bir kaçından bahsedecek olursak askeri, endüstriyel, güvenlik, sağlık, eğlence ve bireysel alanlarda diyebiliriz.

Dijital görüntü teknolojileri her geçen gün daha yüksek çözünürlük sistemler geliştirmekte. Bugün itaberi ile 4k çözünürlüklerinde ürünler herkes tarafından kullanılmakta ve yakın bir zamanda 8k çözünürlüğündeki ürünler herkes tarafından kullanılmaya başlancak. Görüntü teknolojilerin bu kadar hızlı gelişmesi sonucunda ortaya bazı sorunları beraberinde getirmiştir.

Bunlardan birincisi verilerin boyut olarak büyümesine paralel olarak kablosuz iletişim teknolojilerin gelişememiştir. Kablosuz iletişim bantların bu kadar büyük verileri zamanın önemli olduğu durumlarda ne yazık ki yetersiz kalmaktadır. Zamanın önemli olduğu durumlardan kastımız video akışıdır. Video akışının kullanıldığı yerleri düşündüğümüzde görüntülü konuşma, canlı yayınlar, televizyonlar, insansız hava araçları, güvenlik kameraları, ev eğlence sistemleri gibi bir çok alan bulunmakta. Bu problemin çözümü için bir çok görüntü sıkıştırma ve çözümleme teknikleri geliştirilmiş ve yaygın birşekilde günümüzde kullanılmaktadır.

Bunlardan ikincisi ise büyüğen bu dijital görüntülerin işlenmesi için daha yüksek işlem birimlere ihtiyaç duyması. İşlem birimleri gelişmesi yıllara göre izlendiğinde günümüzde artık işlem birimleri hız konusunda üst limitlerine yaklaşmış durumunda. Bu soruna çözüm olarak tek işlem biriminde çalışan algoritmalar yerine çoklu işlem birimlerimde çalışan algoritmaların geliştirilmesi yane paralel algoritmalar ortaya çıkmıştır. Bu sayede daha düşük hızlara sahip birkaç işlem birimi ile görüntü sıkıştırma ve çözümleme işlemlerini yapabiliriz. Ayrıca günümüzde yarıiletken teknolojisinin ilerlemesi ile daha düşük enerji harcıyan çoklu işlem birimleri içeren tek karttlı bilgisayar ortaya çıkmıştır. Bu bilgisayarlardan yapılan işe göre bir veya birkaç adet kullanılarak istenilen görüntü sıkıştırma ve çözümleme işlemleri daha düşük maliyetli ve daha az enerji tüketen sistemler yapılmasına olanak sağlıyor. Bir diğer avantajı bu sistemlerim belli bir kutu içerisine koyularak taşınabilir olmasına olanak sağlıyor.

## 1.1 Literatür Araştırması

Kablosuz sensör ağları (WSN) ortamdaki bilgi ve içerikleri toplayan cihazlardır. Fakat zamanın değişmesi ile birlikte bu cihazların da değişmesi ve gelişmesine ihtiyaç duyularak ‘’kablosuz multimedya sensör ağı’’ (WMSN) geliştirilmiştir. WMSN’deki verilerin depolama alanındaki sorunlarını en aza indiren ve WMSN üzerinden işleme ve aktarım nedeniyle güç tüketimini azaltan en etkili çözüm görüntü sıkıştırmadır. Bu sıkıştırma yöntemi ise hiyerarşik ağaçlarda bölümlendirmeyi ayarlama (SPIHT) ile yapılır. Fakat SPIHT ’in kodlama maliyetinin yüksek olması kullanımını zorlaştırmaktadır. Bundan dolayı bu dezavantajların üstesinden gelmek için WMSN için geliştirilmiş listesiz SPIHT 'i bir hat ile birleştirerek etkili bir sıkıştırma şeması sunmaktadır [1].

Görüntü, insanların hayatındaki bilgi iletişiminde ve bilgi içeren medyada en önemli taşıyıcıdır. Birbiriyle son derece ilişkili olan piksellerden oluşur. Ancak bu ilişki nedeniyle büyük depolama alanı kaplar ve iletim bant genişliğini fazla harcar. Gözlenen üç çeşit veri fazlalığı vardır. İlki uzamsal fazlalıktır, komşu piksellerle ilişkilendirilir, bu nedenle görüntü boyutunu azaltmak için bir kare içindeki gereksiz tekrarlanan verilerin kaldırılması gerekir. İkincisi geçici fazlalıktır, görüntü gösterimi için gereken bit sayısının azalmasına duyulan ihtiyaçtır. Sonuncusu, Spektral fazlalıktır, farklı renk düzlemleri arasındaki ilişkidir. Bu fazlalıkların üstesinden gelmek için, orijinal görüntüye yeterince yakın olan yeniden oluşturulmuş görüntünün kabul edilebilir görsel kalitesini korurken minimum depolama ve düşük bant genişliği için sıkıştırma yapılmalıdır. Görüntü kalitesi, sıkıştırma oranı, sıkıştırmanın hızı ve güç tüketimi sıkıştırma performansı için kullanılan başlıca ölçümlerdir. Performans ölçütleri işletim platformuna göre farklılık gösterir. Kaynaklar kısıtlıdır diğer bir deyişle, kablosuz multimedya sensör ağlarının hafıza, güç tüketimi ve işlemci özellikleriyle ilgili bazı kısıtlamaları vardır. Bu nedenle SPIHT, kullanılabilecek en güçlü tekniktir [2].

Mevcut İnternet kullanıcılarının yetenekleri ve istekleri ile uyumlu esnek görüntü ve video sıkıştırma programları kullanıcılarını farklı bilgi işlem ve ağ erişim hızları motive etti. Bu görüntü sıkıştırma şemaları ölçeklenebilirdir ve iki ana ölçeklenebilirlik türü vardır: oran ve çözünürlük ölçeklenebilirliği. İki boyutlu ayrık dalgacık dönüşümü (2D-ADD), çoklu çözünürlük doğası nedeniyle ve zaman-frekans lokalizasyon özellikleri

nedeniyle ölçeklenebilir görüntü sıkıştırmasında önemli bir rol oynar. Oran ölçeklenebilirliği ise tüm ADD görüntü piksellerinin en önemli bitten en az önemli bit düzlemine kadar kodlaması ile başarılabilir [3].

Bir görüntünün hızlı iletimi ve depolanması için en önemli ihtiyaçlardan biri sıkıştırmadır. Günümüzde birçok görüntü sıkıştırma yöntemi vardır ancak bunlardan en kayda değer olanları EZW, SPIHT ve EBCOT algoritmalarıdır. SPIHT ve EZW algoritmaları karşılaştırıldığında ikisinin de benzer düzeyde performans gösterdiği görülür ancak SPIHT daha ucuz ve kolay çözümler sağlamaktadır. Algoritmaların önemlilik testinin uygulandığı alt kümelerin sıralama bilgisi üçe ayrılmaktadır: LIP, önemsiz katsayıların listesidir ve geçerli eşik değerine göre önemsiz katsayıların koordinatlarını içermektedir. LSP, önemli katsayıların listesidir ve geçerli eşik değerine göre önemli katsayıların koordinatlarını içermektedir. LIS, önemsiz kümelerin listesidir ve önemsiz alt ağaçların köklerinin koordinatlarını içermektedir [4].

Bant ihtiyacının ana sebebi, görüntülerin depolanması sırasında gereğinden fazla depolama alanı ve görüntülü iletişim amacıyla ağ kullanılmasıdır. Bu durumun nedeni ise algılama ve görüntüyü işlemedeki hız, kalitedeki artış ve dolaysı ile dosya boyutundaki artıştır. Son zamanlardaki hizmet ve uygulamalar dolayısı ile yeni ihtiyaçları giderebilmek için geliştirilen yüksek verimli video kodlama tekniği HEVC ’dir. Depolanan videoları sıkıştırmak için dizayn edilse de görüntü-içi kodlamalarda da kullanılmaktadır. Sonuç olarak HEVC test edilmiştir ve kodlama verimliliğinin artmasıyla birlikte görüntü boyutlarının azaldığı gösterilmiştir. [5]

Dalgacık dönüşümü, görüntü işleme, sıkıştırma, sinyal analizi ve tıbbi görüntüleme gibi birçok alanda önemli ölçüde katkıda bulunmuştur. Ayrık dalgacık dönüşümü (ADD), çok büyük bir bellek gereksinimi gerektirir ve özellikle kuadratik hesaplama karmaşıklığına sahip 2 boyutlu dönüşüm için hesaplama açısından yoğundur. Bu makalede, 2048 x 1536 görüntü boyutu için maksimum 138 FPS verim elde edebilen gerçek zamanlı görüntü ve video işleme için 2 boyutlu bir ADD sistem mimarisi önermektedir. Mimari ölçeklenebilir ve birden fazla ADD motorunu destekleyebilir bunlara ek olarak her ADD motoru bağımsız olarak çalışabilir. Yapı aynı zamanda düşük dahili tampon boyutu, düşük kontrol karmaşıklığı, hızlı hesaplama süresi ve düşük güç tüketimi avantajlarına sahiptir. ADD sistemi yeterince modüler ve esnektir ve dalgacık

dönüşümü gerektiren herhangi bir uygulama için gelecekte genel ADD işlemcisi olarak kullanılabilir. İhtiyaç duyulan tek değişiklik, gelecekte programlanabilir hale getirilebilecek olan hesaplamalı blok tasarımıdır. [6]

Ayrık dalgacık dönüşümü (ADD), dinamik alt-bant ayrışmasındaki sinyali temsil eder. Ayrık dalgacık paket dönüşümü (ADPD), frekans ekseninin uyarlamalı bir ayrışmasını gerçekleştirir. Dalgacık algoritmaları verileri farklı ölçeklerde veya çözünürlüklerde işler. Fourier Transformasyon (FT) ve hızlı algoritmalarıyla (FFT) birçok doğal sinyalin analizi ve işlenmesi için önemli bir araçtır. FT, durağan olmayan birçok doğal sinyalin karakterizasyonu konusunda belirli sınırlamalara sahiptir. ADD ise görüntü sıkıştırmak için giderek daha fazla kullanılıyor. Bunun nedeni, ADD 'nin kaliteye ve çözünürlüğe göre sıkıştırılmış görüntü kolaylığı ve ilgilenilen bölge kodlaması gibi özellikleri desteklemesidir [7].

Dalgacık Dönüşümü, durağan olmayan sinyaller üzerinde çalışırken önerilen çözümlerden biridir. Dalgacık Dönüşümü yaygın olarak sinyal analizi, gürültüyü gidermede ve sıkıştırmada kullanılır. Ayrık Dalgacık Dönüşümü (ADD) ilk olarak Mallat tarafından tanımlandı. Konumun hem zaman hem de frekans alanlarında ayarlanabilir olduğu çoklu-çözünürlük sinyal analizi yapar. Yüksek frekanslı bileşenlerin yerel maksimum değerleri arasındaki ilişki, sinyalin çok ölçekli dalgacık ayrıştırmasından çıkarılır. Düşük karmaşıklık ve düşük güç ile Ayrık Dalgacık Dönüşümü (ADD) tasarlamak için farklı teknikler tartışılmaktadır. Karmaşık olmayan tasarımı nedeniyle, kaldırma düzeni daha az güç tüketimi sağlar [8].

Görüntü sıkıştırmanın temel amacı, depolama alanı gereksinimini azaltmak, iletim hızını artırmak ve belirli uygulamalara bazı kısıtlamalar uygulayan yonga ek yükünü en aza indirmektir. Görüntü işlemede Lifting yönteminin asıl amacı, gerçek zamanlı uygulamalar için yüksek donanım ve yazılım yapıları elde etmektir. Görüntü sıkıştırma için Lifting tekniği uygulamak, hesaplamada tasarruf etmenizi sağlar [9] .

Güncel çalışmalar, medikal alanda tanısal görüntüleme kullanımının son yirmi yılda yüksek bir çıkış yakaladığını belirtmektedir. Bu gelişme, görüntüleme sistemine yardımcı olacak donanım ve yazılımların hızlı bir şekilde gelişmesinin bir sonucudur. Ayrıca, analogdan dijital sistemlere geçişin daha güvenilir, daha hızlı ve stressiz

görüntüleme sunması da bu gelişimle ilişkilidir. Görüntü toplama teknolojisi, X-Ray, Ultrason, MRI / fMRI, Nükleer Tıp, PET, CT ve DXA ile gerçekleştirilir. Ne yazık ki, bu görüntülerin piksel boyutlarının büyük olması depolama cihazlarının kapasitelerini ciddi şekilde zorlamaktadır. Bu görüntüleri sıkıştırmak için birçok teknik geliştirilmiştir. En çok tercih edilen ise hibrit görüntüleme tekniğidir. Hibrit görüntüleme tekniği sayısal ve fonksiyonel analizlerde, dalgacıkları ayrı ayrı örneklenen ve ayrık zaman sinyallerini işlemek için geliştirilen herhangi bir dalgacık dönüşümüdür [10]

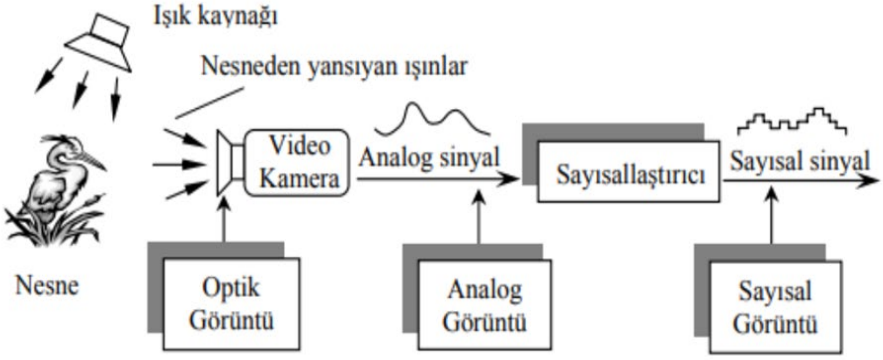
# BÖLÜM 2. YÖNTEM

Bu bölümde öncelikle görüntünün ne olduğu, görüntünün sıkıştırılması kullanılıcak Haar ayrık dalgacık dönüşümü, kodlama için kuantalama ve exp-golomb teknikleri incelenecek. Sonrasında görüntü sıkıştırmak için kullandığımız bu yöntemin paralelleştirdiğimizden paralel programlamın genel yapısı sınıflandırılması ve genel kullanılan komutlarını açıklayacağız.

## 2.1 Görüntü

Görüntü (image), çeşitli yollar kullanılarak elde edilen bilgilerin görüntüsel olarak saklanmasına ve tekrar gösterimine imkan sağlayan resimlere denir. Görüntü dediğimiz kavram üçe ayrılır Şekil 2.1'de görüldüğü gibi;

- Optik Görüntü
- Analog Görüntü
- Sayısal Görüntü




Şekil 2. 1 - Görüntünün yakalanması ve sayısal bilgiye çevrilmesi [26]

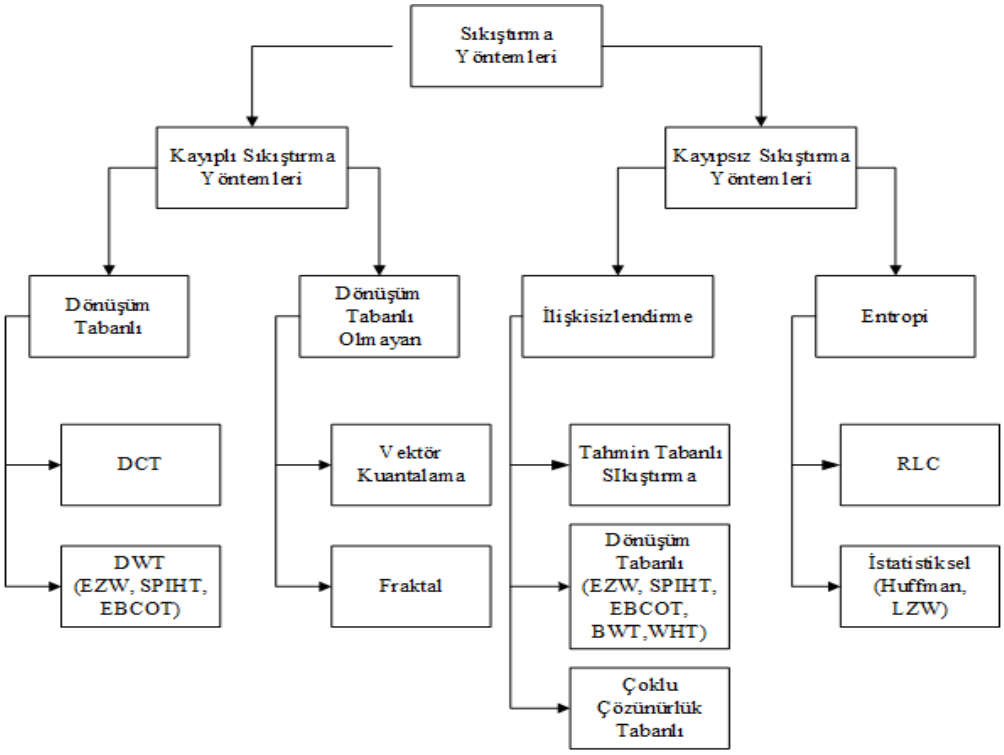
Sayısal görüntü gerçek hayattaki cisimlerin ışık kaynakalrından yansıttığı optik görüntülerin belirli bir anda yakalanıp iki değişkenkli bir fonksiyon f(x,y) olarak ifade edilmesidir. Elde edilen sayısal görüntü iki boyutlu bir sahne olarak düşünülürse x ve y sahnedeki her bir noktanın adresi olacaktır. Bir (x,y) noktasında bulunan her nokta piksel olarak adlandırılır. Sayısal görüntüler bilgisayar ortamında saklanır ve görüntülenirler. Her bir piksel sayısal değerlerden oluşur. Sayısal görüntüler, görüntünün sakladığı piksellerin değer aralıklarına göre üçe ayrılır;

- İkili görüntü (siyah-beyaz)
- Gri tonlamalı görüntü
- Renkli görüntü

İkili görüntüde sadece siyah ve beyaz renkleri ifade edilir, dolayısıyla her bir piksel 0 ya da 1 değerlerini içerir. Gri tonlamalı görüntüde örneğin her piksel 8 bit ile ifade edilecek olursa, her pikselin siyaha ya da beyaza yakınlığına göre 0 ile 255 sayıları arasında bir değer alır. Renkli görüntülerde ise bir gri tonlamalı görüntüyü ifade eden matris yapısından 3 katman vardır. Her bir katman kırmızı, yeşil ve mavi renklerinin o pikselde ne kadar yoğunlukta bulunduğu bilgisini yine 0 ile 255 arasında ifade ederler.

## 2.2 Görüntü Sıkıştırma

Görüntünün boyutu olarak ifade ettiğimiz, görüntünün bilgisayarın depolama alanında kapladığı yeri ifade eden miktarın azaltılması için kullanılan yöntemlere görüntü sıkıştırma denir. Görüntü sıkıştırma, kayıplı ve kayıpsız olmak üzere iki ana yöntem bulunmaktadır. Şekil 2.2'de görüleceği üzere bu iki ana yöntem altında literatürde bir çok sıkıştırma yöntemi vardır.




Şekil 2. 2 - Literatürdeki görüntü sıkıştırma yöntemlerinin sıkıştırılması [2]

Kayıpsız sıkıştırmada sıkıştırlan görüntü geri açılırken herhangi bir bilgi kaybı söz konusu değildir. Ancak kayıplı görüntü sıkıştırmada görüntünün ifade ettiği bilgilerin bir kısmı kaybolur. Bu da sıkıştırılan görüntü tekrar açıldığında ilk haline oranla belirli bozulmaların olacağı anlamına gelir. Bu noktada görüntünün ilk hali ile arasında ne kadarlık bir bozulma olduğunu ve bunun nasıl ölçülmesi gerektiği noktasındaki yöntemlerden iki tanesi şunlardır;

- Ortalama Karesel Hata (MSE, Mean Square Error)
- En Yüksek Sinyal Gürültü Oranı (PSNR Peak Signal To Noise Ratio)

### 2.2.1 MSE ve PSNR

Ortalama karesel hata görüntüdeki bozulmanın ölçümünde, sıkıştırmadan önceki pikselin değeri ile sıkıştırmadan sonraki pikselin değeri arasındaki farkın kareleri toplanıp toplam piksek sayısına bölünerek bulunur. Denklem 2.1'de gösterilmiştir.

$$MSE = \frac{1}{m \times n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \quad (2.1)$$

PSNR ise Denklem 2.2'de gösterilmiştir. [11]

$$PSNR = 10 \cdot \log_{10} \left( \frac{R^2}{MSE} \right) \quad (2.2)$$

Burada "R" pikselin alabileceği en yüksek değerdir. 8 bitlik bir piksel değeri için 255 olur.

### 2.2.2 Dalgacık Dönüşümü Tabanlı Görüntü Sıkıştırma

Görüntülerin yapısı gereği içerisinde ihtiyaç dışı ve gereksiz bilgi bulunur. Bu bilgilerin aktarma işlemi yaparken bize gereğinden fazla yük çıkarmaktadır. Görüntüyü sıkıştırma sırasında bu fazla olan bilgiler atılırken orijinal görüntüsünün yapısında bozulma çok düşük seviyede olmasına özen gösterilir. Bu fazlalık olan bilgileri kurtulmak için dalgacık tabanlı sıkıştırma işlemleri yapılabilir.

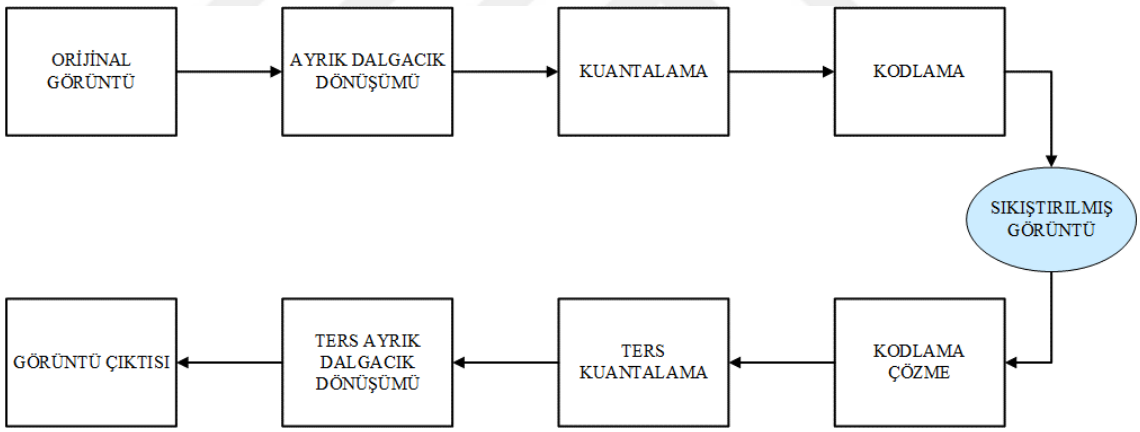
"Dönüşüm kodlamasında görüntü, bulunduğu boyuttan daha farklı bir boyuta aktarılır ve orada katsayılar kodlanır. Dönüşüm kodlaması sırasında görüntü elemanları arasında bulunan korelasyon azaltılmış olur. Korelasyon azaldığı zaman fazla bilginin

devamlı kodlanmasına gerek kalmaz. Dönüşüm kodlamasının getireceği diğer bir avantaj ise görüntü enerjisinin büyük bir bölümünü daha küçük bir alana toplamasıdır. Bu özelliğe dayanarak bütün katsayılar kullanılmadan kodlama yapılır" [12].

### 2.2.2.1 Ayrık Dalgacık Dönüşümü

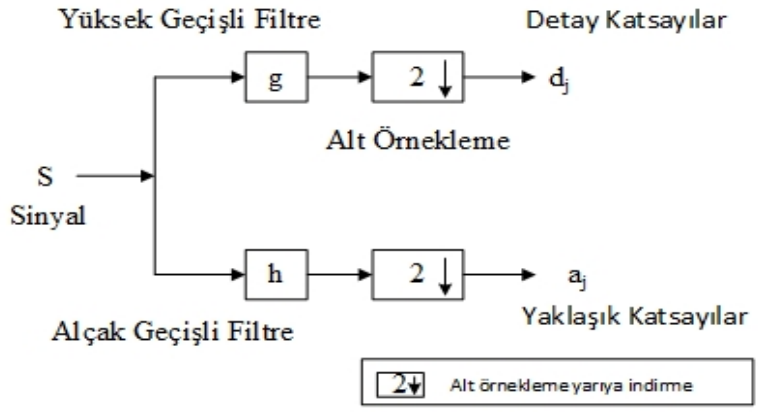
Bir görüntünün dalgacık tabanlı sıkıştırma işlemi yapılmak istenirse şekil 2.3'deki gibi bir yol izlenebilir.

"Dalgacık dönüşümü sırasında ana dalgacığın olası tüm ölçekleri ve sinyalin tüm zaman aralığı kullanılırsa hesaplanan katsayılar çok büyük veri yığınları oluşmasına sebep olur. Bu katsayıların tümü üzerinde işlem yapmak zorlaşır. Bu zorluğu aşmak için belirli ölçek ve konum aralıklarında dönüşüm yapılır. Ölçek ve konum değerlerinin 2'nin kuvvetleri olacak şekilde seçilmesi dönüşüm işlemini en hızlı şekilde olmasını sağlayacaktır. Bu işleme ayrık dalgacık dönüşümü denir" [13].



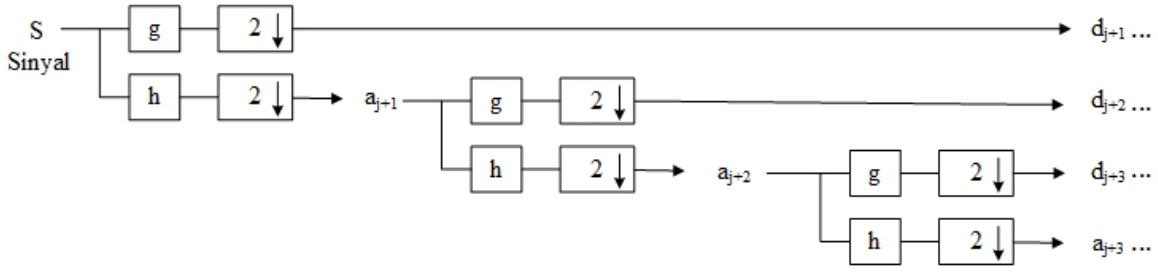

Şekil 2. 3 – Dalgaçık dönüşüm tabanlı görüntü sıkıştırma

ADD yüksek geçişli (g) ve alçak geçişli (h) filtreler eş zamanlı olarak sinyale (S) uygulanır. Yüksek geçişli filtrenin çıktısı, sinyalin detay katsayılarını ($d_j$); alçak geçişli filtrenin çıktısı, sinyalin yaklaşık katsayılarını ($a_j$) verir. Filtrelerden çıkan katsayıların örnek sayısı, alt örnekleme (downsampling) yapılarak yarıya düşürülür. Şekil 2.4 gösterilmiştir.




Şekil 2. 4 – Filtrelerle ayrık dalgacık dönüşümü

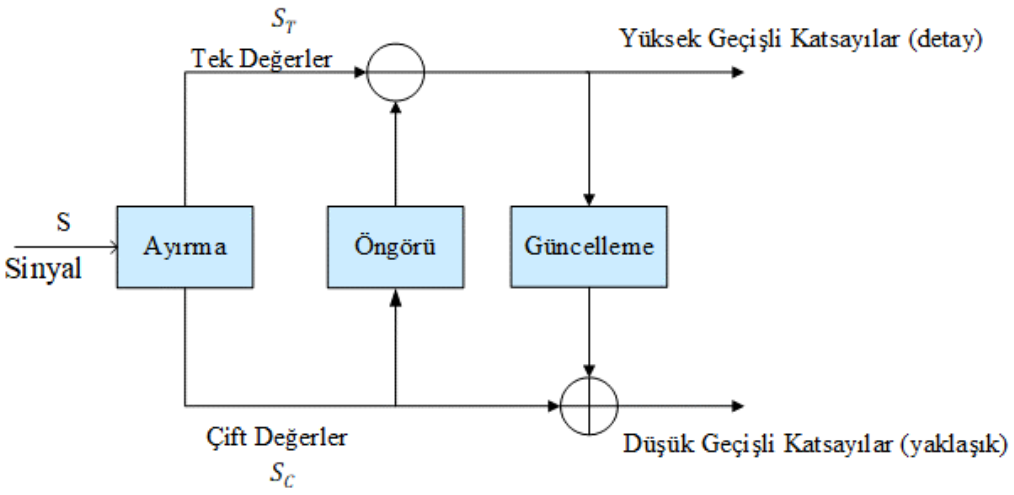
Ayrık dalgacık dönüşümünde birden fazla filtreleme ve alt örnekleme işlemi yapılabilir. Eğer birden fazla ayrık dalgacık dönüşümü uygulamak istersek, alçak geçiren filtrenin çıktısı olan yaklaşık katsayılarına filtreleme ve alt örnekleme tekrar tekrar uygulanır. Şekil 2.5 gösterilmiştir.




Şekil 2. 5 – Seviyeli ayrık dalgacık dönüşümü

### 2.2.2.2 Lifting Tabanlı Ayrık Dalgacık Dönüşümü

Klasik dalgacık dönüşüm işlemlerinde kullanılan filtreler karmaşık matematik işlemleri içerir bunu azaltmak için Lifting yöntemi geliştirilmiştir. Lifting yöntemi dalgacık dönüşümü için basit ve en verimli yöntemdir [14] .




Şekil 2. 6 – Bir boyutlu lifting dalgacık dönüşümü

Lifting yöntemi; ayırma, öngörü ve güncelleme olmak üzerek üç bölümden oluşmaktadır. Filtreler yerine öngörü ve güncelleme işlemleri uygulanır. Şekil 2.6'da bir boyutlu Lifting dalgacık dönüşümü görülmektedir.

Orijinal S sinyali öncelikle tek ve çift bileşenlerine ayrılır. Denklem 2.3 ve 2.4'de gösterilmiştir. Dalgacık katsayılarını Öngörü işlemi çift değerlere dayanarak tek değerlerin öngörü edilmeye çalışır. Bu işlemin sonucunda yüksek geçişli katsayılar (detay) elde edilir. Detay katsayısı *d* gösterilmiştir. Detay katsayısının denklemi 2.5'te gösterilmiştir.

$$S_C = S_{2k} \tag{2.3}$$

$$S_T = S_{2k+1} \tag{2.4}$$

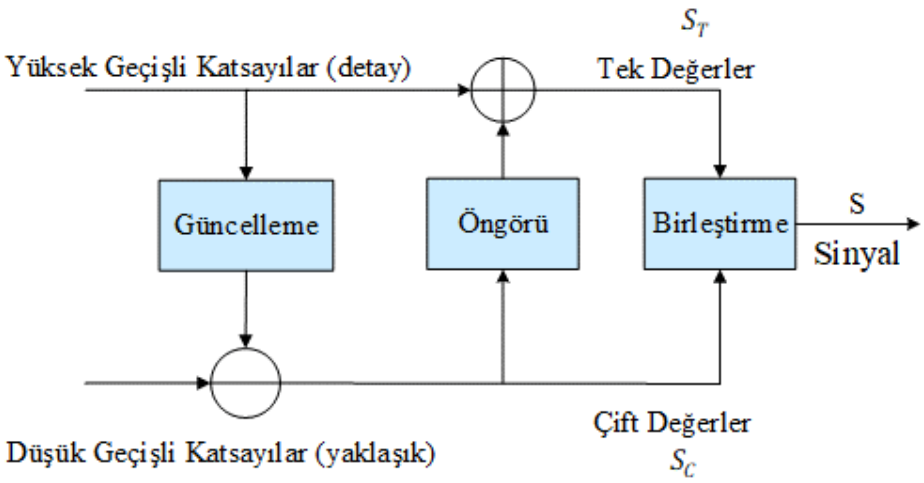
$$\mathrm{d} = S_T + \ddot{o}ng\ddot{o}r\ddot{u}(S_C) \tag{2.5}$$

Güncelleme adımında, verileri pürüzsüzleştirmek için ölçeklendirme yapılır ve ardından ADD için düşük geçişli filtrelenmiş değerleri sağlamak için çift değerleri eklenir. Bunun sonucunda sinyale ait düşük geçişli katsayılar (yaklaşık) verir. Yaklaşık katsayıları *y* gösterilmiştir. Yaklaşık katsayısının denklemi 2.6'te gösterilmiştir.

$$\mathrm{y} = S_c + \mathrm{güncelleme(d)} \tag{2.6}$$

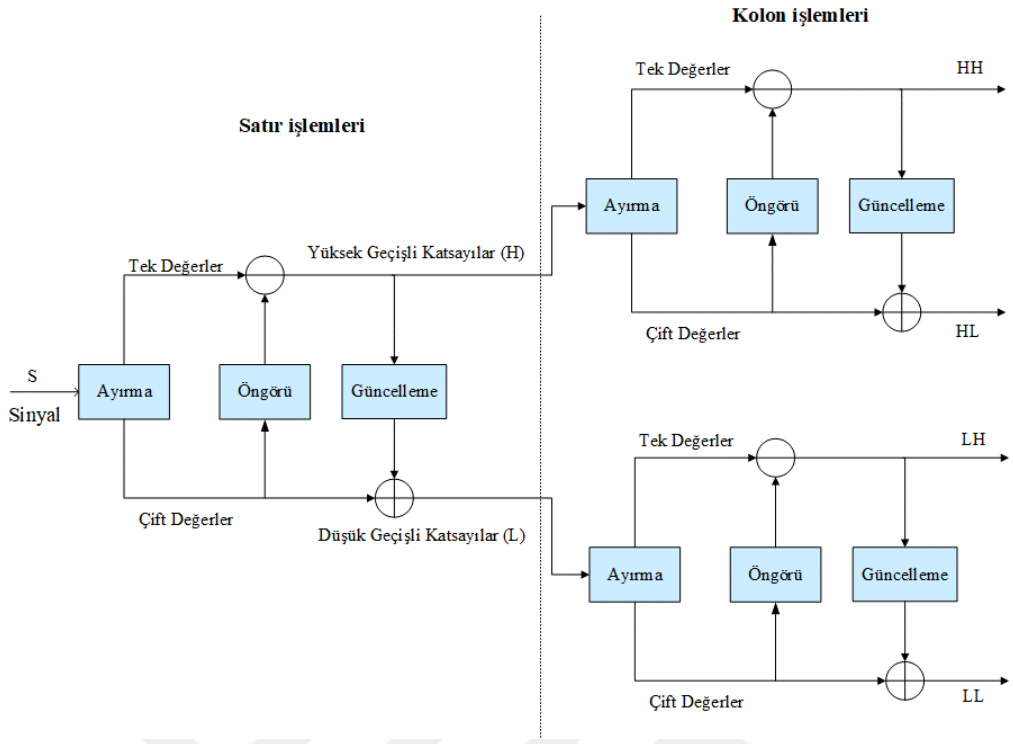
Elde edilen yüksek ve düşük geçişli katsayılara dönüşümde kullanılan işlemlerin tam tersi uygulanarak ters dönüşüm elde edilir. Bu şekilde lifting dönüşümün tersini Orijinal sinyalimizi tekrar elde etmiş oluruz. Şekil 2.7'de gösterilmiştir.

Bir görüntüye iki boyutlu lifting yöntemini gerçeklemek istersek ilk önce satırlara bir boyutlu lifting uygulanır. Bunun sonucunda yüksek ve düşük geçişli katsayılar elde




Şekil 2. 7 - Bir boyutlu ters lifting dalgacık dönüşümü

edildikten sonra kolonlara tekrar bir boyutlu lifting uygulanarak. Dört adet katsayı elde ederiz. Yüksek-yüksek geçişli katsayı (HH), yüksek-düşük geçişli katsayı (HL), düşük-yüksek geçişli katsayı(LH), düşük-düşük geçişli katsayı (LL) elde edilir. Şekil 2.8'de gösterilmiştir.




Şekil 2. 8 – Görüntüye 2-boyutlu lifting dalgacık dönüşümü

### 2.2.2.3 Lifting Yönteminde Haar Dalgaçık Dönüşümü

Haar dalgacık en temel dalgacık vektörüdür. Bir çok ortogonal vekörün oluşturmuş oldu bir dizidir. Bu vektörler +1 ile -1 sınırları aralığındaki değerleri alıp diğerlerinde sıfır değeri alır. Haar hafızada etkili, hızlı ve tam olarak geri dönüşümlüdür ama sınırlarda etkili değildir [15].

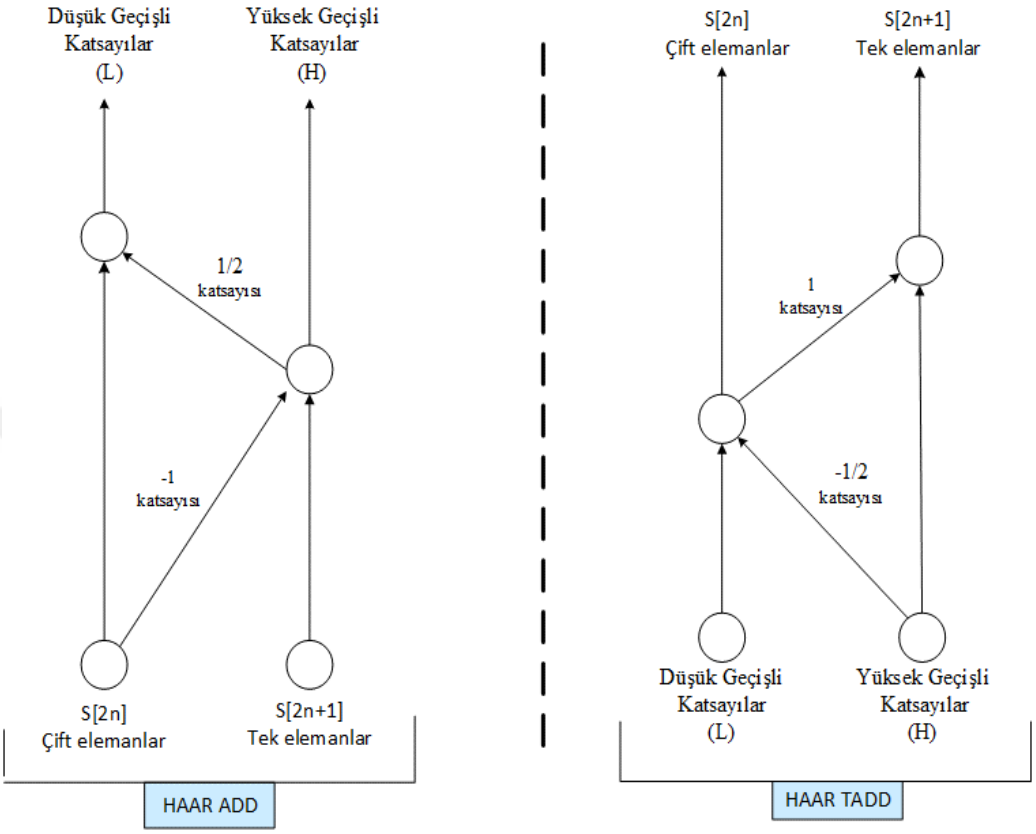
Haar dalgacık lifting yönteminde kullanmak istersek haar dalgacığın öngörü ve güncelleme adımlarında haar katsayıları kullanılır. Bu katsayılar; α için -1 ve β için 0.5 dir.

$$\mathrm{d} = S_T + \ddot{o}ng\ddot{o}r\ddot{u}(S_C) \tag{2.7}$$

Haar ADD alınırken detay katsayısı belirlemek için denklemde 2.7'deki öngörü yerine α katsayısı kullanılır. Haar ADD alınırken yaklaçık katsayısı belirlemek için denklemde 2.8'deki güncelleme yerine β katsayısı kullanılır.

$$\mathrm{y} = S_c + \mathrm{güncelleme(d)} \tag{2.8}$$

Ters dönüşümü elde etmek istersek şekil 2.9'daki haar idwt uygulayarak bunu yapabilir.




Şekil 2. 9 – Haar lifting dalgacık dönüşümü ve ters dönüşümü

## 2.3 Kuantalama

Kuantalama, görüntü verilerindeki sürekli değerler kümesinin, sonlu (tercihen küçük) bir değerler kümesi ile ifade edilmesi işlemime denir. Sinyal işlemede, bir sinyali sonsuz gösterimden, sınırlı miktarda bilgi içeren ayrık değerlere dönüştürmek için kuantalama kullanılır [16]. Bu işlem açıkça bir yakınsatma ve örnekleme fonksiyonudur. Dolayısıyla kuantalama işleminin kalitesini orijinal ve sürekli değerlerden oluşan bilginin, ne kadar az kayıpla ya da bozulmayla ayrık değerler şeklinde ifade edilmesi belirler. Sayısal ve vektörel olmak üzere iki tip kuantalama vardır. Sayısal kuantalamada, her bir giriş sinyaline ayrı ayrı işlem uygulanıp çıktı elde edilirken, vektörel kuantalamada belirli bir grup girdi sinyali kuantalama fonksiyonuna girerek bir çıktı oluştururlar. Bir grup halinde işlemek ve sonuç elde etmek, kuantalama fonksiyonunun verimliliğini

artırırken aynı zamanda hesaplama karmaşıklığını da artırır. Bizim bu çalışmada kullandığımız kuantalama fonksiyonu sayısaldır. Hesaplama maliyetinin düşük olması görüntü sıkıştırma alanında daha çok sayısal kuantalama yöntemlerini ön plana çıkarır.

Basite indirgemek adına görüntüleri gri tonlamalı olarak ele alırsak, bir görüntüyü iki boyutlu bir matis şeklinde ($x[n_1, n_2]$) ifade edebiliriz. Daha da basite indirgemek adına, n'in iki boyuttaki bütün indeksleri ifade ettiğini varsayarsak görüntüyü x[n] şeklinde ifade edebiliriz. Bu durumda bir görüntüye uygulanacak en basit yakınsama işlemi, her bir piksele ayrı ayrı uygulanan sayısal bir kuantalama fonksiyonu şeklinde tanımlanabilir. Kuantalamanın denklemi 2.9'da gösterilmiştir. [17]

$$q[n] = \langle \frac{x[n]}{\Delta} \rangle \tag{2.9}$$

Burada $\langle \quad \rangle$ bu işareti bir fonksiyon olarak göz önüne aldığımızda içindeki rasyonel değere en yakın tam sayı değerini veren bir fonksiyon olarak düşünülebilir. "Δ" ise kuantalama adım değeridir. Bu durumda q[n], x[n]'in kuantalanmış karşılığıdır. [17]

## 2.4 Exponential (Üstel) - Golomb Kodlaması

Üstel-Golomb kodları düzenli bir yapıya sahip değişken uzunlukta kodlardır. $k$'inci dereceden exp-golomb tek bir önek kodu ve yapılandırılabilir uzunlukta bir sonek ile oluşturulur.

Bir değerin ($v$) önek bitlerin sayısını ($n_o$) belirlemek için denklem 2.10'daki eşitsizlikten faydalanılır. Bulunan önek bitlerinin sayısının ($n_o$) 1 eksi kadar 0 bulunur ve sonuna 1 koyularak önek oluşturulur.

$$2^k(2^{n_o-1}) \leq v \leq 2^k(2^{n_o} - 1) \tag{2.10}$$

Sonekteki ($n_s$) bit sayısı bulmak için 2.11'deki denklemi kullanarak önek ($n_o$) kodunun değeri ile belirlenir.

$$n_s = k + n_o - 1 \tag{2.11}$$

Sonekin değerini $(v_s)$ bulmak için 2.12'deki denklemden faydalanılır. Bulunan değer ikili (binary) koda çevrilir. Burada dikkat etmemiz gereken $n_s$ uzunluğunda olmalıdır.

$$v_s = v - 2^k(2^{n_o-1} - 1) \tag{2.12}$$

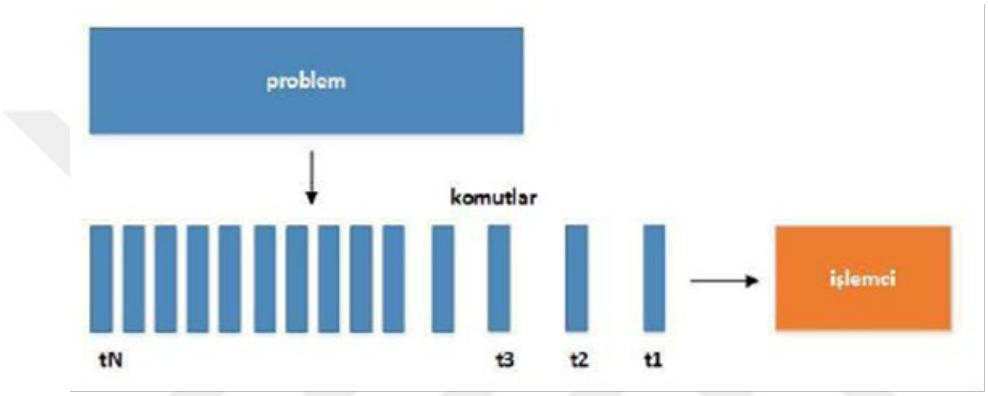
| $v$ | Exp-golomb Kodu |
|---|---|
| 0 | 1 |
| 1 | 010 |
| 2 | 011 |
| 3 | 00100 |
| 4 | 00101 |
| 5 | 00110 |
| 6 | 00111 |
| 7 | 0001000 |
| 8 | 0001001 |
| … | … |

Tablo 2. 1 - k=0 için Exp-Golomb kodlama tablosu

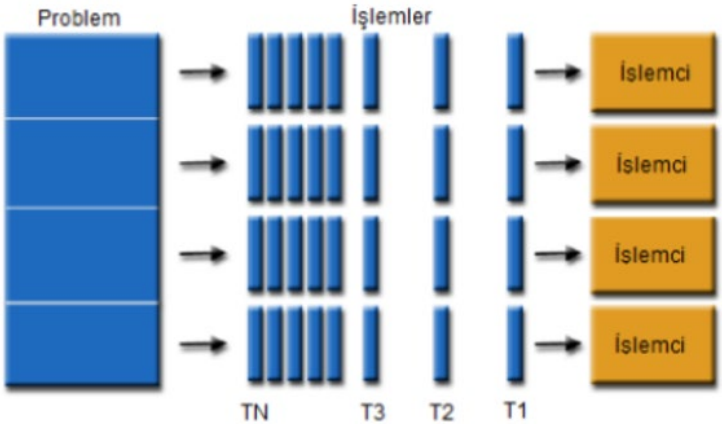
## 2.5 Paralel Programlama

Algoritma tekrarlayan bir problemin çözümünde kullanılan belirli ve tek bir anlamı olan komutlar serisidir. Algoritmalar bu doğaları gereği problemin çözümüne uygun belirli komutların belirli bir sırada uygulanmasını gerektirir. İşlemci gücünün giderek ucuzladığı günümüz dünyasında yarı-iletkenlere bağımlı işlemci tasarımları hem malzeme olarak hem de belirli bir alana sığdırılan transistör sayısı bakımından doğal sınırlarına yaklaşmış ve dolayısıyla hız konusunda limitlere dayanmıştır. Bu sebeple hem donanımsal olarak hem de yazılımsal olarak hız artırmak bilgisayar bilimi literatüründe önemli bir yer teşkil etmiştir. Algoritmaların doğalarındaki belirli bir sırada çalışmaya bağlı zorunluluk her zaman hız artırma çabalarının odaklandığı temel problem olmuştur. Çünkü algoritma için gerekli ilk komutlar çalıştırılırken bu ilk baştaki komutlara bağımlı olmayan ve kendi başlarına çalışıp problemin çözümü için gerekli olan belirli aşamaları hazır hale getirebilecek sonraki komutları çalıştırabilmek, hız kazanmak için her zaman odaklanılan ilk noktalardan biri olmuştur. Bu bağlamda birbirine bağımsız komutların paralel çalıştırılarak çok yollu işlemcilerin tasarlanması donanımsal yönden bu çözümün en bariz örneklerinden biridir. Yazılımsal anlamda ise buna benzer bir çözüm yazılım

katmanında birbirinden bağımsız kod bloklarını farklı işlemci birimlerine göndererek paralel çalışmalarını sağlayıp hız kazanmaktır. Literatürde geleneksel olarak programın algoritmanın belirlediği sırada tek tek çalışmasına seri programlama denirken yazılımsal anlamda yapılan bu paralelleştirmeye paralel programlanma denir. Şekil 2.3'de görüldüğü üzere seri programlamada komutlar tek bir işlemci birimine sırayla giderken, şekil 2.4'de görüldüğü üzere komutlar farklı işlemci birimlerine gönderilerek bağımsız blokların aynı anda çalışması ve kısmi sonuçlar üretmesi sağlanır. [18]




Şekil 2. 10 – Seri programlama çalışma mantığı




Şekil 2. 11 - Paralel programlama Çalışma Mantığı

### 2.5.1 Farklı Seviyelerdeki Paralellik Kavramları

Yukarıda bahsettiğimiz gibi donanım ve yazılım seviyesinde olmak üzere farklı seviyelerde paralellik kavramları vardır [19]. Bunları şu şekilde gruplandırabiliriz:

- Bit seviyesinde paralellik
- Komut seviyesinde paralellik
- Veri seviyesinde paralellik
- Görev paylaşımlı paralellik

Bit seviyesindeki paralellik bir saat darbesi süresince işlemci içinde aynı anda işlenebilecek veri miktarını belirtir. Komut seviyesindeki paralellik ise aynı anda işlemci içinde çalıştırılabilecek komut sayısını belirtir. Veri seviyesindeki paralellik üzerinde çalışılan verinin aynı anda kaç farklı işlemci üzerinde işlendiğini belirtir. Son olarak görev paylaşımlı paralellik ise farklı işlemci ya da çekirdeklerde aynı ya da farklı verilerin işlenmesini belirtir.

### 2.5.2 Fylinn Sınıflandırması

Bilgisayarlar mimari yapılarına göre aynı anda erişilebilir durumda olan komut sayısı ve bilgi akışına göre dört ana gruba ayrılır.

- Tek komut tek veri akışı (SISD)
- Tek komut çok veri akışı (SIMD)
- Çok komut tek veri akışı (MISD)
- Çok komut çok veri akışı (MIMD)

#### 2.5.2.1 Tek Komut Tek Veri Akışı (SISD)

Tek bir işlemci, tek bir control birimi ve tek bir veri akış yolu vardır. Sadece seri algoritmaların çalışmasına izin verir. Daha çok Von Neumann mimarisi olarak da bilinir.

#### 2.5.2.2 Tek Komut Çok Veri Akışı (SIMD)

Bu mimari yapıda N faklı SISD yapılı işlemci kullanılır ve her bir işlemcinin kendine ait bir yerel belleği vardır. Merkezi kontrol birimi tarafından kontrol edilen kontrol edilen tek bir komut akışı söz konusudur. İşlemciler her bir komutu farklı veriler

ile senkron bir şekilde çalıştırır. Veri alışverişi ve sonuç paylaşımı amacıyla işlemciler birbirleri arasında haberleşir. Bu haberleşme mekanizması paylaşımlı bir bellek üzerinden yapılır.

#### 2.5.2.3 Çok Komut Tek Veri Akışı (MISD)

N adet işlemci kendilerine ait kontrol üniteleri yardımıyla birbirinden bağımsız bir şekilde çalışır. Her işlemci birimi gelen komutu kendi çalıştırır ve sonucu işlem sırasında bir sonra gelen işlemci birimine iletir.

#### 2.5.2.4 Çok Komut Çok Veri (MIMD)

Paralel hesaplamada en genel ve en güçlü sınıftır. N adet işlemci N adet kontrol birimiyle gelen komutları birbirinden bağımlı ya da bağımsız çalıştırabilirler. Paylaşımlı bir bellek yapısı kullanılır.

### 2.5.3 Paralel Programlama Modelleri

Paylaşımlı bellekli, dağıtık bellekli ve veri paralelli modeleleri inceleyeceğiz. Ortak bellekli model ikiye ayrılır; İş parçacıklı (Thread) ve İş parçacıksız (Without Thread).

#### 2.5.3.1 Paylaşımlı Bellek Modeli

**İş Parçacıklı (Thread)**

Bu modelde bir adet ana işlem (main) vardır. Bu işlemin görevi olan uzun bir algoritma daha küçük algoritmalara bölünüp ana işlemin oluşturduğu, işletim sistemi tarafından programlanabilir iş parçacıklarına dağıtılır.

Genelde bu modelde ana işlem seri olarak bazı yerel ve küresel (global) değişkenleri ilişkilendirip ihtiyacı olduğu kadar iş parçacığını oluşturur. Oluşturulan iş parçacıkları işletim sisteminin programlama algoritmasına bağlı olarak ana işlem gibi programlanmaya başlar. İş parçacıkları yerel bazı kaynaklara sahip olduğu gibi aynı zamanda ana işlemin sahip olduğu tüm kaynaklara da ulaşabilir durumdadır. İşlemler ile iş parçacıkları arasındaki temel fark iş parçacıklarının kendilerine ait kaynakları yoktur

ve ana işlemin kaynaklarına doğrudan erişebilir durumdadır. Bu durum ucuz bir maliyetle işletim sistemi tarafından programlanabilir yapılar oluşturarak algoritmanın paralelleştirilmesi sağlanır. Küresel bellek alanlarındaki aynı adrese aynı anda okuma yazma işlemleri için işlem içi kilit yapıları kullanılır.

**İş Parçacıksız (Without Thread)**

Bu model de işlemler arası üzerinde anlaşılmış bir bellek alanı açılır ve her işlem bu bellek üzerinde birbiriyle paralel bir şekilde çalışır. Aynı bellek adreslerinde işlem yapabilmek adına işletim sistemi çapında kilit yapıları kullanılır.

#### 2.5.3.2 Dağıtık Bellek Modeli

Bu modelde en çok kullanılan yöntem mesaj geçirme yöntemidir. Bütün görevler çalışma sırasında yerel bellekler kullanırlar. Bu modelin en büyük avantajlarından biri de mesaj geçirilen görevlerin aynı anda aynı bilgisayarda çalışabilmesi yanında farklı bilgisayarlarda da çalışabilirler.

#### 2.5.3.3 Veri Paralel Modeli

Veri paralel modeli en basit algoritma modellerinden biridir. Bu modelde, görevler işlemlere (process) statik veya yarı statik olarak eşlenir ve her görev farklı veriler üzerinde benzer işlemleri gerçekleştirir. Aynı işlemlerin farklı veri kalemlerine aynı anda uygulanmalarının bir sonucu olan bu paralellik türüne veri paralelliği denir. İş fazlar halinde yapılabilir ve farklı fazlarda işlenen veriler farklı olabilir. Tipik olarak veri paralel hesaplama aşamaları, görevleri senkronize etmek veya görevlere yeni veriler almak için etkileşimlerle serpiştirilir. Tüm görevler benzer hesaplamalar yaptığından, sorunun görevlere ayrıştırılması genellikle veri bölümlemesine dayanır, çünkü yükün dengesini sağlamak için sabit bir haritalamanın ardından homojen bir veri bölümlemesi yeterlidir. Veri paralel algoritmalar hem paylaşılan adres alanı hem de mesajlaşma paradigmalarında uygulanabilir [20].

### 2.5.4 MPI Kütüphanesi ve Temel Fonksiyonları

MPI bir haberleşme kütüphanesidir. Temel amacı farklı işlemci birimlerine mesaj geçirme ve onlardan mesaj alma işlemlerinin kolaylaştırılmasıdır. Paralel programlama

modellerinden dağıtık bellek modelinin mesaj geçirme yöntemi MPI kütüphanesi yardımıyla kolayca gerçekleşebilir. Bu alt bölümde MPI kütüphanesi ve temel fonksiyonları tanıtılacaktır.

#### 2.5.4.1 Ana Fonksiyonlar

En temel 4 ana fonksiyondan bahsedebiliriz. Bunlar;

- *MPI_init*: İşlemin temel mesaj alışverişi sağlayabilmesi için gerekli olan ilklendirmelerin yapıldığı ve kütüphaneyi kullanmak isteyen programın çağırması gereken ilk fonksiyondur.
- *MPI_Comm_Rank*: Her bir işlemciye sayısal bir sıra vermek için kullanılır. Genelde sıfırdan başlar ve sıfır genelde algoritmanın başladığı ana makinedir.
- *MPI_Comm_Size*: Kümedeki toplam işlem biriminin sayısını belirlemek için kullanılır.
- *MPI_Finalize*: Ana işlemin artık mesaj alışverişine ihtiyacının kalmadığı bir anda çağrılarak daha önce ilklendirilmiş olan kütüphane verilerinin iade edilmesi ve tüketilen kaynakların bırakılması gibi işlemlerinin yapıldığı sonlandırma fonksiyonudur.

#### 2.5.4.2 Noktadan Noktaya Mesaj Alışverişi

- *MPI_Send*: Send fonksiyonunda veri bir işlemci biriminden diğerine veri gönderilir. Bloklu ve bloksuz olmaz üzere çeşitli varyasyonları vardır. Bloklu göndermede gönderme işlemi bitmeden o işlemci birimi çalışmaya devam etmezken bloksuz veride bitip bitmemesi önemsiz bir şekilde veri gönderilir.
- *MPI_Recv*: Recv fonksiyonunda bir işlemci biriminden veri alınır. Yine send fonksiyonlarındaki gibi bloklu ve bloksuz çeşitleri vardır.

#### 2.5.4.3 Noktadan Noktalara Mesaj Alışverişi

Bu altbölümde noktadan noktalara iletişim için kullanılan temel fonksiyonlar incelenecektir. Noktadan noktaya olan fonksiyonlar yardımıyla karşılaşılan birçok problem çözülebilir olsa da noktadan noktalara mesaj alış verişi yapabilen fonksiyonlar yazılımcının işini büyük oranda kolaylaştırmaktadır. Noktalardan kasıt burada bütün işlemci birimleri olabileceği gibi belirli bir grup işlemci birimleri de olabilir.

- *MPI_Barrier*: Bu fonksiyon çoklu iş parçacıklı programlamada temel olarak kullanılan ve çalışan bütün işlem veya işlem barçacıklarının tek bir belirli noktada senkron olmasını sağlayan bariyer (barrier) yapısının MPI kütüphanesi tarafından gerçeklenmiş halidir. Örneğin bütün işlemci birimlerinin belirli bir V verisini aldıktan sonra aynı anda bu veriyi işlemeye başlamaları isteniyorsa V verisi *recv* fonksiyonu ile alındıktan sonra bu fonksyion çağrılmalıdır.
- *MPI_Bcast*: Bu fonksiyon seçilen ana işlemciden diğer bütün işlemcilere mesaj gönderilmesini sağlar.
- *MPI_Scatter*: Scatter parçalanabilir bir veriyi diğer bütün işlemcilere eşit parçalara bölüp gönderir.
- *MPI_Gather*: Diğer bütün işlemci birimlerinden veri toplamak üzere kullanılır. Scatter fonksiyonunun tersi olarak düşünülebilir.
- *MPI_Allgather*: Gather fonksiyonu tek bir işlemci birimi üzerinde çalışıp belirtilen veriyi o işlemciye toplarken allgather fonksiyonu her bir işlemci biriminin gather çağırmasıdır.
- *MPI_Reduce*: Reduce fonksiyonu gather işleminin ardından gelen veriye ortak bir operasyon uygulanmasını sağlar.
- *MPI_Allreduce*: Reduce fonksiyonun tek bir işlemci birimi yerine verilen işlemci grubunda çalıştırılmasına olanak sağlar.

# BÖLÜM 3. PARALEL SİSTEMİN KURULUMU VE PARALEL GÖRÜNTÜ SIKIŞTIRMA ALGORİTMANIN GERÇEKLEŞTİRİLMESİ

Günümüzde görüntü kayıt cihazları çok yaygın bir şekilde kullanılmaktadır. Bunlar başlıca cep telefonları, tabletler, bilgisayarlar, fotoğraf makineleri vb. söylenebilir.

Günümüzde dijital görüntü kayıt cihazların gün geçtikçe çözünürlük ve saniye başı kare sayısı (fps) artmaktadır. Bunların artması ile dijital görüntülerin kapladığı alanlar hızlı bir şekilde artmaktadır. Bugün itibari ile elimizdeki cep telefonlar artık 4k (4096 x 2160 piksel) çekim neredeyse bir standart haline gelmiş bulunmaktadır. 4k'lık sıkıştırılmamış bir fotoğraf yaklaşık 25,31Mb yer kaplamakta veya 4k çözünürlüğünde ve 30fps'lik bir saniyelik sıkıştırılmamış bir video ise yaklaşık 759Mb yer kaplamaktadır.

Yakın bir zamanda ise 8k çözünürlüğünde teknoloji kullanan ürünler çıkmış bulunacak ve bu da bize dijital görüntülerin boyutları daha fazla yer kaplayacağını göstermektedir.

Kablosuz olarak gerçek zamanlı görüntü aktarmak istediğimizde bu kadar büyük dijital görüntüler için iletişim bantları genel olarak yetersiz kalmaktadır.

Sıkıştırılmamış dijital görüntülerin yüksek boyutları olması bizim için hem fazla alan kaplaması hem de kablosuz olarak gerçek zamanlı aktarmakta istediğimiz akışı sağlamakta sorun olmaktadır. Bunun bir çözümü olarak dijital görüntüyü sıkıştırma işlemi yaparak boyutlarını düşürmektedir.

Bizim gerçeklememizde haar ayrık dalgacık dönüşümünü lifting yöntemi kullanarak kayıplı görüntü sıkıştırma tercih edilmiştir.

Görüntü sıkıştırma işlemi ham görüntünün üzerine yönteme göre değişen matematiksel işlem yapmak demektir ve buda belli bir zaman almaktadır. Bu zamanın süresini belirleyen etkenler görüntünün büyüklüğü, kullanılan sıkıştırma yöntemi ve işlem birimin hızına göre değişmektedir.

Gerçek zamanlı görüntü aktarımında zaman önemlidir. Çünkü belli bir zaman içinde belli adet görüntü karesini göndermesi gerekmektedir. Sıkıştırma ve çözümleme işleminin bu süre içinde olması gerekmektedir. Bunun için bazen tek işlem birimi yetmemektedir. Buna bir çözüm olarak paralelleştirme işlemi yapılmaktadır.

Algoritmamızı paralelleştirme için bir cluster yapısı kuruldu. Cluster yapısı RaspberryPi 3 tercih edilmiştir. RaspberryPi'ların küçük, taşınabilir, fiyat performans bakımından uygun ve kapalı bir kutu içinde sistem olarak kurulmaya elverişlidir.

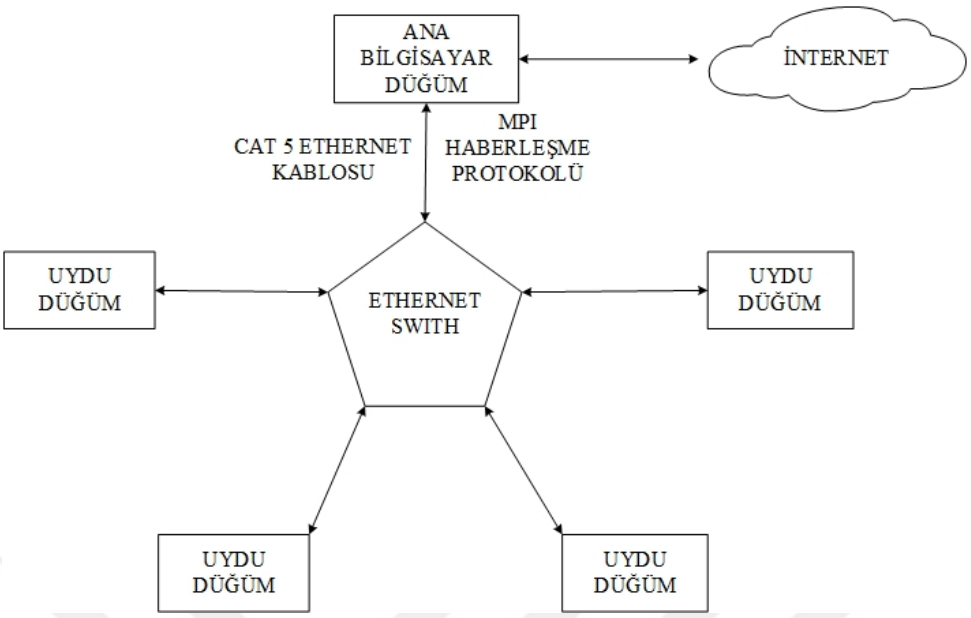
Algoritmayı python dili ile yazılmıştır. Python dili nesne tabanlı olması, kullanım dilinin kolaylığı, kütüphane zenginli, paralel programlama için hazır kütüphanesinin bulunma gibi sebeplerden tercih edilmiştir.

Tasarladığımız sistem dijital görüntünün sıkıştırma ve çözümleme algoritmasını paralelleştirilmesini sağlayan temel bir çalışmadır. Ayrıca kapalı bir sistem olarak taşınabilir hale getirmiştir.

## 3.1 RaspberryPi 3 İle Cluster Kurulumu

Bu bölümde paralel programlama algoritmamızı gerçeklemek için gerekli olan cluster (küme) yapısı kurulacaktır ve cluster kavramı hakkında genel bilgi verilecektir.

Cluster bilgisayar bilimlerinde, daha fazla işlem gücü elde etmek amacıyla birden fazla bilgisayarın tek bir bilgisayar gibi çalışmasına verilen isimdir. Genelde birden fazla bilgisayar birbirine oldukça hızlı bir ağ bağlantısı ile bağlanır ve bilgisayarların üzerinde çalıştırılan özel yazılımlar ile istenen işin paylaştırılması hedeflenir. Şekil 3.1'de dağıtık bellek yapılı cluster modeli gösterilmektedir.




Şekil 3. 1 - Dağıtık sistemli çoklu bilgisayar modeli

Kurcağımız cluster yapısı şekil 3.1'deki modele benzer yapıda olacaktır. Şekil 3.1'deki düğümlerin her birinde 1 adet RaspberryPi 3 olacaktır.

### 3.1.1 Donanım

Cluster sistemini kurmak için elimizde bulunan ekipmanlar aşağıdadır. Bu sistem için daha az veya daha fazla RaspberryPi kullanılabilir.

- 5 adet RaspberryPi 3 Model B
- 5 adet Micro Sd Kart
- 5 adet Cat5 Ethernet Kablo
- 5 adet RaspberryPi 3 Enerji kablosu usb-b
- 1 adet Zyxel 8 port 10/100Mbps Ethernet Anahtarı
- 1 adet Anker Powerport 6 60W (6 usb bağlantısı ve her bir port için maksimum 2.4A)

RaspberryPi'lar için tasarladığım özel plakalara elimizde bulunan 5 adet RaspberryPi 3 takılır. Her bir RaspberryPi güç vermek için hepsini USB arayüzlü güç adaptörüne bağlanır. RaspberryPi'ların 1 adet 10/100Mbps Ethernet bağlantısına sahiptir. Tüm RaspberryPi'lar Ethernet anahtarına bağlanır.

RaspberryPi 3 işlemci (CPU) içinde 4 adet işlem birimi (core) bulunmaktadır. Kurduğumuz sistemde 5 adet RaspberryPi bulunması ile elimizde toplam 20 adet işlem birimi (core) bulunmaktadır. Bu sayede yazdığımız algoritmayı 20 işlem (process) oluşturduğumuzda her bir işlem (process) ayrı bir işlem biriminde (core) yürütülür.

### 3.1.2 Yazılım

Cluster bilgisayarların (düğümler) birbiri ile haberleşmesi için MPI (Message Passing Interface) kullanabilir. MPI bir bilgisayar iletişim protokolüdür. Dağıtık bellekli bir sistemde düğümlerin arasındaki iletişim için kullanılabilen standart bir protokoldür. MPI kullanımını destekleyen birçok dil ve farklı paketler bulunmaktadır. Bu tezde dil olarak python - 3.5.3 sürümü ve paketler olarak MPICH2 – 3.3 ve MPI4PY – 3.0.0 kullanılmıştır.

### 3.1.3 Sistemin Kurulumu

Her bir RaspberryPi içinde Raspbian işletim sistemi yüklüyoruz. Raspbian debian tabanlı bir Linux dağıtımıdır ve RaspberryPi'ın orijinal işletim sistemidir.

Sistemimizdeki her bir RaspberryPi, düğüm olarak adlandıra biliriz. Sistemimizde bir ana düğüm ve dört uydu düğüm bulunmaktadır. Her düğüme sırası ile MPICH2 – 3.3, python – 3.5.3 ve MPI4PY – 3.0.0 kururuz. MPICH2 bir MPI paketidir. MPICH2 kurduğumuzdan artık her bir düğüm kendi içinde MPI haberleşme protokolünü kullanabilir. Programlama dili olarak python tercih edilmiştir. Python'da MPI protokollerini kullanmak için MPI4PY paketi gereklidir. Algoritmamız için gerekli olan iki kütüphane olan numpy ve pillow her bir düğüme kuruyoruz. Buraya kadar gerekli olan tüm yazılımlar yüklenmiştir.

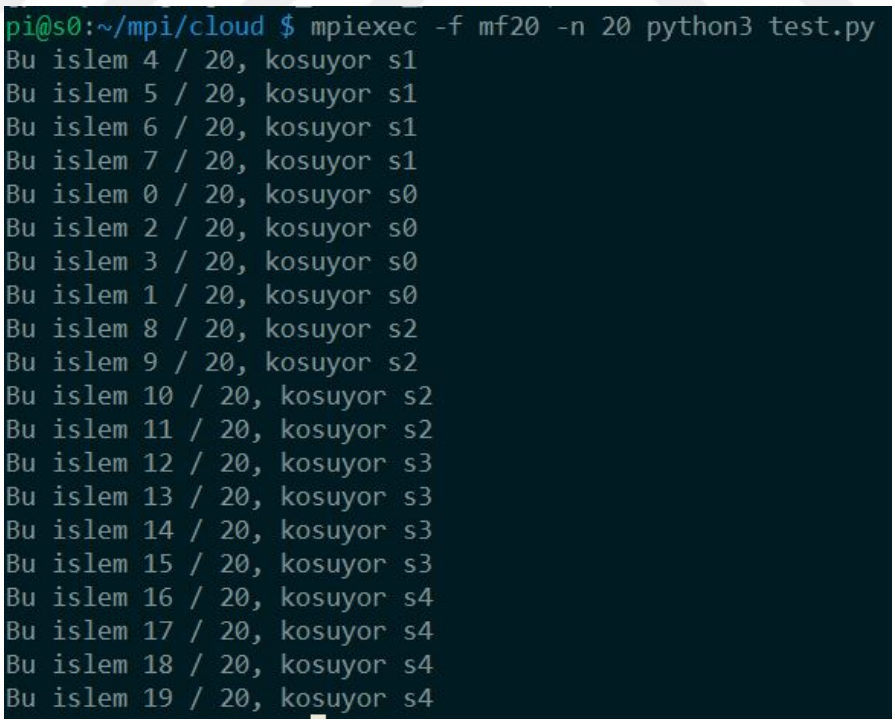
Her bir düğüme kaç işlemin (process) verileceği veya hangi düğümlere işlem (process) verileceğini belirlemek için bir machinefile (metin) dosyası oluşturulur. Bu dosyanın içinde düğümlerin ip adresleri yazılır ve kaç işlem (process) atılacağı belirtilir. Düğüme kaç işlem (process) verilebileceğini ip adresin sonuna ":" ile birlikte yazılır. Bu yüzden tüm düğümlere sabit ip adresi verilir.

Şu anlık düğümler birbirleri ile güvenlik sebebi ile haberleşemiyor. Haberleşmeleri için ssh (güvenlik kabuğu) ayarlarını yapmalıyız. Ana düğümde ssh anahtarı oluşturulur ve tüm uydu düğümlerin authorized_keys'e eklenir. Uydu düğümlerde ssh anahtarı oluşturulur ve ana düğümün authorized_keys'e eklenir. Artık ana düğüm, uydu düğümlere veri alıp gönderme işlemi yapabilir.

Sistemi daha kullanışlı bir hale getirmek için ana düğüm üzerinde diğer uydu düğümlerin ulaşabileceği ortak bir klasör oluşturulur ve gerekli ayarlar yapılır. Bunun için NFS (Network File System) servisinden faydalanılır. Yazdığımız kaynak kodunu buraya yükleyerek tüm düğümlerin buradan okumasını sağlarız.

Artık sistemimizin donanımsal ve yazılımsal kurulumları ve ayarları yaptık. Oluşturduğumuz cluster sisteminde çalıştırmak istediğimiz kaynak kodu şu komut ile çalıştırırız : *mpiexec –f [düğümlerinBulunduğuDosya] –n [kaçIşlemVerilcek] python3 [kaynakKodDosyası]* örnek olarak şekil Verilmiştir.

Şekil 3.20'deki kurduğumuz sistemin sorunsuz bir şekilde çalıştığı görülmektedir.



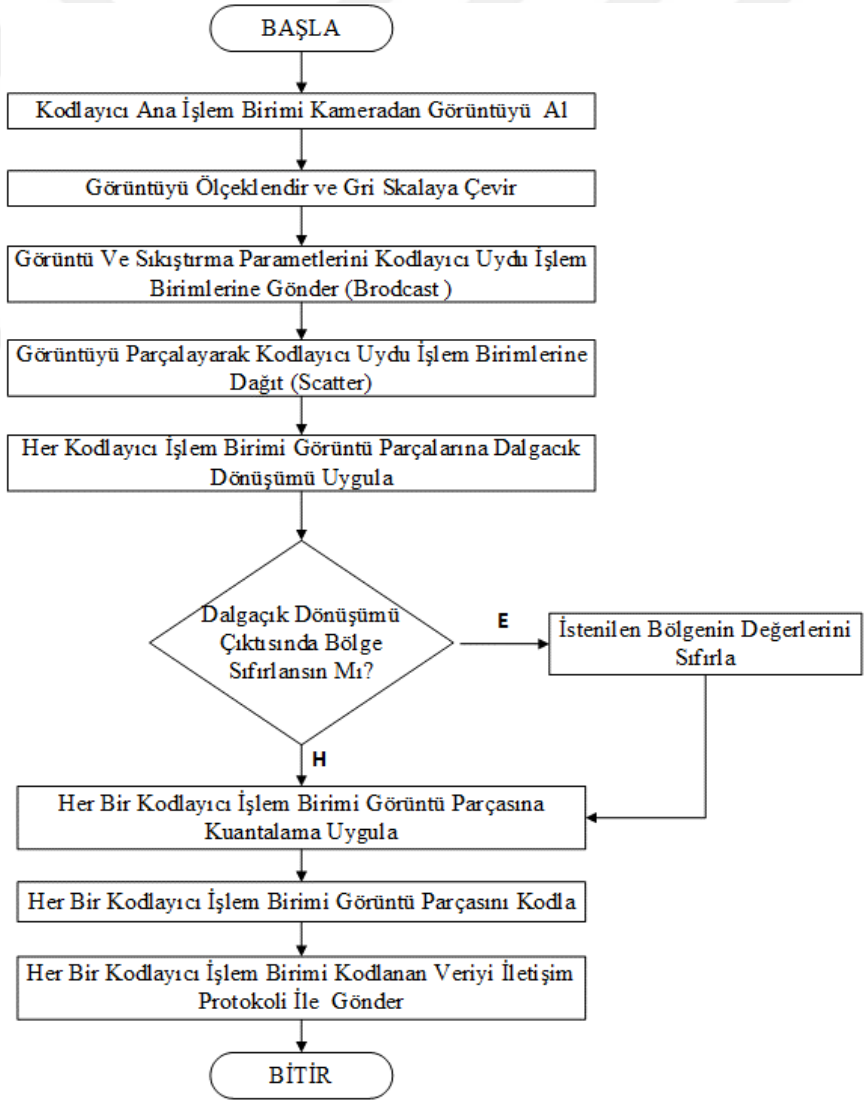

Şekil 3. 2 – 20 İşlemin paralel olarak çalışma çıktısı

## 3.2 Görüntü Sıkıştırma Ve Çözümleme İçin Paralel Algoritmanın Oluşturulması Ve Gerçeklemesi

Bu bölümde kurmuş olduğumuz cluster sisteminde çalışacak algoritmamız anlatılacak. Algoritmamızda görüntü sıkıştırma için önceden bahsettiğimiz şu bölümlerdeki 2.2.2.3, 2.3 ve 2.4 metodlar kullanılmıştır. Görüntü sıkıştırma ve görüntü çözme tekniğini paralelleştirmek için algoritmalar tasarlanmıştır. Bu algoritmalar python dili ile yazılmıştır.

### 3.2.1 Görüntü Sıkıştırma Algoritması

Şekil 3.3'de görüntü sıkıştırma algoritmasının genel akış şeması verilmiştir.




Şekil 3. 3 - Görüntüyü paralel olarak kodlama akış şeması

Paralel programlama başlatmak için tüm kodlayıcı işlem birimlerin bir birleri ile haberleşmeleri gerekmektedir. Bunu için kullanabileceğimiz komutlardan biri MPI kütüphanesinde bulunan '*MPI.COMM_WORLD*' komutudur. Toplam kodlayıcı işlem sayısı öğrenmek için MPI kütüphanesinden '*Get_size()*' kullanılır. Kodlayıcı işlemlere görev atamak için sırasına ihtiyaç duyulur. Bu bilgiyi edinmek için '*Get_rank()*' fonksiyonu kullanılır. Bu işlemler Şekil 3.4'deki kod parçası ile yapılmıştır.

```
1    comm = MPI.COMM_WORLD
2    size = comm.Get_size()
3    rank = comm.Get_rank()
```

Şekil 3. 4 - İşlemlerin bir birleri ile haberleşme kodu

Kodlayıcı işlem birimlerinin birbirleriyle haberleşmesini sağladıktan sonra dijital görüntünün kodlayıcı ana işlem biriminde olması için bir kamera ile görüntü alınabilir veya görüntü direk içine yüklenebilir. Önemli olan noktalardan bir tanesi görüntünün yüksekliği ve genişliği çift sayı olması gerekmektedir. Sıkıştırma için uygulayacağımız bölüm…. yöntem buna ihtiyaç duymaktadır. Dijital görüntünün yüksekliği veya genişliği çift değilse ihtiyaç duyulduğu kadar sıfırlardan oluşan satır veya kolon eklenir. Bu sayede dijital görüntü algoritmamızın uygulanması için hazır hale gelmiş bulunur. Hazır hale gelmiş dijital görüntüyü tek katmana indirmek için gri skalaya çevrilir. Gri 'ye dönüştürülen dijital görüntüyü numpy kütüphanesi kullanarak '*numpy.asarray()*' fonksiyonu ile dijital görüntü 2 boyutlu dizi'ye aktarılır.

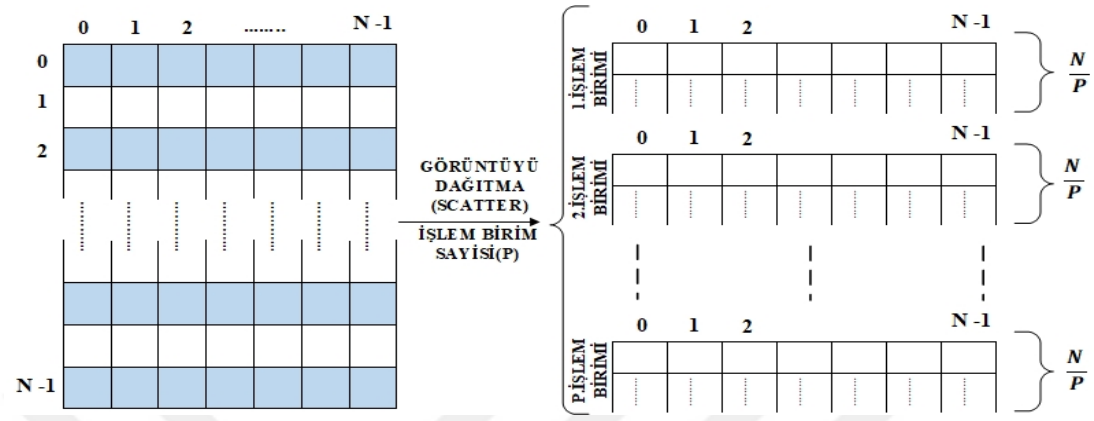
Kodlayıcı ana işlem birimi dijital görüntünün yüksekliği, genişliği, hangi bölgenin sıfırlanacağı bilgisi, kuantalama adımı, exp-golomb derecesi bilgilerini mpi4py kütüphanesi içindeki '*MPI.bcast*' fonksiyonu kullanılarak. Tüm kodlayıcı işlem birimlerine bu değerler gönderilir. Bu işlemler Şekil 3.5'deki kod parçası ile yapılmıştır.

```
1    comm.bcast(foto_yukseklik, root=0)
2    comm.bcast(foto_genislik, root=0)
3    comm.bcast(sifirlama_degeri, root=0)
4    comm.bcast(delta, root=0)
5    comm.bcast(exp_derece, root=0)
```

Şekil 3. 5 - Parametrelerin tüm işlem birimlerine gönderilme kodu

2 boyutlu dizi'ye dönüştürülen dijital görüntümüz. Paralel bir şekilde işlem yapabilmek için elimizdeki diziyi satır satır parçalayarak diğer tüm kodlayıcı işlem

birimlerine dağıtırız. Kodlayıcı işlem birimlerine ne kadarlık bir satır bloğu gideceğini şu şekilde hesaplanır. P toplam kodlayıcı işlem birimi adedi olmak üzere, kodlayıcı ana işlem biriminde bulunan 2 boyutlu *NxN* diziyi satır sayısı *N* olduğundan *N/P* şeklinde hesaplanır. Şekil 3.6'da gösterilmiştir.




Şekil 3. 6 - Görüntünün oluşturulan işlem birimlerine dağıtılması

Bu işlemi yapmak için MPI kütüphanesi içinde '*comm.Scatter()*' fonksiyonu kullanılmıştır. Bu fonksiyonun kod parçası Şekil 3.7'de gösterilmiştir.

```
dugumlere_dagitilan_parca = numpy.empty(
        (foto_yukseklik // size, foto_genislik), dtype=numpy.int16)
comm.Scatter(foto_grey_veri, dugumlere_dagitilan_parca, root=0)
```

Şekil 3. 7 - 2-Boyutlu diziyi işlem birimlerine dağıtma kodu

Burada dikkat edilmesi gerekenlerden bir konu *N/P* bir çift sayı çıkması gerekir. Daha önceden bahsettiğimiz bölüm 2.2.2.3'de anlattığımız yöntemi uygulamak için dizinin satır sayısı çift olması gerekmektedir. Eğer *N/P* çift sayı değilse dağıtılmadan önce 2 boyutlu dizinin satır sonuna gerektiği kadar sıfır elemanlarından oluşan satır eklenir.

Artık tüm kodlayıcı işlem birimlerinde 2 boyutlu dizinin eşit büyüklükteki parçaları bulunmaktadır. Her kodlayıcı işlem birimi elinde bulunan 2 boyutlu parçaya ilk önce satırlara haar dalgacık dönüşümü uygulanır. Bunun için *dwt_yatay* fonksiyonu kullanılmıştır. Bu fonksiyonun kod parçası Şekil 3.8'de gösterilmiştir.

```
def dwt_yatay(data):
    even_yatay = data[:, 0::2]
    odd_yatay = data[:, 1::2]
    even_yatay_listesi = (odd_yatay + even_yatay) / 2
    # even_yatay_listesi = even_yatay_listesi.astype(numpy.int16)
    odd_yatay_listesi = odd_yatay - even_yatay
    dwt_yeni = numpy.concatenate(
            (even_yatay_listesi, odd_yatay_listesi), axis=1)
    return dwt_yeni
```

Şekil 3. 8 – Satırların ADD alan python fonksiyonu

Yatay olarak haar dalgacık dönüşümü sonrası elde edilen 2 boyutlu diziye sütunlara haar dalgacık dönüşümü uygulanır. Bunun için *dwt_dikey* fonksiyonu kullanılmıştır. Bu fonksiyonun kod parçası Şekil 3.9'da gösterilmiştir.

```
def dwt_dikey(data):
    even_dikey = data[0::2]
    odd_dikey = data[1::2]
    even_dikey_liste = (even_dikey + odd_dikey) / 2
    # even_dikey_liste = even_dikey_liste.astype(numpy.int16)
    odd_dikey_liste = odd_dikey - even_dikey
    dwt_yeni = numpy.concatenate(
            (even_dikey_liste, odd_dikey_liste), axis=0)
    dwt_yeni = dwt_yeni.astype(numpy.int16)
    return dwt_yeni
```

Şekil 3. 9 – Sütunların ADD alan python fonksiyonu

Bu iki işlemin sonunda dağıtılan dijital görüntünün 1 boyutta haar dalgacık dönüşümü alınmış olur ve sonucunda dört adet haar dalgacık katsayıları elde edilmiş olur. Bunlar Yüksek-yüksek geçişli katsayı (HH), yüksek-düşük geçişli katsayı (HL), düşük-yüksek geçişli katsayı (LH), düşük-düşük geçişli katsayı (LL) elde edilir. Bu katsayıların oluşturduğu HL, LH, LL bölgelerden bir veya daha fazlasını sıfırlayarak görüntüyü kayıplı bir şekilde sıkıştırmamızda yardımcı olur. Bunun için *goruntu_eleman_sifirlama* fonksiyonu kullanılmıştır. Bu fonksiyonun kod parçası Şekil 3.10'da gösterilmiştir.

```
def goruntu_eleman_sifirlama(liste, sayi):
    if sayi == 0:
        liste
    elif sayi == 1:
        liste[:foto_yukseklik // (size * 2), foto_genislik // 2:] = 0
    elif sayi == 2:
        liste[foto_yukseklik // (size * 2):, : foto_genislik // 2] = 0
    elif sayi == 3:
        liste[foto_yukseklik // (size * 2):, foto_genislik // 2:] = 0
    elif sayi == 4:
        liste[foto_yukseklik // (size * 2):] = 0
    elif sayi == 5:
        liste[:, foto_yukseklik // 2:] = 0
    elif sayi == 6:
        liste[:foto_yukseklik // (size * 2), foto_genislik // 2:] = 0
        liste[foto_yukseklik // (size * 2):, : foto_genislik // 2] = 0
    elif sayi == 7:
        liste[:foto_yukseklik // (size * 2), foto_genislik // 2:] = 0
        liste[foto_yukseklik // (size * 2):] = 0
    return liste
```

Şekil 3. 10 - Bölgeleri sıfırlayan python fonksiyonu

Dalgacık dönüşümü yapılan dijital görüntüyü daha iyi bir sıkıştırma oranı yakalaya bilmek için kuantalama işlemi yapılır. Kuantalama işlemi daha geniş değer aralığını daha küçük değerler kümesi ile temsil etmektir. Bu sayede dalgacık katsayılar daha düşük değerlere indirgenir. Bunun için *kuantalama* fonksiyonu kullanılmıştır. Bu fonksiyonun kod parçası 3.11'de gösterilmiştir.

```
1   def kuantalama(data,
2    delta, yukseklik, genislik):
3       kuantalama_dizi = []
4       for i in numpy.reshape(
5            data, (yukseklik * genislik)):
6           q = i // delta
7           kuantalama_dizi.append(q)
8       return kuantalama_dizi
```

Şekil 3. 11 – Kuantalama alan python fonksiyonu

Kuantalanmış dalgacık katsayılarımızı exp-golomb ile kodlama yaparak dijital görüntümüzü sıkıştırmış oluruz. Kuantalanmış katsayıların çoğu düşük değerler içerdiğinden exp-golomb derecesi sıfır seçilmiştir. Bu sayede exp-golomb düşük değerli

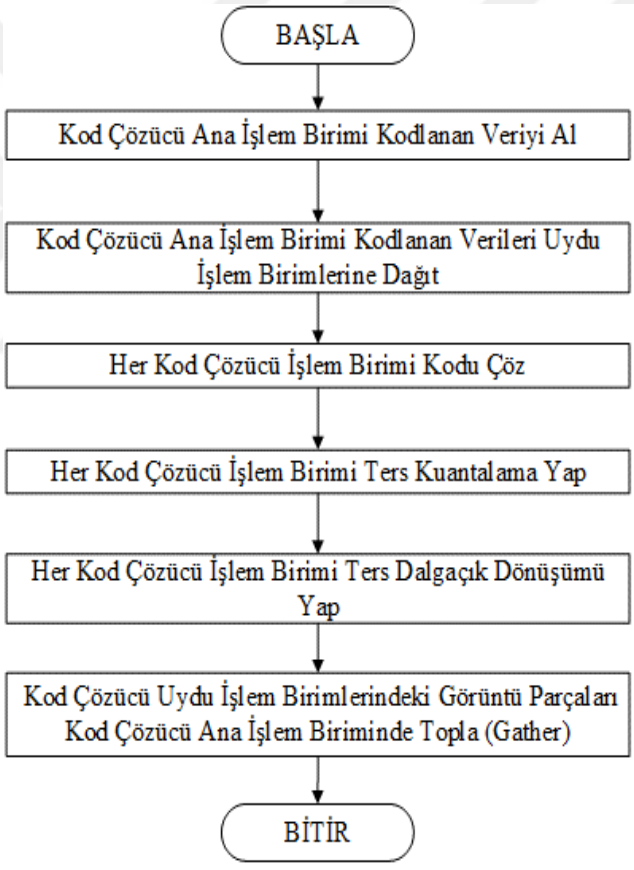
```
1   def exp_golomb(dizi, k_derece, yukseklik, genislik):
2       exp_golomb_dizi = ""
3       for dizi_elemani in numpy.reshape(
4            dizi, (yukseklik * genislik)):
5           for np_uzunlugu in range(1, 10):
6               if math.pow(2, np_uzunlugu - 1) - 1 <=
7                dizi_elemani < math.pow(2, np_uzunlugu) - 1:
8                   on_ek = (np_uzunlugu - 1) * "0" + "1"
9                   ns_uzunlugu = k_derece + np_uzunlugu - 1
10                  vs = int(dizi_elemani - math.pow(2, k_derece
11                  ) * (math.pow(2, np_uzunlugu - 1) - 1))
12                  son_ek = numpy.binary_repr(vs, ns_uzunlugu)
13                  if ns_uzunlugu == 0:
14                      exp_golomb = on_ek
15                  else:
16                      exp_golomb = on_ek + son_ek
17                  exp_golomb_dizi += exp_golomb
18                  break
19      return exp_golomb_dizi
```

Şekil 3. 12 – Exp-Golomb alan python fonksiyonu

ifadeleri daha az bit ile ifade edebilir. . Bunun için *exp_golomb* fonksiyonu kullanılmıştır. Bu fonksiyonun kod parçası 3.12'de gösterilmiştir.

### 3.2.2 Görüntünü Çözme Algoritması

Dijital görüntünün parçaları her bir kodlayıcı işlem birimlerinde, kodlanmış bir şekilde bulunmaktadır. Buraya kadarki işlemler sonucunda dijital görüntümüzün sıkıştırma işlemleri tamamlanmıştır. Bundan sonraki adımlar sıkıştırılan dijital görüntünün iletilmesi ve sıkıştırılmış olan görüntünün çözülmesidir. Sıkıştırılan görüntü iletişim protokolüne uygun bir şekilde gönderilir. Burada sıkıştırılmış dijital görüntünün nasıl paralel olarak çözümlenmesine değineceğiz ve bunun için geliştirdiğimiz algoritmayı anlatacağız. Sıkıştırılmış görüntüyü çözümlemek için sırası ile kod çözme, ters kuantalama, ters dalgacık dönüşümü yapılır. Şekil 3.13'de bu algoritmanın genel akış şeması verilmiştir.




Şekil 3. 13 – Kayıplı görüntüyü paralel olarak geri elde etme akış şeması

Kod çözücü ana işlem birimine gelen sıkıştırılmış dijital görüntü verileri, kod çözücü uydu birimlerine '*comm.Scatter()*' fonksiyonu ile dağıtır. Her kod çözücü birimi elindeki exp-golomb ile kodlanmış dijital görüntü parçasını çözümler. Bunun için

*inverse_exp_golomb* fonksiyonu kullanılmıştır. Bu fonksiyonun kod parçası 3.14'de gösterilmiştir.

```
def inverse_exp_golomb(dizi, k_derece, yukseklik, genislik):
    orj_dizi = []
    dizi_eleman_sayisi = 0
    for n in range(yukseklik * genislik):
        sifir_adeti = 0
        for i in range(dizi_eleman_sayisi, len(dizi)):
            sifir_adeti = sifir_adeti + 1
            if dizi[i] == "1":
                np_uzunluk = sifir_adeti
                ns_uzunluk = k_derece + np_uzunluk - 1
                np = dizi[i - np_uzunluk + 1: i + 1]
                ns = dizi[i + 1: i + ns_uzunluk + 1]
                dizi_eleman_sayisi = i + ns_uzunluk + 1
                if ns != "":
                    vs = int(ns, 2)
                    v = vs + math.pow(2, k_derece
                    ) * (math.pow(2, np_uzunluk - 1) - 1)
                    orj_dizi.append(int(v))
                else:
                    vs = 0
                    orj_dizi.append(int(vs))
                break
    return orj_dizi
```

Şekil 3. 14 - Ters exp-golomb alan python fonksiyonu

Her kod çözücü işlem birimi kodları çözdükten sonra elde edilen diziye ters kuantalama işlemi uygulanır. Artık elimizde dalgacık dönüşümü alınmış dalgacık katsayılar bulunmaktadır. Bu dalgacık katsayıları kuantalama nedeni ile bazı verilerde kayıplar gelişmiştir. Ters kuantalama işlemi için *inverse_kuantalama* fonksiyonu kullanılmıştır. Bu fonksiyonun kod parçası 3.15'de gösterilmiştir.

```
def inverse_kuantalama(data, delta):
    ikuantalama_dizi = []
    for dizi_elemani in data:
        q = round((dizi_elemani + 0.5) * delta)
        ikuantalama_dizi.append(q)
    return ikuantalama_dizi
```

Şekil 3. 15 - Ters kuantalama alan  python fonksiyonu

Her bir kod çözücü işlem birimin elinde bulunan dalgacık katsayıların olduğu dijital görüntü parçalarına ters dalgacık dönüşümü uygulanacaktır. Ters dalgacık dönüşümü ilk önce sütunlara uygulanır. Bunun için *idwt_dikey* fonksiyonu kullanılmıştır. Bu fonksiyonun kod parçası şekil 3.16'da görülmektedir.

```
def idwt_dikey(fotonun_parcasi):
    foto_low, foto_high = numpy.vsplit(
            fotonun_parcasi, 2)
    yeni_low = foto_low - 0.5 * foto_high
    yeni_high = foto_low + 0.5 * foto_high
    yeni_low = yeni_low.astype(numpy.int16)
    yeni_high = yeni_high.astype(numpy.int16)
    sutun_low_high = numpy.empty(
            [foto_yukseklik // size, foto_genislik],
             dtype=numpy.int16)
    for x in range(foto_genislik):
        cift = 0
        tek = 1
        for i in range(foto_yukseklik // (size * 2)):
            sutun_low_high[cift, x] = yeni_low[i, x]
            sutun_low_high[tek, x] = yeni_high[i, x]
            cift += 2
            tek += 2
    return sutun_low_high
```

Şekil 3. 16 – Sütunlara ters ADD alan python fonksiyonu

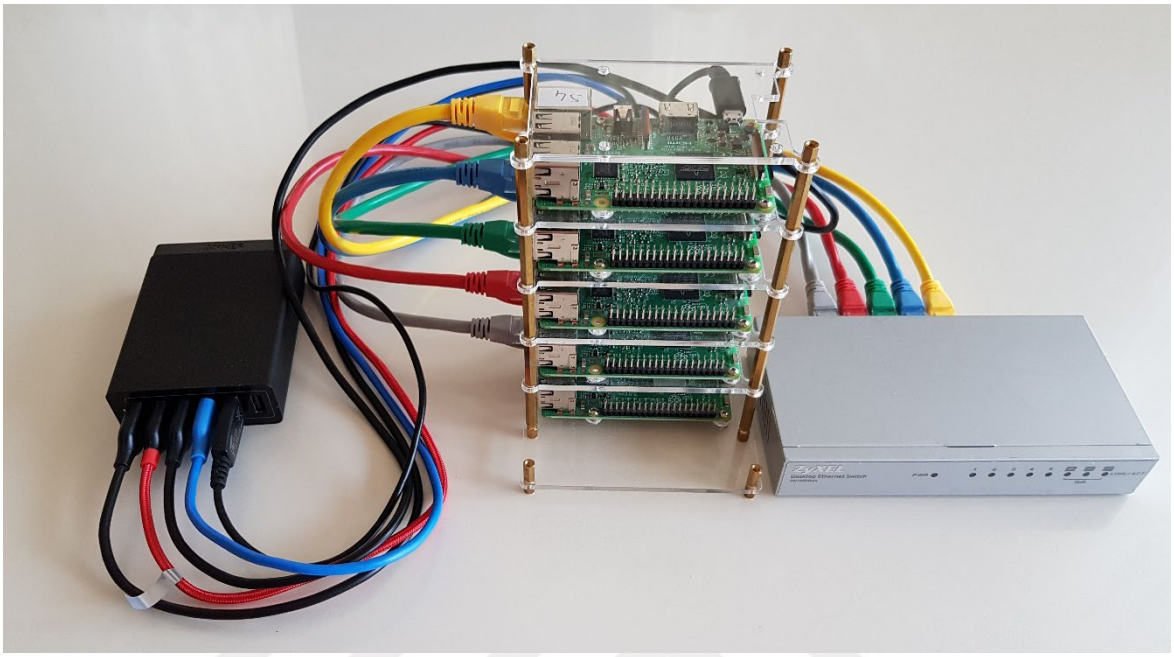
Sütunlarda biten ters dönüşüm işleminden sonra her bir kod çözücü işlem birimi satırlarda ters dönüşüm işlemi yapar. Bunun için *idwt_yatay* fonksiyonu kullanılmıştır. Bu fonksiyonun kod parçası şekil 3.17'de görülmektedir.

```
def idwt_yatay(fotonun_parcasi):
    foto_low, foto_high = numpy.hsplit(fotonun_parcasi, 2)
    even = foto_low - 0.5 * foto_high
    odd = foto_low + 0.5 * foto_high
    even = even.astype(numpy.int16)
    odd = odd.astype(numpy.int16)
    even_odd_birlestirme = numpy.empty(
            [foto_yukseklik // size, foto_genislik],
             dtype=numpy.int16)
    for x in range(foto_yukseklik // size):
        cift = 0
        tek = 1
        for i in range(foto_genislik // 2):
            even_odd_birlestirme[x, cift] = even[x, i]
            even_odd_birlestirme[x, tek] = odd[x, i]
            cift += 2
            tek += 2
    return even_odd_birlestirme.astype(numpy.int16)
```

Şekil 3. 17 - Satırlara ters ADD alan python fonksiyonu

Artık sıkıştırılmış dijital görüntüyü çözmüş bulunmaktayız. Çözülen dijital görüntü parçalar halinde tüm işlem birimlerinde bulunmaktadır. Parçalar halinde bulunan dijital görüntüyü ana kod çözücü işlem biriminde toplanmalıdır. Ana kod çözücü işlem biriminde toplamak için MPI kütüphanesi içinde '*comm.Gather()*' fonksiyonu kullanılır. Bunu sonucunda çözülmüş dijital görüntünün tamamı ana çözücü işlem biriminde bulunur ve sıkıştırılmış görüntü geri elde edilmiştir.

# BÖLÜM 4. DENEYSEL BULGULAR VE YORUMLAR



Şekil 4. 1 – Kurulan dağıtık bellekli paralel sistem

Bu çalışmada bölüm 3.1'de anlatıldığı üzere kurduğumuz deney düzeneği şekil 4.1'de görülmektedir. Bu deney düzeneğin üzerinde Bölüm 3.2'de anlatığımız algoritmayı iki farklı açıdan test edilmiştir.

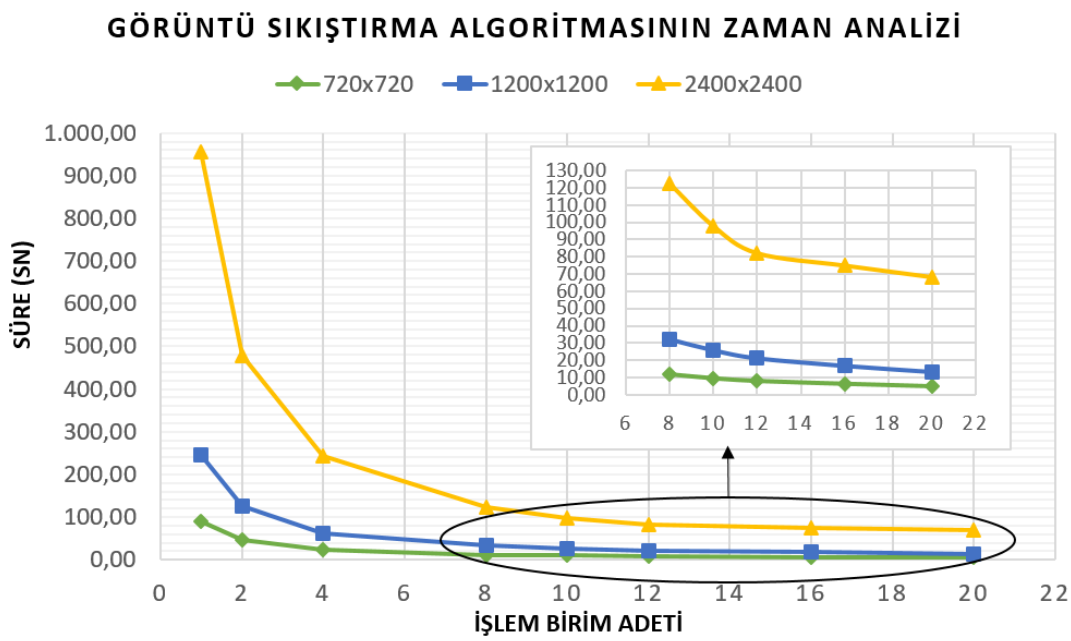
Birinci olarak paralel algoritmamızın görüntü sıkıştırma ve çözme işlemlerin farklı adet işlem birimleri ile ne kadarlık bir süre performansı sağlayacağını test edeceğiz. İkinci olarak görüntü sıkıştırma algoritmamızın PSNR ve sıkıştırma katsayısını test edeceğiz.

## 4.1 Görünüt Sıkıştırma Algoritmasının Zaman Analizi

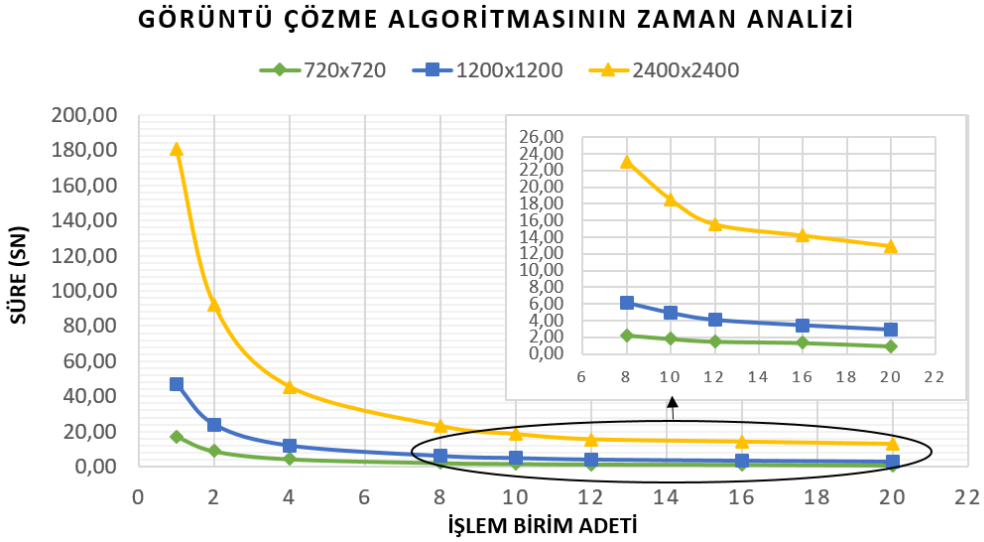
İşlem birim âdeti değişimine göre görüntü sıkıştırma ve görüntü çözme işlemlerinin ne kadar süre aldığı ölçülmüştür. Bunun için üç farklı çözünürlükte 720x720, 1200x1200, 2400x2400 piksellik çözünürlüğe sahip  görüntüler üzerinde sıkıştırma ve çözme işlemleri yapılmıştır. Bu görüntülerin kodlama ve çözme işlemlerini sırası ile 1, 2, 4, 8, 10, 12, 16, 20 adet işlem birimlerinde test yapılmıştır.

Zaman ölçümü yapılırken sıkıştırma ve çözümleme (görüntüyü geri elde etme) ayrı ayrı ölçülmüştür. Görüntüyü sıkıştırma işleminde zaman ölçümü başlangıcı ana kodlayıcı, görüntüyü uydu kodlayıcı birimlerine dağıttıktan sonra başlar ve sırası ile ADD dönüşümü, kuantalama, exp-golomb kodlama işlemleri bitimi ile zaman ölçümü biter. Görüntü çözümleme (görüntüyü geri elde etme) işleminde zaman ölçümü başlangıcı ana kod çözücü sıkıştırılmış görüntüyü uydu kod çözücülere dağıttıktan sonra başlar ve ters exp-golomb, ters kuantalama, ters ADD işlemleri bitimi ile zaman ölçümü biter.

Şekil 4.1'deki görüntü sıkıştırma ve şekil 4.2'daki görüntü çözme sonuçlarına göre işlemci birim adetinin artması ile sürenin düştüğü gözlenmiştir. İşlem biriminin arttığı oranda sürenin doğrusal oranda düşmesi beklenirken bu tam olarak gerçekleşmemiştir. Çünkü işlem birimlerin arasındaki iletişim bir yük getirmektedir. Bu da ek bir süre demektir. Düğüm sayısının artması ile bu işlem yükü artmaktadır.




Şekil 4. 2 - İşlem birim âdetine göre görüntü sıkıştırma süreleri




Şekil 4. 3 - İşlem birim âdetine göre görüntü çözme süreleri

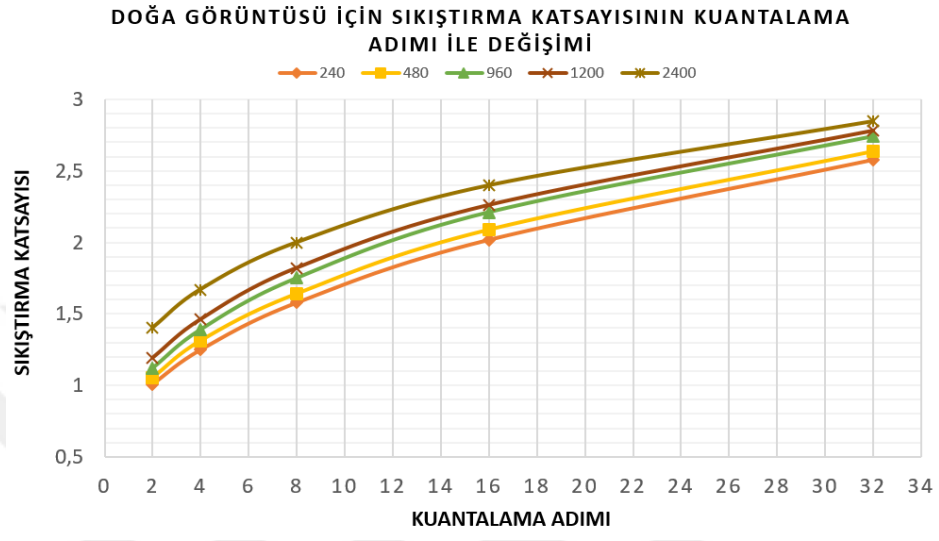
## 4.2 Algoritmanın Başarım Analizi Ve PSNR Bulguları

Dijital görüntümüzü sıkıştırmak için kullandığımız algoritma test etmek için 5 adet farklı kategoride dijital görüntü seçilmiştir. Bunlar doğa, şehir, insan yüzü, nesne ve sanal görüntülerden oluşmaktadır. Bu dijital görüntülerin 5 farklı 240x240, 480x480, 960x960, 1200x1200, 2400x2400 piksellik çöznürlükteki görüntülerin kuantalama adımı değiştirilerek PSNR ve sıkıştırma katsayıları karşılaştırılmıştır. Exp-golomb derecesi 0 olarak seçilmiştir.

### 4.2.1 Doğa Görüntüsü

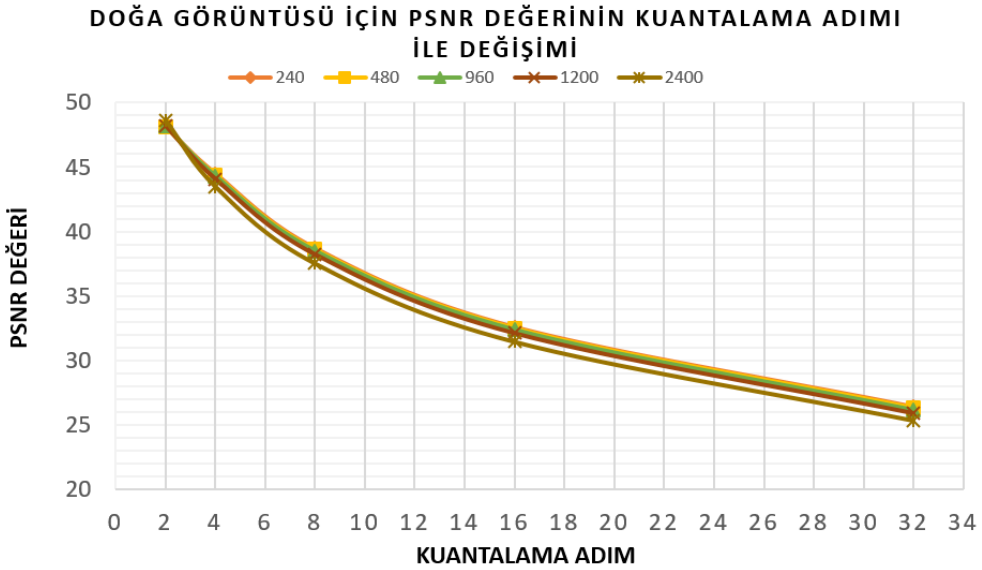
Şekil 4. 4 – Orijinal doğa görüntüsü [21]

Şekil 4.4'deki görüntünün yukarda belirtiğimiz 5 farklı çözünürlükleri üzerinde Haar Dalgacık Dönüşümü, belli kuantalama değerleri ile birlikte ve sıkıştırma uygulanmıştır. Bunun sonucunda elde edilen katsayıların sonucu şekil 4.5'de ve PSNR sonuçlarıda şekil 4.6'da bulunmaktadır.




Şekil 4. 5 - Doğa görüntünün PSNR Grafiği

K uantalama değerini artırdıkça görüntünün sıkıştırma katsayısı artmaktadır. Görüntünün düşük çözünürlükten yüksek çözünürlüğe doğru gidildikçe sıkıştırma katsayısı artmaktadır.




Şekil 4. 6 - Doğa görüntünün sıkıştırma grafiği

Kuantalama adımları artıkça PSNR değeri düştüğü görülmektedir. Kuantalama adımı 4 ve 4 ‘ten sonra yüksek çözünürlükten düşük çözünürlüğe doğru gidildikçe PSNR değeri düşmektedir.

a) Kuantalama basamağı = 8 b) Kuantalama basamağı = 32

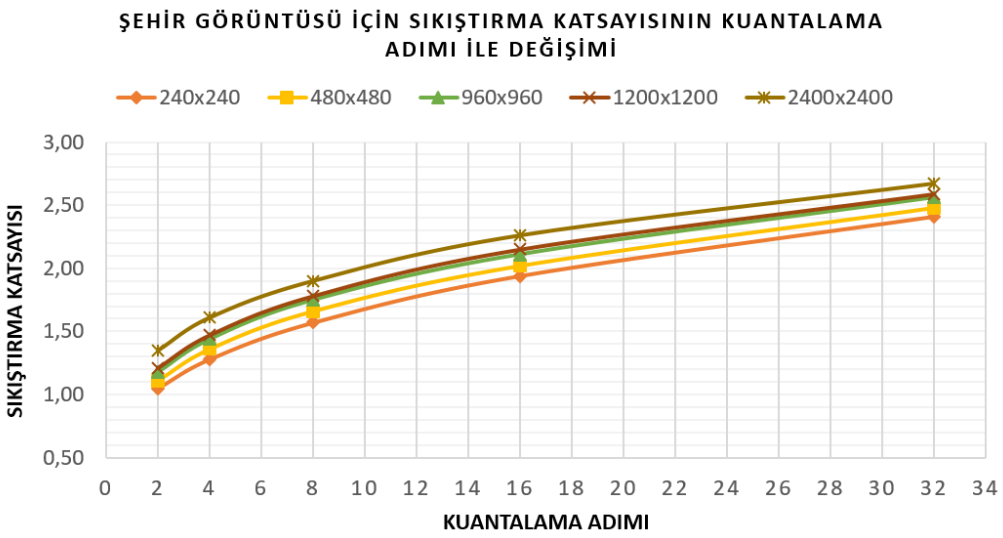
Şekil 4. 7 – 2048x2048 Piksel Görüntünün geri elde edilmesi

Şekil 4.7’de orijinal görüntüye belli kuantalama adımları uygulandıktan sonraki görüntünün geri elde edilmesi görmekteyiz. Kuantalama basamağı 8 uygulanan görüntü ile orijinal görüntü arasında gözle görünür çok az bir kayıp bulunmaktadır. Kuantalama basamak 32 için gözle görünür bir kayıp bulunmakta ama hala görüntünün detayları gayet belirgindir. Buna karşılık sıkıştırma katsayılarında 2’den 2,85’e çıkmıştır.

### 4.2.2 Şehir Görüntüsü

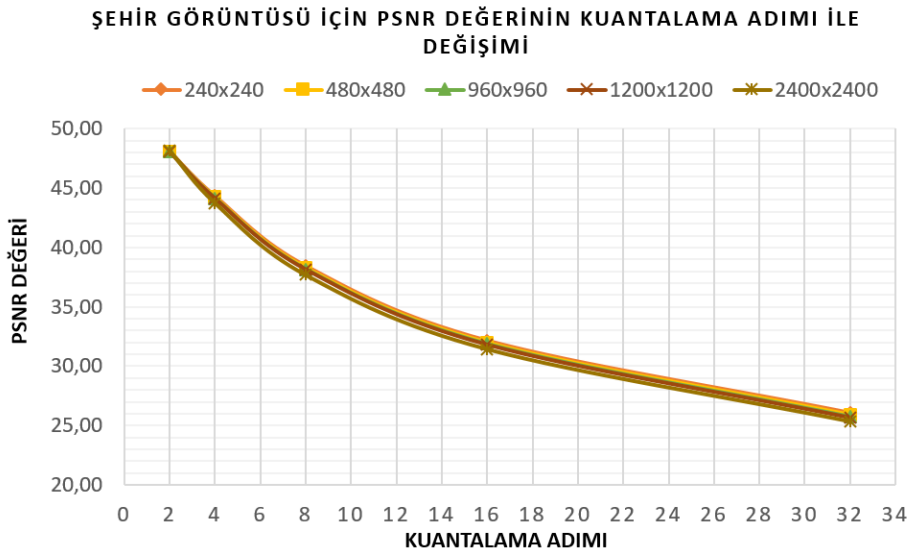
Şekil 4. 8 - Orijinal Şehir Görüntüsü [22]

Şekil 4.8'deki görüntünün yukarda belirtiğimiz 5 farklı çözünürlükleri üzerinde yukarıda bahsettiğimiz işlemler uygulanmıştır. Bunun sonucunda elde edilen katsayıların sonucu şekil 4.9'de ve PSNR sonuçlarıda şekil 4.10'da bulunmaktadır.




Şekil 4. 9 - Şehir görüntüsünde sıkıştırma grafiği

Kuantalama adımını artırdıkça görüntünün sıkıştırma katsayısı artmaktadır. Görüntünün düşük çözünürlükten yüksek çözünürlüğe doğru gidildikçe sıkıştırma katsayısı artmaktadır.




Şekil 4. 10 - Şehir görüntünün PSNR grafiği

Kuantalama adımı artıkça PSNR değeri düştüğü görülmektedir. Yüksek çözünürlükten düşük çözünürlüğe doğru gidildikçe PSNR değeri düşmektedir.

Şekil 4. 11 - 2048x2048 Piksel görüntünün geri elde edilmesi

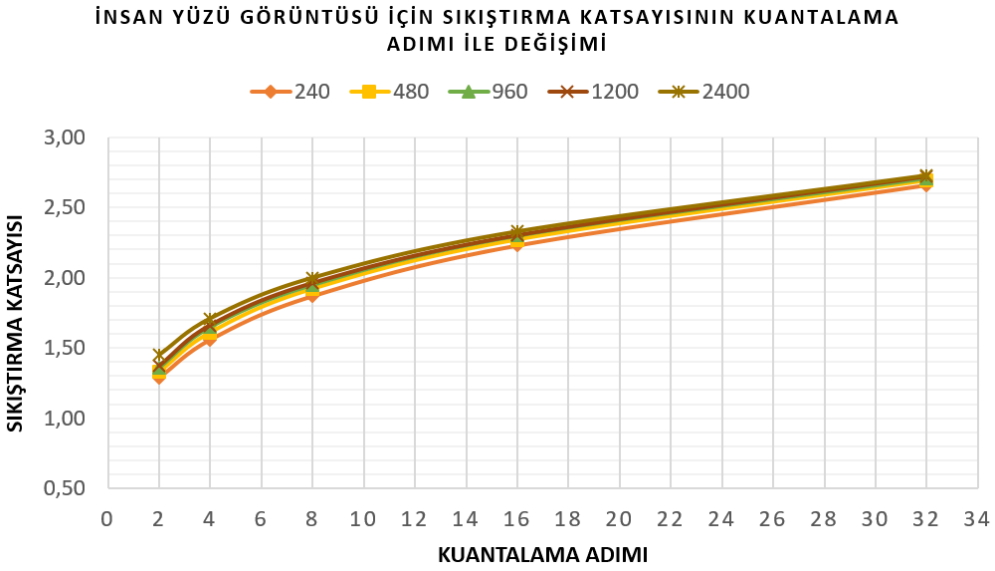
Şekil 4.11'de orijinal görüntüye belli kuantalama değerlerin uygulandıktan sonraki görüntünün geri dönüşümünü görmekteyizi. Kuantalama 8 değeri uygulanan görüntü ile orijinal görüntü arasında gözle görünür çok az bir kayıp bulunmaktadır. Kuantalama 32 değeri için gözle görünür bir kayıp bulunmakta ama hala görüntünün

detayları gayet belirgindir. Buna karşılık sıkıştırma katsayılarında 1.90'den 2,67'e çıkmıştır.

### 4.2.3 İnsan Yüzü Görüntüsü

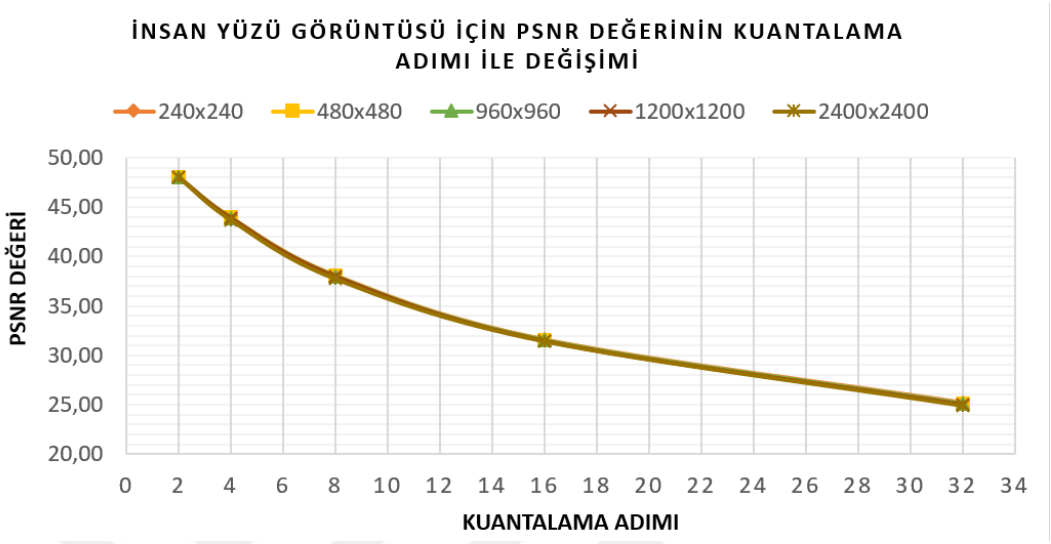
Şekil 4. 12 – Orijinal insan yüzü görüntüsü [23]

Şekil 4.12'deki görüntünün yukarda belirtiğimiz 5 farklı çözünürlükleri üzerinde yukarıda bahsettiğimiz işlemler uygulanmıştır. Bunun sonucunda elde edilen katsayıların sonucu şekil 4.13'de ve PSNR sonuçlarıda şekil 4.14'da bulunmaktadır.




Şekil 4. 13 – İnsan yüzü görüntünün sıkıştırma Grafiği

Kuantalama adımı artırdıkça görüntünün kodlama katsayısı artmaktadır. Görüntünün düşük çözünürlükten yüksek çözünürlüğe doğru gidildikçe kodlama katsayısı artmaktadır.




Şekil 4. 14 – İnsan Yüzü Görüntüsünün PSNR Grafiği

Kuantalama adımı artıkça PSNR değeri düştüğü görülmektedir. Yüksek çözünürlükten düşük çözünürlüğe doğru gidildikçe PSNR değeri düşmektedir şayet o değerler farkı çok azdır.

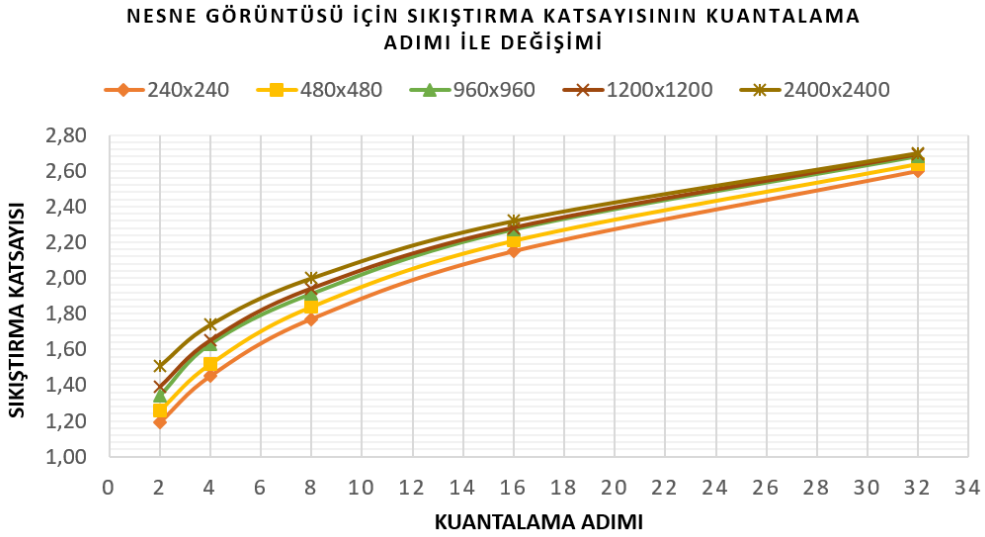

Şekil 4. 15 - 2048x2048 Piksel görüntünün geri elde edilmesi

Şekil 4.15’de orijinal görüntüye belli kuantalama adımları uygulandıktan sonraki görüntünün geri dönüşümünü görmekteyiz. Kuantalama adımı 8 uygulanan görüntü ile orijinal görüntü arasında gözle görünür çok az bir kayıp bulunmaktadır. Kuantalama adımı 32 için gözle görünür bir kayıp bulunmakta ama hala görüntünün detayları gayet belirgindir. Buna karşılık sıkıştırma katsayılarında 2’den 2,73’e çıkmıştır.

### 4.2.4 Nesne Görüntüsü

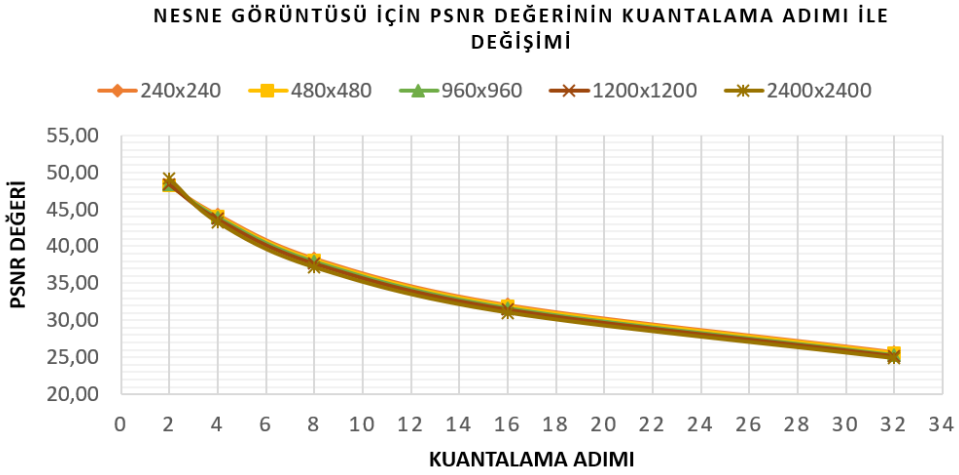
Şekil 4. 16 – Orijinal nesne görüntüsü [24]

Şekil 4.16’deki görüntünün yukarda belirtiğimiz 5 farklı çözünürlükleri üzerinde yukarıda bahsettiğimiz işlemler uygulanmıştır. Bunun sonucunda elde edilen katsayıların sonucu şekil 4.17’de ve PSNR sonuçlarıda şekil 4.18’da bulunmaktadır.




Şekil 4. 18 - Nesne görüntüsünün sıkıştırma grafiği

Kuantalama adımı artırdıkça görüntünün sıkıştırma katsayısı artmaktadır. Görüntünün düşük çözünürlükten yüksek çözünürlüğe doğru gidildikçe sıkıştırma katsayısı artmaktadır.




Şekil 4. 17 - Nesne Görüntüsünün Grafiği

Kuantalama adımı artıkça PSNR değeri düştüğü görülmektedir. Kuantalama adımı 4 ve 4 ‘ten sonra yüksek çözünürlükten düşük çözünürlüğe doğru gidildikçe PSNR değeri düşmektedir.

a) Kuantalama basamağı = 8 b) Kuantalama basamağı = 32

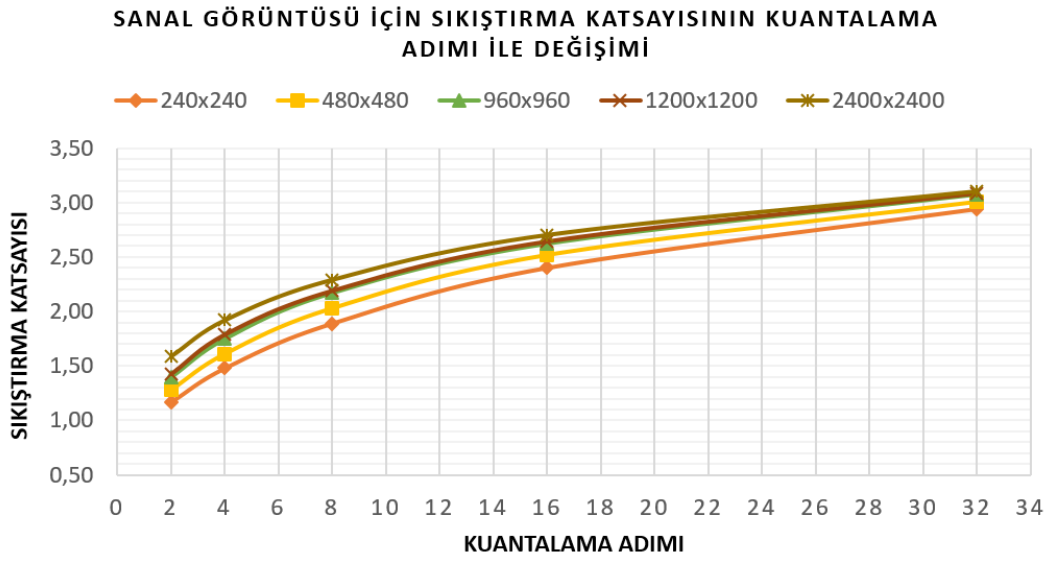
Şekil 4. 19 - 2048x2048 Piksel görüntünün geri elde edilmesi

Şekil 4.19'de orijinal görüntüye belli kuantalama adımları uygulandıktan sonraki görüntünün geri dönüşümünü görmekteyizi. Kuantalama adımı 8 uygulanan görüntü ile orijinal görüntü arasında gözle görünür çok az bir kayıp bulunmaktadır. Kuantalama adımı 32 için gözle görünür bir kayıp bulunmakta ama hala görüntünün detayları gayet belirgindir. Buna karşılık sıkıştırma katsayılarında 2'den 2,70'e çıkmıştır.

### 4.2.5 Sanal Görüntüsü

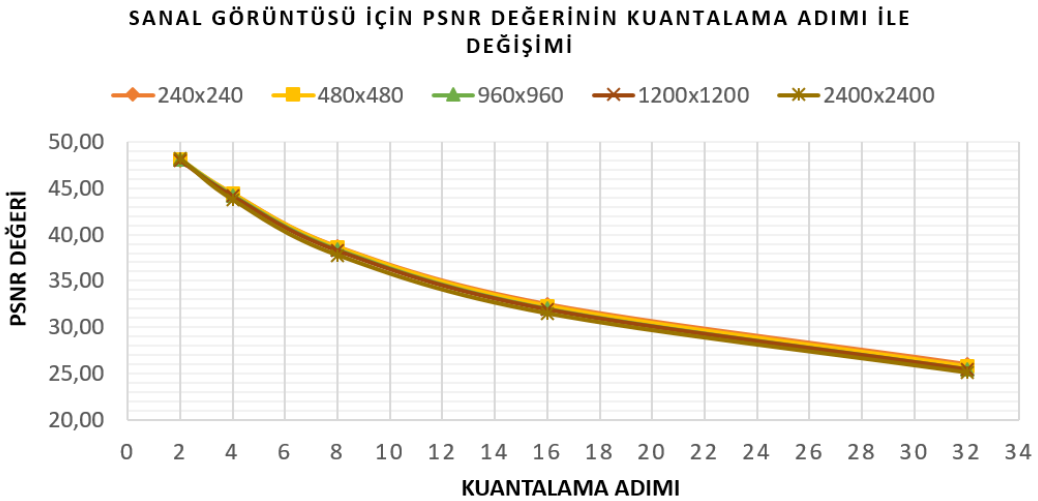
Şekil 4. 20 – Orijinal sanal görüntü [25]

Şekil 4.20'deki görüntünün yukarda belirtiğimiz 5 farklı çözünürlükleri üzerinde yukarıda bahsettiğimiz işlemler uygulanmıştır. Bunun sonucunda elde edilen katsayıların sonucu şekil 4.21'de ve PSNR sonuçlarıda şekil 4.22'da bulunmaktadır.




Şekil 4. 21 – Sanal görüntüsünün sıkıştırma grafiği

Kuantalama adımı artırdıkça görüntünün sıkıştırma katsayısı artmaktadır. Görüntünün düşük çözünürlükten yüksek çözünürlüğe doğru gidildikçe sıkıştırma katsayısı artmaktadır.




Şekil 4. 22 – Sanal görüntüsünün PSNR grafiği

Kuantalama adımı artıkça PSNR değeri düştüğü görülmektedir. Yüksek çözünürlükten düşük çözünürlüğe doğru gidildikçe PSNR değeri düşmektedir..

a) Kuantalama basamağı = 8 b) Kuantalama basamağı = 32

Şekil 4. 23 - 2048x2048 Piksel görüntünün geri elde edilmesi

Şekil 4.23'de orijinal görüntüye belli kuantalama değerlerin uygulandıktan sonraki görüntünün geri dönüşümünü görmekteyiz. Kuantalama adımı 8 uygulanan görüntü ile orijinal görüntü arasında gözle görünür çok az bir kayıp bulunmaktadır. Kuantalama adımı 32 değeri için gözle görünür bir kayıp bulunmakta ama hala görüntünün detayları gayet belirgindir. Buna karşılık kodlama katsayılarında 2,29'den 3,1'e çıkmıştır.

# BÖLÜM 5. SONUÇ

Bu tez çalışmasında görüntüyü sıkıştırmak ve görüntüyü çözümleme tekniğini paralel algoritma olarak gerçekleştirilmiş. Görüntüdeki piksel değeri artması ile yapılan görüntüyü sıkıştırmak ve çözme işlemleri sürenin uzadığı görülmektedir. Görüntünün çözünürlüğü sabit tutulup işlem biri adeti artırıldığında görüntü sıkıştırma ve çözümleme süresi düşmektedir. Normalde beklenen işlem birimi kaç kat arttıysa aynı oranda sürenin düşmesidir. Bu tez çalışmasında yapılan testler sonucunda bu durum böyle olmadığı saptanılmıştır. Çünkü işlem birim sayısı arttıkça bu işlem birimlerin birbirleri ile haberleşme yükü de artmaktadır. Bu da sisteme yük getirmektedir. Elimizdeki 20 işlem birimine kadar denediğimizde hala görüntü sıkıştırma ve çözümlemede süre azalması görülmektedir.

Paralel algoritma ile yapılan görüntü sıkıştırma işlemlerinin sonucunda, ölçtüğümüz sıkıştırma katsayısı ve PSNR değerlerinin seri algoritma ile yapılan aynı ölçümler kıyaslandığında bir fark olmadığı görülür.

Görüntü sıkıştırma algoritmamızda kuantalama adımı ve görüntü çözünürlüğü artırdıkça görüntünün sıkışma kat sayısı artmaktadır. Şayet kuantalama adımı artıkça görüntünün bozulmasını ölçen PSNR değeri düşmektedir. Bu bozulmanın insan gözü ile net fark edilme eşiği bizim testlerimizide kuantalama adımı 32 olduğunu söylenebilir. Bu 5 farklı görüntünün kuantalama adımı 32 seçilen PSNR sonuçlarını incelediğimizde hepsinin aldığı değer 28'in altındadır. Buda görüntülerin net bozulduğu gösteriyor. Görüntü çözünürlüğü artıkça sıkıştırma katsayısı artığı halde PSNR değeri neredeyse değişmemektedir. Buda günümüzde artan görüntü çözünürlükleri için kullanılmasında daha faydalı olabileceğini söyleyebiliriz.

Bu 5 farklı görüntüde sıkıştırma katsayılarını karşılaştırdığımızda genel olarak en yüksek katsayıdan en düşük katsayıya doğru sayarsak sanal, doğa, insan yüzü, nesne, şehir görüntüleri gelmektedir. Buda bize bu sıkıştırma algoritmasının kullanım alanı seçerken bu noktayı göz önünde bulundurabiliriz.

Elde ediğimiz sonuçlara göre en yüksek sıkıştırma katsayısı 3,10'dur. Eğer bu sonuçları daha yükseltilmek istenirse dalgacık dönüşümünün seviyesi ilerletilebilir. Bu

çalışmanın amacı görüntü sıkıştırma işleminin paralel bir şekilde yapılabilmesini göstermektir.

# KAYNAKÇA


[1] H. ZAINELDIN, M. A. ELHOSSEINI ve H. A. ALI, «A modified listless strip based SPIHT for wireless multimedia sensor networks,» *Computers and Electrical Engineering,* 2015.

[2] H. ZAINELDIN, M. A. ELHOSSEINI ve H. A. ALI, «Image compression algorithms in wirelessmultimedia sensor networks: A survey,» %1 içinde *Ain Shams Engineering Journal*, 2015, pp. 481-490.

[3] A. K. AL-JANABI, «Efficient and simple scalable image compression algorithms,» *Ain Shams Engineering Journal,* 2019.

[4] A. AKMAN ve S. ÇEKLİ, «An efficient SPIHT algorithm and system architecture for image compression,» %1 içinde *25th Signal Processing and Communications Applications Conference*, 2017.

[5] S. ÇEKLİ ve A. AKMAN, «Image Compression Performance of HEVC Intra Prediction with Different Transform Coefficient Coding Methods,» %1 içinde *26th IEEE Signal Processing and Communications Applications Conference*, 2018.

[6] B. H. ANG, U. U. SHEIKH ve M. N. MARSONO, «2-D DWT system architecture for image compression,» *Journal of Signal Processing Systems,* cilt 78, no. 131-137, 2013.

[7] C. R. GEETHA, H. BASAVARAJU, R. C. MANJUNATHA, C. P. LATHA ve H. D. GIRIPRAKASH, *Efficient Algorithm for Image,* 2014.

[8] P. H. ANILKUMAR ve P. A. S. BEULET, «Lifting-based Discrete Wavelet Transform for Real-Time Signal Detection,» *Indian Journal of Science and Technology,* pp. 1-6, 2015.

[9] A. SAFARI ve Y. KONG, «The Application of Lifting DWT in Digital Image Processing,» *Lecture Notes in Electrical Engineering,* pp. 449-453, 2013.

[10] P. N. T. AMMAH ve E. OWUSU, «Robust medical image compression based on wavelet transform and vector quantization,» *Informatics in Medicine Unlocked,* pp. 1-11, 2019.

[11] A. K. JAIN, «Fundamentals of digital image processing,» 1989.

[12] E. AKSAN ve S. DOĞAN, «Dalgacık Tabanlı Görüntü Sıkıştırma Tekniği,» *Havacılık ve Uzay Teknolojileri Dergisi,* pp. 47-53, 2004.

[13] A. DERİNOĞLU, *Kablosuz Multimedya Algılayıcı Ağlarda Verimli Görüntü Sıkıştırma Ve İletim Teknikleri,* 2018.

[14] P. A. S. BEULET ve P. H. ANILKUMAR, «Lifting-based Discreate Wavelet for Real-Time Signal Detection,» *Indian Journal of science and technology,* 2015.

[15] V. K. INGLE ve J. G. PROAKIS, Digital signal processing using MATLAB, PWS Publishing Com., 1997.

[16] P. VAIDYANATHAN, «Theory and design of M-channel maximally decimated quadrature mirror filters with arbitrary M, having the perfect-reconstruction property,» %1 içinde *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1987, pp. 476-492.

[17] D. TAUBMAN, «Advanced Modeling and Coding Techniques for Image Compression,» %1 içinde *Document and Image Compression*, 2006, p. 52.

[18] V. ALTINTAŞ ve E. D. YEĞENOĞLU, «Görüntü İşlemede Seri ve Paralel Programlamanın Performansı,» %1 içinde *6th International Advanced Technologies Symposium*, Elazığ, 2011.

[19] M. AKÇAY, B. ŞEN, İ. M. ORAK ve A. ÇELİK, «Paralel Hesaplama ve CUDA,» %1 içinde *6th International Advanced Technologies Symposium*, Elazığ, 2011.

[20] A. GRAMA, A. GUPTA, G. KARYPİS ve V. KUMAR, Introduction to Parallel Computing, Second Edition, 2003.

[21] [Çevrimiçi]. Available: https://gorgiran.com/.

[22] [Çevrimiçi]. Available: http://www.aljanh.net/panorama-of-san-francisco-wallpapers/3453316349.html.

[23] [Çevrimiçi]. Available: https://www.haber365.com.tr/bensu-soral-hakkinda-bilinmeyen-gercekler-g13323.html.

[24] [Çevrimiçi]. Available: http://www.milscint.com/tr/tusasin-anka-s-sihasi-ile-tsk-uydu-kontrolu-ile-vuracak/.

[25] [Çevrimiçi]. Available: http://www.teamopenoffice.org/topk/wallpaper-games-hd/.

[26] K. YAMAN, A. SARUCAN, M. ATAK ve N. AKTÜRK, «Dinamik Çizelgeleme İçin Görüntü İşleme Ve Arıma Modelleri Yardımıyla Veri Hazırlama,» *Gazi Üniv. Müh. Mim. Fak. Der.,* cilt 16, no. 1, pp. 19-40, 2001.