TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YER GÖZLEM UYDU OPERASYONLARI İÇİN JPEG 2000 KOD ÇÖZÜCÜ PERFORMANS OPTİMİZASYONU

YÜKSEK LİSANS TEZİ

Derviş Utku UFUK

Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı Doç. Dr. Ahmet Murat ÖZBAYOĞLU

Nisan 2019

Fen Bilimleri Enstitüsü Onayı

.....
Prof. Dr. Osman EROĞUL
Müdür

Bu tezin Yüksek Lisans derecesinin tüm gereksinimlerini sağladığını onaylarım.

.....
Prof. Dr. Oğuz ERGİN
Anabilimdalı Başkanı

TOBB ETÜ, Fen Bilimleri Enstitüsü'nün 151111030 numaralı Yüksek Lisans Öğrencisi **Derviş Utku UFUK**'un ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"YER GÖZLEM UYDU OPERASYONLARI İÇİN JPEG 2000 KOD ÇÖZÜCÜ PERFORMANS OPTİMİZASYONU"** başlıklı tezi 24.05.2019 tarihinde aşağıda imzaları olan jüri tarafından kabul edilmiştir.

Tez Danışmanı:	**Doç. Dr. Ahmet Murat ÖZBAYOĞLU** TOBB Ekonomi ve Teknoloji Üniversitesi	.....
Jüri Üyeleri:	**Prof. Dr. Oğuz ERGİN (Başkan)** TOBB Ekonomi ve Teknoloji Üniversitesi	.....
	Doç. Dr. Ahmet Murat ÖZBAYOĞLU TOBB Ekonomi ve Teknoloji Üniversitesi	.....
	Prof. Dr. Alptekin TEMİZEL Orta Doğu Teknik Üniversitesi	.....

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, alıntı yapılan kaynaklara eksiksiz atıf yapıldığını, referansların tam olarak belirtildiğini ve ayrıca bu tezin TOBB ETÜ Fen Bilimleri Enstitüsü tez yazım kurallarına uygun olarak hazırlandığını bildiririm.

Derviş Utku UFUK

ÖZET

Yüksek Lisans Tezi

YER GÖZLEM UYDU OPERASYONLARI İÇİN JPEG 2000 KOD ÇÖZÜCÜ PERFORMANS OPTİMİZASYONU

Derviş Utku UFUK

TOBB Ekonomi ve Teknoloji Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı Doç. Dr. Ahmet Murat ÖZBAYOĞLU

Tarih: Nisan 2019

Yüksek sıkıştırma performansı, bit hatalarına karşı dayanımı ve kayıpsız modun yanısıra esnek kayıplı modları desteklemesi nedeniyle JPEG 2000, uydu görüntü işleme, uzaktan algılama ve diğer birçok alanda sıklıkla kullanılan bir görüntü sıkıştırma standardıdır. Kamera ve haberleşme teknolojisinin gelişmesiyle birlikte yer gözlem uydu görüntülerinin çözünürlükleri ve indirme hızları gün geçtikçe artmaktadır. Dolayısıyla JPEG 2000 kod çözme dahil olmak üzere, uydu operasyonu dahilindeki tüm görüntü işleme aşamalarının eş zamanlı olarak hızlandırılması büyük önem arz etmektedir. Literatürde JPEG 2000 kod çözme işleminin GPU'da hızlandırılmasına dair, yer gözlem uydu operasyonlarına doğrudan uyarlanabilecek çalışmalar yetersiz kalmaktadır. Bu çalışmada, öncelikli olarak JPEG 2000 kod çözücünün GPU'da CUDA optimizasyon yöntemleriyle hızlandırılması konu alınmıştır. Buna ek olarak, GPU ve CPU işlemcilerin birlikte tam verimlilikte kullanıldığı bir hibrit kod çözücü tasarımı önerilmiştir. Son olarak, homojen düğümlerden oluşan, esnek ve ölçeklenebilir bir dağıtık JPEG 2000 kod çözücü mimarisi tasarlanmıştır. Bu çalışmanın, uydu görüntü işleme sürecindeki diğer aşamaların hızlandırılmasına yönelik gelecek çalışmalar için faydalı bir referans ve perspektif kazandıracağı öngörülmektedir.

Anahtar Kelimeler: JPEG 2000, Uydu görüntüleri, Görüntü işleme, Görüntü sıkıştırma, Yer gözlem uyduları, GPGPU.

ABSTRACT

Master of Science

PERFORMANCE OPTIMIZATION OF JPEG 2000 DECOMPRESSION FOR GROUND OBSERVATION SATELLITE OPERATIONS

Derviş Utku UFUK

TOBB University of Economics and Technology
Institute of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Assoc. Prof. Dr. Ahmet Murat ÖZBAYOĞLU

Date: April 2019

Due to its high compression performance, bit-error resilience, and support for both lossly and flexible lossy modes, JPEG 2000 is an image compression standard which is widely used in satellite image processing, remote sensing and many other fields. With the recent advancements in camera and telecommunication technology, resolution and download rate of ground observation satellite imagery are continually increasing. This makes it crucially important to simultaneously speed-up all image processing stages in satellite operations, including JPEG 2000 decompression. Existing work in literature regarding the optimization of JPEG 2000 for GPU is insufficient in terms of applicability to ground observation satellite operations. This work primarily focuses on speeding-up the JPEG 2000 decoder for GPU using CUDA optimization techniques. Also a hybrid decoder has been proposed which involves utilization of CPU and GPU together at maximum efficiency. Lastly, a distributed homogeneous JPEG 2000 decoder architecture has been proposed which is highly flexible and scalable. This work is anticipated to provide a useful reference and perspective to future studies regarding the speed-up of other stages in satellite image processing.

Keywords: JPEG 2000, Satellite imagery, Image processing, Image compression, Ground observation satellites, GPGPU.

TEŞEKKÜR

Çalışmalarım boyunca değerli yardım ve katkılarıyla beni yönlendiren hocalarım Doç. Dr. Ahmet Murat ÖZBAYOĞLU ve Prof. Dr. Alptekin TEMİZEL'e, kıymetli deneyimlerinden faydalandığım TOBB Ekonomi ve Teknoloji Üniversitesi Bilgisayar Mühendisliği Bölümü öğretim üyelerine, yüksek lisans öğrenimim boyunca bana burs veren TOBB Ekonomi ve Teknoloji Üniversitesi'ne, fikirleriyle çalışmalarıma katkı sağlayan meslektaşlarım Ömer Berat SEZER ve İbrahim Serdar AÇIKGÖZ'e, destekleriyle her zaman yanımda olan eşim Kübra UFUK'a, kızım Yağmur UFUK'a, aileme ve arkadaşlarıma çok teşekkür ederim.

İÇİNDEKİLER

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

KISALTMALAR

API : Application Programming Interface
ASIC : Application-Specific Integrated Circuit
BAC : Binary Arithmetic Coding
BPC : Bit Plane Coding
CC : Compute Capability
CPU : Central Processing Unit
CUDA : Compute Unified Device Architecture
DWT : Discrete Wavelet Transform
EBCOT : Embedded Block Coding with Optimal Truncation
FPGA : Field-Programmable Gate Array
GPGPU : General-Purpose Graphics Processing Unit
GPU : Graphics Processing Unit
HaaS : Hardware as a Service
IDWT : Inverse Discrete Wavelet Transform
IP : Internet Protocol
JMS : Java Message Service
NVVP : NVIDIA Visual Profiler
SIMT : Single Instruction, Multiple Threads
SM : Streaming Multiprocessor

1. GİRİŞ

Günümüzde kamera teknolojisinin gelişmesiye birlikte ticari, bilimsel ve mühendislik uygulamaları dahilinde üretilen optik görüntülerin mekansal ve spektral çözünürlükleri gün geçtikçe artmaktadır. Veri depolama maliyetinin azalmasıyla ve donanımsal olarak gittikçe daha az yer kaplamasıyla bulutta, sunucularda, iş istasyonlarında ve mobil cihazlarda büyük miktarlarda görüntü verisinin depolanması mümkün hale gelmektedir. Haberleşme teknolojisinin gelişmesiyle birlikte ise bu verilerin kablolu ve kablosuz çeşitli haberleşme yöntemleriyle dağıtımı gitgide yaygınlaşmaktadır. Bütün bunlar büyük miktarlarda görüntü verisinin üretilme, depolanma ve iletilme süreçlerini hızlandıran unsurlar olmakla birlikte; eş zamanlı olarak hızlandırılmadığı takdirde bu görüntülerin işlenmesi, ilgili uygulamalarda doğrudan darboğaz haline gelebilmektedir.

Elbette bir diğer yandan işlemci teknolojilerinin gelişmesi, görüntü işleme uygulamalarında performansı olumlu yönde etkilemektedir. CPU ve özellikle GPU işlemcilerde bulunan çekirdeklerin sayıca artmasıyla birlikte, özellikle piksel seviyesinde veya buna yakın seviyede paralelleştirmeye uygun görüntü işleme algoritmaları önemli oranlarda hızlandırılabilmektedir.

Yüksek performans gerektiren görüntü işleme uygulamalarında istenilen verimlilik seviyelerine ulaşabilmek için, öncelikle her süreç için doğru paralellik modelini seçmek, sonrasında ise gerekli donanım ve yazılım mimarilerini ve doğru optimizasyon yöntemlerini kullanmaktır. Örnek vermek gerekirse, veri paralelliğine ve dolayısıyla SIMT (Single Instruction, Multiple Threads) modeline [3] uyumlu algoritmaları hızlandırabilmek için, gelişmekte olan GPU teknolojisinden faydalanmak ve GPGPU programlama tekniklerini doğru bir şekilde uygulamak çoğu durumda doğru bir seçim olacaktır. Sistem kaynaklarının tam kapasitede kullanılabilmesi için sunucu ve iş istasyonlarındaki CPU ve GPU işlemcilerin bir arada kullanıldığı hibrit yapılar fiyat-performans açısından oldukça verimlidir. Son olarak, uygulanan çözümlerin ölçeklenebilir olması için yerel veya bulut tabanlı dağıtık mimarilerden faydalanmak çoğu durumda önemli performans kazançları sağlayacaktır.

1.1 Problem ve Motivasyon

Gün geçtikçe kamera çözünürlükleri ve haberleşme hızları artan yer gözlem uydu sistemlerinin operasyon verimliliği açısından; görüntü işleme sürecindeki tüm aşamalarının mümkün olduğunca hızlı ve verimli bir şekilde gerçekleştirilmesi büyük önem arz etmektedir. JPEG 2000 yer gözlem uydularından tarafından çekilen görüntülerin sıkıştırılmasında yoğunluklu olarak kullanılan bir standarttır. Sıkıştırılmış olan görüntü verisinin geri kazanılması, uydu görüntü işleme sürecinin önemli aşamalarından bir tanesidir. Bu nedenle tez çalışması kapsamında, yer gözlem uydu operasyonlarına yönelik olarak JPEG 2000 kod çözme algoritmasının hızlandırılması amaçlanmıştır.

2012 yılında fırlatılan milli ve istihbari yer gözlem uydumuz GÖKTÜRK-2'nin operasyon faaliyetleri, Hava Kuvvetleri Komutanlığı'na bağlı olan Keşif Uydu Tabur Komutanlığı'nın Ankara Ahlatlıbel'de bulunan yer istasyonunda yürütülmektedir.[1] Bu yer istasyonunda JPEG 2000 kod çözme aşaması dahil tüm görüntü işleme sürecinde yalnızca CPU'lar kullanılmakta; GPU'lardan herhangi bir şekilde faydalanılmamaktadır. Ayrıca mevcut sistemde görüntüler sıcak yedekli tek bir iş istasyonunda işlenmekte; dağıtık bir mimari bulunmamaktadır. Bu sebeplerden dolayı, görüntü karesi başına JPEG 2000 kod çözme işlemi Çizelge 1.1'de gösterildiği gibi indirme hızının yaklaşık 3 katı kadar zaman almakta, gerçek zamanlı görüntü işleme mümkün olmamaktadır. Üstelik GÖKTÜRK-2 görüntülerinin günümüz için çok yüksek sayılmayacak hızlarda indirilmekte olduğu düşünüldüğünde, problemin ciddiyeti daha belirgin bir şekilde göze çarpmaktadır.[2]

Görüntü çözünürlüğü ve haberleşme hızı GÖKTÜRK-2'den daha yüksek olacak şekilde tasarlanan İMECE ise, henüz üretim aşamasında olan ve 2020 yılında fırlatılması planlanan başka bir milli ve istihbari yer gözlem uydumuzdur. İMECE'nin yer istasyonu görüntü işleme yazılım ve donanımları geliştirme aşamasında olmakla birlikte, Çizelge 1.1'de gösterildiği gibi aşağı yönlü net haberleşme hızının yaklaşık 240 Mbit/s olacağı bilinmektedir.

İdeal operasyon verimliliğine ulaşmak için, görüntü işleme hıznın indirme hızı ile eşit veya daha iyi olması gerekmektedir. Çizelge 1.1'e göre GÖKTÜRK-2 için 50 MB büyüklüğünde sıkıştırılmış bir kare görüntünün indirilmesi 4 saniye sürmekte; aynı görüntünün JPEG 2000 kod çözme süreci ise yaklaşık 12 saniye sürmektedir. Dolayısıyla indirilen her 3 birim görüntü için aynı sürede yalnızca 1 birim görüntü işlenebilmektedir. İMECE'de ise GÖKTÜRK-2'deki mevcut görüntü işleme sürecinin

[1] https://www.hvkk.tsk.tr/Havacılık_Köşesi/Özel_Siteler/Keşif_Uydu_Komutanlığı/Uydularımız/GÖKTÜRK_2
[2] http://uzay.tubitak.gov.tr/tr/projeler/gokturk-2

Çizelge 1.1: GÖKTÜRK-2 ve İMECE için birim görüntü indirme ve işleme hızları

	GÖKTÜRK-2	**İMECE**
Sıkıştırılmış Kare Boyutu	50 MB	50 MB
İndirme Hızı	100 Mbit/s	240 Mbit/s
İndirme Süresi	4 saniye	1.7 saniye
JPEG 2000 İşleme Süresi	12 saniye	?

iyileştirilmediği varsayılacak olursa 1.7 saniyede indirilen birim görüntünün işlenmesi yine 12 saniye sürecek ve operasyon verimliliği GÖKTÜRK-2'ye kıyasla çok daha ciddi biçimde zarar görecektir.

Tez çalışmasında bu problemden yola çıkılarak uydu görüntü işleme süreci dahilinde en çok zaman alan aşamalardan biri olan JPEG 2000 kod çözme algoritmasının hızlandırılması konu alınarak, İMECE gibi gelecekte üretilebilecek yüksek çözünürlük ve haberleşme kapasitesine sahip uydularda karşılaşılması muhtemel olan bu tip verimlilik kayıplarını önlemek hedeflenmiştir.

1.2 JPEG 2000'in Hızlandırılmasına Dair Literatürdeki Çalışmalar

Günümüzde GPU teknolojisi paralel veri işleme performansı bakımından CPU teknolojisine kıyasla daha hızlı gelişme göstermektedir ve bu durum görüntü işleme algoritmalarının hızlandırılması hususunda oldukça fayda sağlamaktadır. Bununla birlikte, yakın maliyetli bir CPU'ya oranla daha yüksek performanslar elde etmek mümkün olsa da, birçok görüntü işleme algoritmasında GPU ile sağlanabilen yüksek performans artışları, SIMT seviyesi paralelliğe tam anlamıyla uyumlu olmamasından dolayı JPEG 2000 söz konusu olduğunda yakalanamamaktadır. Bu durum JPEG 2000 kod çözme aşamasını, yer gözlem uydu operasyonlarının görüntü işleme sürecindeki başlıca performans darboğazlarından biri haline getirebilmektedir.

JPEG 2000'in optimizasyonu konusunda literatürde yapılmış olan çalışmaların sayısı birkaç düzine ile sınırlıdır. GPGPU programlamanın henüz yeterince gelişmediği ve popülerlik kazanmadığı dönemlerde, bu problemin FPGA ve ASIC mimarileri kullanılarak çözülmeye çalışıldığı görülmektedir [4], [5], [6], [7].

Daha sonraki dönemlerde GPU odaklı mimariler polülerleşmeye başladıysa da, güncel çalışmaların birçoğunda sıkıştırma prosedürü ele alınmış; ancak uydu operasyonlarının önemli bir parçası olan kod çözme prosedürüne değinilmemiştir [8], [9], [10].

Üstelik, bazı çalışmalarda SIMT seviyesi paralellik elde ederek algoritma verimliliğinin artırılabilmesi için standarttan uzaklaşılmış ve uyumluluk kırılmıştır [11], [12], [13]. Bu yaklaşım, uydu operasyonları gibi sıkıştırma modülünde değişiklik yapmanın mümkün olmadığı durumlara karşı herhangi bir çözüm sunamamaktadır.

Özellikle JPEG 2000 kod çözme prosedürünü konu alan ve aynı zamanda standarda sadık kalan çalışmaların sayısı oldukça azdır. Bunlar arasında CPU ve GPU'nun birlikte kullanıldığı hibrit mimarilerin önerildiği çalışmalar bulunsa da [14], büyük bir kısmında yalnızca GPU paralelliğine dayalı mimariler önerilmiştir [15], [16], [17].

1.3 Tezin Katkıları

İlgili alanda uydu görüntülerinin işlenme sürecine doğrudan uyarlanabilecek literatür çalışmalarının yetersiz olmasından yola çıkarak, bu tez çalışmasında çeşitli yer istasyonu işlemci ve mimari konfigürasyonları göz önünde bulundurulmuş ve JPEG 2000 kod çözme sürecinin hızlandırılması için çeşitli yaklaşımlar önerilmiştir. Bu süreci teorik olarak darboğaz olmaktan çıkarabilecek bu yaklaşımların, uydu görüntülerinin yer istasyonuna indirilmesi esnasında gerçek zamanlı olarak işlenme hedefi doğrultusunda önemli bir adım olacağı öngörülmektedir. Tez kapsamında yapılan çalışmaları literatürdeki mevcut çalışmalardan ayıran en önemli faktörler şöyle sıralanabilir:

1. JPEG 2000 standardına sadık kalınmış ve uydu görüntüleri için özellikle kod çözme prosedürünün GPU ile hızlandırılmasına odaklanılmıştır.

2. Uydu görüntü işleme süreci için özelleştirilmiş ve otomatik yük dengeleme kabiliyetine sahip bir GPU-CPU hibrit paralel JPEG 2000 kod çözücü tasarımı önerilmiş ve bu sayede görüntü işleme sunucularında bulunabilecek çeşitli CPU ve GPU işlemcilerin tam verimlilikle kullanmanın önü açılmıştır.

3. Çok sayıda görüntü işleme düğümünün bulunabileceği yer istasyonları göz önünde bulundurularak, genişletilebilir ve esnek bir dağıtık kod çözme mimarisi önerilmiştir.

Tez kapsamında performans değerlendirmesinde her ne kadar Çizelge 1.1'de gösterilen İMECE görüntü indirme hızı referans alınmış olsa da, önerilen her bir yöntem ve yaklaşım gelecekte daha yüksek indirme hızları için ölçeklenebilir olarak tasarlanmıştır. İlgili bölümlerde sözkonusu yöntem ve yaklaşımların daha yüksek performans gerektiren durumlarda nasıl ölçeklendirilebileceği detaylı olarak irdelenmektedir.

Tez kapsamında yapılan çalışmalar şu şekilde organize edilmiştir: Bölüm 2'te yer gözlem uydu operasyonları, uydu görüntü işleme süreci ve yer istasyonlarında kullanılan donanım ve mimariler hakkında bilgi verilmektedir. Bölüm 3'de paralel ve dağıtık görüntü işleme yaklaşımları genel olarak özetlenmekte ve NVIDIA GPU'lar özelinde donanım ve CUDA programlama modelleri hakkında bilgi verilmektedir. Bölüm 4'te JPEG 2000 standardını oluşturan temel algoritmalar tanıtılmakta ve kod çözme süreci detaylı bir şekilde anlatılmaktadır. Bölüm 5'te JPEG 2000 kod çözme algoritması için önerilen GPU optimizasyon yöntemleri detaylı bir şekilde açıklanmakta ve elde edilen deney sonuçları üzerinden her bir yöntemin perfromans üzerindeki etkisi tartışılmaktadır. Bölüm 6'da CPU ve GPU'nun eşzamanlı olarak tam verimlilikte kullanıldığı ve otomatik yük dengeleme yeteneğine sahip hibrit-paralel bir JPEG 2000 kod çözücü tasarımı önerilmektedir. Bölüm 7'de homojen düğümlerden oluşan bir yer istasyonu sistemi için dağıtık bir JPEG 2000 kod çözücü mimarisi önerilmektedir. Bölüm 8'de yapılan çalışmalar ve önerilen yöntemler özetlenmekte ve gelecekte konu hakkında yapılabilecek çalışmalardan bahsedilmektedir.

2. UYDU GÖRÜNTÜ İŞLEME SÜRECİNE GENEL BAKIŞ

Bu bölümde, tez çalışmasında önerlien yöntem ve modellerin uygulanması hedeflenen öncelikli alan olan yer gözlem uydu operasyonları konu alınmaktadır. Önerilen modellerin ve yöntemlerin altında yatan gerekçelerin ve motivasyonun daha iyi kavranabilmesi için, öncelikle yer gözlem uydu operasyonlarına dair bir takım önemli noktaların anlaşılması gerekmektedir. İlerleyen kısımlarda yer gözlem uydu operasyon süreci, uydu görüntülerinin karaktersitiği ve bu görüntülerin işlendiği yer istasyonlarında kullanılmakta olan görüntü işleme donanım ve mimarileri anlatılmaktadır.

2.1 Yer Gözlem Uydu Operasyon Süreci

Yer gözlem uydu görüntülerinin ürün haline gelmesi, uydu ve yer istasyonundaki birçok alt sistemin rol oynadığı, bir çok aşamadan oluşan ve karmaşık bir süreçtir. Bu süreci oluşturan ana aşamalar aşağıdaki gibi gruplanabilir:

1. Görüntüleme Talebi Oluşturma
2. Görüntüleme ve İndirme Planları Oluşturma
3. Görüntü Çekimi ve Depolama
4. Görüntü İndirme ve İşleme

Uydu görüntülerinin işlenmesine dair daha iyi bir perspektif sunabilmek için, ilerleyen kısımlarda her bir adım kısaca özetlenmektedir.

2.1.1 Görüntüleme Talebi Oluşturma

Görüntüleme talepleri, uydunun istihbari olup olmamasına göre farklı şekillerde oluşturulabilmektedir. Sivil uydu operasyonlarında, çeşitli kurumların ve/veya şahısların farklı öncelik seviyeleriyle uzaktan talep oluşturmasına izin verilebilmektedir. Askeri uydularda ise talepler dışarıya açık olmayıp, yetkili birim tarafından girilmektedir. Bazı

durumlarda ise talep oluşturma aşaması tamamen atlanarak, görüntüleme görevleri doğrudan uydu operatörleri tarafından planlanabilmektedir.

Çok sayıda talep çeşidi olmakla birlikte, kare ve şerit görüntü talepleri en sık karşılaşılanlar arasındadır [18]. Kare görüntü talebi, spesifik bir bölge çekilmek istendiği zaman oluşturulur ve sonucunda kare veya kareye çok yakın boyutlarda bir görüntü elde edilir. Şerit görüntü talebi ise, genellikle veritabanlarını güncel tutma veya tarama yapma amacıyla oluşturulur ve sonucunda kare görüntüyle aynı genişlikte ancak daha uzun bir görüntü elde edilir. Çoğu durumda şerit görüntüler, birden fazla kare görüntünün kesintisiz olarak arka arkaya eklenmiş haline benzemektedir.

2.1.2 Görüntüleme ve İndirme Planları Oluşturma

Komuta yetkisine sahip olan yer istasyonlarının görev planlama alt sistemi tarafından, periyodik olarak mevcut talepler yörünge, açı ve öncelik vb. kriterlere göre değerlendirmeye alınarak kısa vadeli (örneğin 24 veya 48 saat) görüntüleme ve indirme planları yapılmaktadır. Görüntüleme planı uydunun ne zaman nereyi görüntüleyeceğini; indirme planı ise çekilmiş bir görüntüyü ne zaman hangi yer istasyonuna indireceğini belirlemektedir. Görev planlama alt sistemi tarafından oluşturulan bu planlar, uydunun yer istasyonu üzerinden geçişi esnasında uzkomutlar halinde uyduya yüklenmektedir.

2.1.3 Görüntü Çekimi ve Depolama

Uydu alt sistemlerini yöneten uçuş bilgisayarı, yer istasyonundan gelen görüntülemeye dair uzkomutları, zamanı geldiğinde işletmek üzere faydalı yük alt sistemine iletir. Görüntüleme sonucunda elde edilen faydalı yük verisi uydu veri depolama birimine kaydedilir. Çoğu durumda, aşağı yönlü haberleşme verimliliğini ve güvenliğini artırma amacıyla kameradan gelen faydalı yük verisi sıkıştırlıp şifrelendikten sonra depolanmaktadır.

2.1.4 Görüntü İndirme ve İşleme

Uyduya uzkomutlar aracılığıyla yüklenen indirme planı doğrultusunda, uyduda depolanmış olan faydalı yük verisi zamanı gelince yer istasyonuna indirilir. Özellikle istihbari uydularda, indirilen görüntünün en kısa zamanda işlenmesi oldukça önemlidir. İndirilen ham görüntülerin kullanılabilir hale getirilmesi için; aşağıda genel adımları listelenen görüntü seviyelendirme aşamalarından geçmesi gerekmektedir:

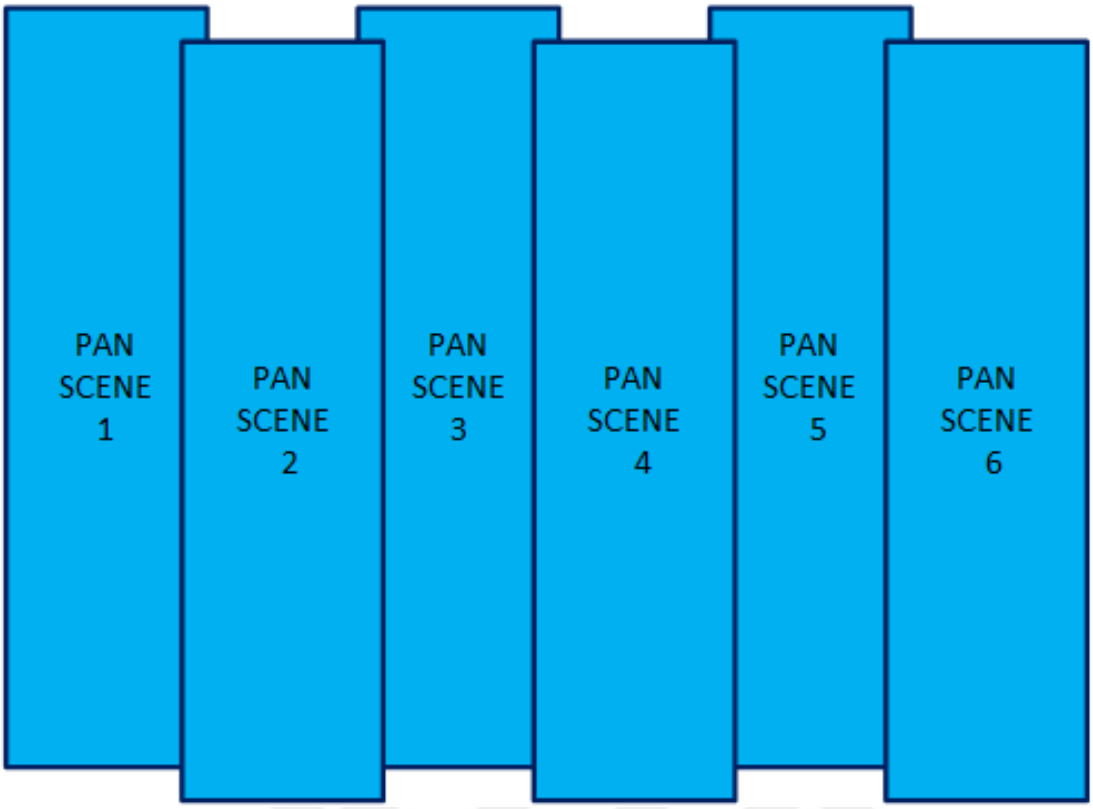
- **Seviye 0**
  - Şifre çözme
  - Sıkıştırma açma
- **Seviye 1**
  - Gürültü giderme
  - Bağıl radyometrik düzeltme
  - Keskinleştirme
- **Seviye 2**
  - Görüntü birleştirme
  - Geometrik düzeltme
- **Seviye 3**
  - Ortorektifikasyon

2.2 Uydu Görüntülerinin Bölünmesi

Uydu tarafından çekilen bir görüntü indirilirken meydana gelebilecek haberleşme hatalarında, görüntünün mümkün olduğunca büyük bir kısmının kurtarılabilir olması istenmektedir. Dolayısıyla çoğu yer gözlem uydusunda görüntüler yekpare halde değil, birçok parçaya bölünmüş olarak yer istasyonuna iletilir.

Uydu kameraları çoğunlukla ayrık sensörlerden meydana gelmekte [19],[20],[21] ve bu sensörler tarafından üretilen görüntüler birbirinden bağımsız olarak yer istasyonuna indirilmektedir. Buna ek olarak, birden fazla spektral bant içeren görüntülerin her bir bandı çoğunlukla ayrı birer gri seviyeli görüntü olarak yer istasyonuna indirilmektedir [18]. Örneğin, 6 adet sensörden oluşan bir kamera tarafından çekilen 18×18 kilometrelik tipik bir kare uydu görüntüsü, Şekil 2.1'te bir tanesi görselleştirilmiş olan 5 adet spektral banttan (pankromatik, kırmızı, yeşil, mavi ve kızılötesi), her bir bantta ise 3'er kilometrelik sensör verisinden oluşabilmektedir. Bu durumda ortaya çıkan görüntü parçalarının her biri 3×18 kilometrelik şeritlere denk gelen gri seviyeli birer görüntü olacaktır.

Şerit görüntüler ise renk bantları ve sensör çıktılarının yanısıra, düşey düzlemde eşdeğer kare sayısına göre ayrıca bölünmekte; sonuç olarak N adet kare görüntü formatında yer istasyonuna indirilmektedir.



Şekil 2.1: Örnek bir kare uydu görüntüsünü oluşturan spektral bantlardan birinin sensör bazında daha küçük parçalara bölünmesi

2.3 Uydu Yer İstasyonlarında Kullanılan Görüntü İşleme Donanımları

Yer gözlem uydu operasyonlarının yürütüldüğü yer istasyonlarında, görüntü işleme operasyonları yoğunlukla CPU ve GPU işlemciler kullanılarak yapılmaktadır. Her ne kadar FPGA ve ASIC mimarileri görüntü işleme ve derin öğrenme gibi paralelleştirmeye uygun algoritmalar özelinde daha yüksek performanslar ve düşük güç tüketimi sağlasa da [22] [23], [24]; aşağıda listelenen başlıca sebeplerden dolayı bu mimariler yer istasyonlarında görüntü işleme amacıyla kullanılmamakta, CPU ve GPU'lar tercih edilmektedir:

- **Genel Kullanılırlık:** FPGA ve özellikle ASIC mimarileri, her uygulama için özel olarak tasarlanmakta ve üretilmektedir. Dolayısıyla uydu görüntü işleme sürecinde bu mimarilerin tercih edilmesi halinde, Bölüm 2.1.4'de bahsi geçen tüm algoritmalar için ayrı ayrı özelleştirilmiş donanımlar tasarlanması gerekecektir. Diğer taraftan görüntü işleme dahil tüm yer kesimi yazılımlarında kullanılabilen genel amaçlı CPU ve GPU'ların tüm yer istasyonunda kullanılması sayesinde, fazladan donanım tasarım ve tedarik maliyetlerinden kaçınılabilmektedir.

- **Düşük Geliştirme Süresi ve Maliyeti:** Bir görüntü işleme algoritması için CPU ve GPU geliştirme süreleri ve maliyetleri, FPGA ve ASIC'e göre önemli ölçüde düşüktür. Bölüm 3.2'de değinildiği gibi, CUDA sayesinde GPGPU programlama süreci oldukça kolaylaşmış ve C++ gibi yüksek seviye bir dilde bilgisayar yazılımı geliştirmeyle yaklaşık olarak aynı seviyeye yükselmiştir.

- **Düşük Modifikasyon ve Hata Ayıklama Süresi ve Maliyeti:** Uydu görüntü işleme süreci birçok karmaşık algoritmalardan meydana gelmektedir. Yüksek hesapsal karmaşıklığa sahip olan JPEG 2000 ve uydunun yörünge ve yönelim verilerinin görüntü verisiyle birlikte kullanıldığı yüksek mühendislik bilgisi gerektiren geometrik düzeltme gibi algoritmalar buna örnek olarak gösterilebilir. Özellikle uydu görev ömrünün ilk aylarında ilgili yazılımların güncellenmesi ve olası hataların ayıklanması için yoğunlukla ve sıklıkla çalışılmaktadır. CPU ve GPU mimarilerde bu tarz güncelleme ve hata ayıklama çalışmaları, FPGA ve ASIC mimarilerine kıyasla çok daha düşük maliyetle yapılabilmektedir.

Bir sonraki bölümde, görüntü işleme algoritmalarının hızlandırılmasına yönelik literatürdeki paralel ve dağıtık mimarilere dair çalışmalar ve yaklaşımlar konu alınmaktadır.

3. PARALEL VE DAĞITIK GÖRÜNTÜ İŞLEME

Tezin ana konusu olan JPEG 2000 kod çözücü dahil olmak üzere, Bölüm 2.1.4'te bahsi geçen tüm uydu görüntü işleme aşamalarının hızlandırılması, yer gözlem uydu operasyon verimliliği açısından büyük önem arz etmektedir. Bu bölümde, yer gözlem uydu operasyonlarının görüntü işleme sürecine dahil olan algoritmalarının hızlandırılmasında fayda sağlayabilecek olan ve aynı zamanda genel olarak görüntü işleme literatüründe yaygın olarak kullanılan paralel ve dağıtık mimarilerin ve yaklaşımların bazılarından bahsedilmektedir.

Tezde önerilen çözümlerin dayalı olduğu GPGPU programlama ve CUDA hakkında ise bu bölümün devamında daha detaylı bilgi verilmekte, NVIDIA GPU'ların donanım modeli ve CUDA programlama modeli anlatılmaktadır.

3.1 Genel Yaklaşımlar

CPU teknolojisinde hızlanmanın gitgide azaldığı günümüzde, büyük verideki patlamanın da etkisiyle görüntü işleme uygulamalarının neredeyse tamamında SIMT seviyesinde paralel mimarilerden ve/veya dağıtık mimarilerden faydalanılmaktadır. Her iki yaklaşımın da kendine göre avantaj ve dezavantajları bulunmaktadır. Tasarım kararı olarak hangi yaklaşımın benimseneceği, uygulamalarnın ve projelerin spesifik karakteristiğine, gereksinimlerine, altyapı imkanlarına ve bütçe kısıtlarına göre belirlenmelidir.

3.1.1 Paralel Görüntü İşleme

Görüntü işleme uygulamalarında paralellik elde etmeye yönelik olarak kullanılan yaklaşımlar arasında en yaygın olanlar aşağıda özetlenmektedir [25]:

1. **Veri Paralellği:** Veri paralelliği, görüntünün parçalara bölünüp çok sayıda işlemci birimlere dağıtılmasına dayanmaktadır. Uygulamadan uygulamaya değişiklik göstermekle birlikte, genellikle görüntüler piksel, satır, sütun veya blok bazında bölünmektedir. Buradaki en kritik noktalardan biri, homojen olmayabilecek işlemci

birimleri için yük dengeleme stratejisinin iyi belirlenmesi gerekliliğidir. Yük dengelemenin optimize edilmemesi, daha yüksek hızlı olan işlemcilerin boşta beklemesi sonucunda verimlilikte kayıplara neden olabilmektedir.

2. **İş Paralelliği:** İş paralelliği, uygulamanın birbirinden bağımsız görüntü işleme operasyonları içermesi durumunda faydalanılabilecek bir yaklaşımdır. Kendi çıktısı başka bir operasyonun çıktısına bağımlı olmayan görüntü işleme operasyonları; farklı işlemci birimlere atanarak sistemdeki kaynaklar bu yaklaşım sayesinde verimli bir şekilde kullanılabilmektedir.

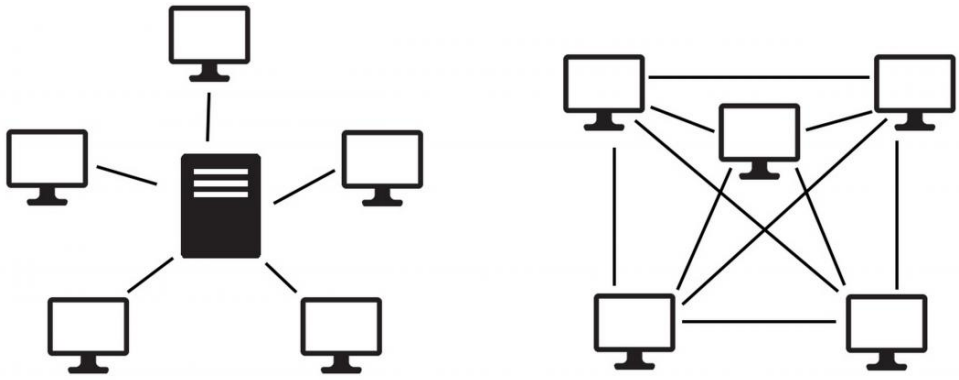
3. **Boru Hattı Paralelliği:** Bazı görüntü işleme algoritmaları çıktıları birbirine bağımlı olan birden fazla adımda tamamlanmaktadır. Boru hattı paralelliği, birden fazla görüntünün işlenmesi gerektiği durumlarda böyle algoritmalar için kullanılabilecek bir yaklaşımdır. Buna göre her bir görüntü işleme aşaması ayrı bir işlemci birime atanır. Boru hattı sayesinde ise her bir görüntünün belirli bir zaman noktasında farklı aşamalarda olması sağlanarak paralellik gerçekleştirilmiş olur.

3.1.2 Dağıtık Görüntü İşleme

Paralel görüntü işlemeye benzer şekilde, dağıtık görüntü işlemede uygulanabilecek yaklaşımlar da çeşitlilik göstermektedir. Mimari olarak bunlardan en yaygın olan iki yaklaşım [25] aşağıda özetlenmiştir ve topolojiler arasındaki fark Şekil 3.1'de gösterilmiştir.

1. **Efendi-Köle (Master-Slave) Mimarisi:** Efendi-köle mimarisi, merkezi (efendi) düğüm olan bir sunucu veya iş istasyonu tarafından iş akışının planlanmasına ve yine efendi düğüm tarafından verinin köle düğümlere dağıtılmasına ve çıktıların toplanmasına dayanmaktadır. Bu yaklaşıma göre yük dengelemesi, veri transferi ve iş akışı tamamen merkezi bilgisayarın sorumluluğundadır. Köle bilgisayarların görevi ise, kendilerine atanan veriyi işleyip sonucu merkezi bilgisayar geri döndürmektir.

2. **Eşler Arası (Peer-to-Peer) Mimari:** Efendi-köle mimarisinin aksine, bu yaklaşımda süreci koordine eden tek bir merkezi bilgisayar bulunmamaktadır. Bunun yerine, ağdaki her bir bilgisayar birbirine bağlıdır ve her biri aynı yetkinlik seviyesine ve fonksiyonaliteye sahiptir. Dolayısıyla görüntü işleme süreci bu bilgisayarlardan herhangi biri tarfından başlatılabilmekte ve görüntülerin işlenmesi bu bilgisayarların tümünde gerçekleştirilmektedir.



Şekil 3.1: (a) Efendi-köle mimarisi (b) Eşler arası mimari

Yerel çözümlerin yanısıra, günümüzde bulut üzerinden dağıtık görüntü işleme uygulamaları gitgide yaygınlaşmaktadır. Amazon Elastic Compute Cloud (EC2)[3], Google Compute Engine[4] ve Microsoft Azure[5] bu alanda dünyadaki en yaygın HaaS (hardware as a service) altyapıları arasındadır.

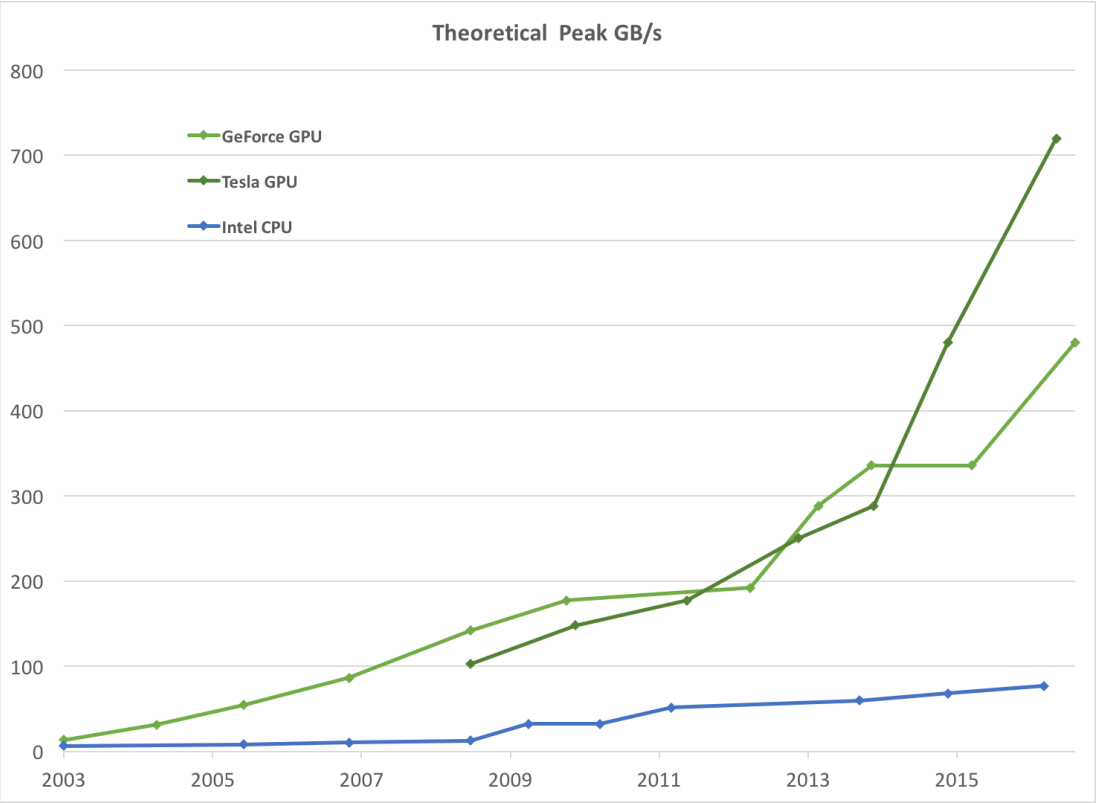
Yaklaşık sabit ve önceden kestirilebilir büyüklükte görüntü işleme taleplerinin bulunduğu ve bilgi güvenliği kısıtları içeren uygulamalar için lokal çözümler tercih edilirken; ölçeklemeye açık ve işleme taleplerinin değişkenlik gösterebileceği uygulamalar için bulut işleme sistemleri özellikle ekonomik açıdan oldukça avantajlıdır [26].

Görüntü işleme taleplerinin büyüklük olarak değişken olmasına örnek olarak uydu görünütü işleme operasyonları gösterilebilir. Normal operasyonda günlük yaklaşık olarak işlenecek görüntü miktarı uydunun yörünge periyoduna bağlıdır ve çok fazla değişkenlik göstermemektedir [18]. Bununla birlikte, görüntü işleme algoritmalarından birinde değişiklik veya iyileştirme olması halinde, geçmişe yönelik arşiv görüntülerinin de toplu olarak tekrar işlenme gerekliliği ortaya çıkabilmektedir. Bu ve bunun gibi ender durumlar için çok büyük bir altyapı kaynağına yatırım yapmak ekonomik olarak verimsiz olacağı için, lokal altyapı çözümlerine gidilmesi böyle uç senaryolarda işleme sürelerini talep büyüklüğüyle doğru orantılı olarak artıracaktır. Bu noktada bulut tabanlı bir sistemin, görüntü işleme taleplerinin büyüklüğünden bağımsız olarak işleme sürelerini sabitlemeye olanak sağlamasının yanısıra, anlık kaynak ölçekleme kolaylığı ve kullanımla doğru orantılı ödeme imkanı sayesinde ekonomik olarak da çok daha optimal bir çözüm haline geleceği açıktır.

[3] https://aws.amazon.com/ec2/
[4] https://cloud.google.com/compute/
[5] https://azure.microsoft.com/en-us/

3.2 GPGPU Programlama ve CUDA

GPGPU programlama, Bölüm 3.1.1'de bahsedilen veri paralelliğini elde etmeye yönelik günümüzde gittikçe yaygınlaşan bir yaklaşımdır. CUDA 2006 yılında NVIDIA tarafından sunulmuş olan bir GPGPU programlama modeli ve platformu olup, birçok hesapsal problemi CPU'dan çok daha verimli bir şekilde çözme olanağı sağlamaktadır. Şekil 3.3 ve Şekil 3.2'de Intel CPU'lar ile NVIDIA GPU'ların yıllara göre veri işleme hacmi ve bellek transfer hızları arasındaki fark belirgin bir şekilde görülmektedir.

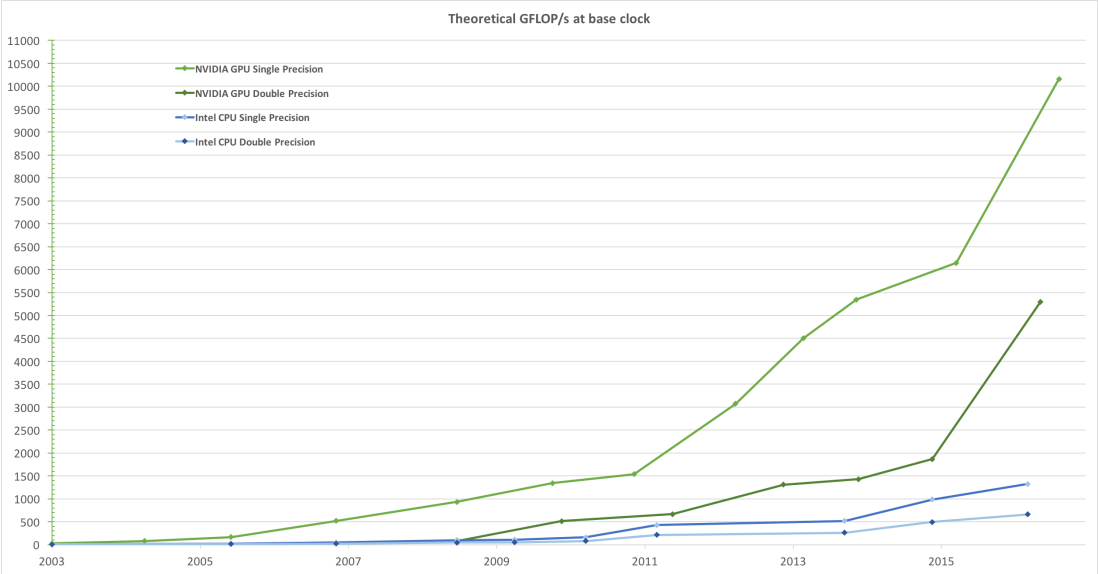


Şekil 3.2: CPU ve GPU için bellek transfer hızları [1]

İlerleyen kısımlarda, NVIDIA GPU thread ve bellek hiyerarşileri ve donanım modeli hakkında bilgi verilmektedir. Buna ek olarak bir makinede birden fazla GPU kartının birlikte kullanımından ve son olarak GPU verimliliğini etkileyen bir takım kriterlerden bahsedilmektedir.

3.2.1 Thread Hiyerarşisi

CUDA, geliştiricilere C/C++ fonksiyonları biçiminde kerneller tanımlamaya izin vermektedir. Bir CUDA kerneli, GPU'da N adet thread tarafından toplam N defa çalıştırılacak bir fonksiyonu ifade etmektedir. NVIDIA GPU'larda *warp* adı verilen 32'lik thread



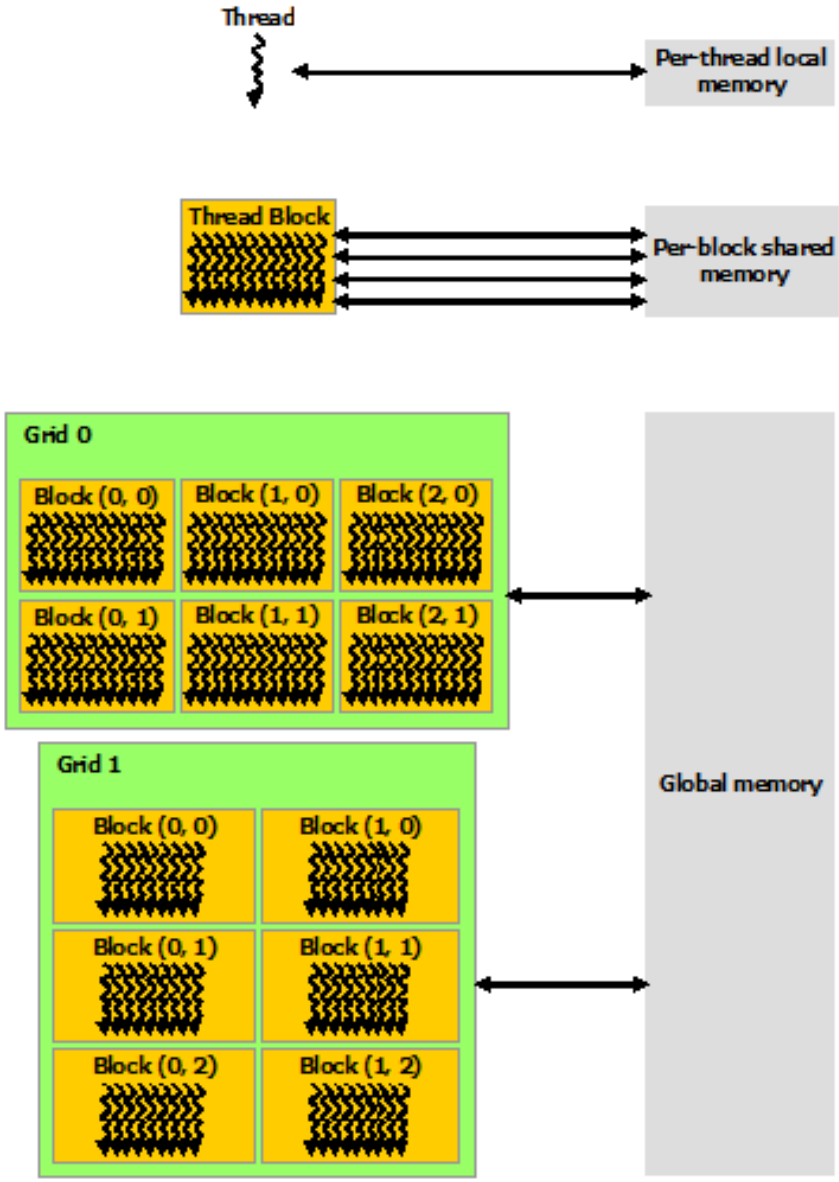
Şekil 3.3: CPU ve GPU için saniye başına kayan nokta operasyon hızları [1]

grupları, aynı anda aynı buyruğu çalıştıracak şekilde tasarlanmıştır. Kernel programlamada thread'ler bir üst seviyede *thread blokları* halinde gruplanır ve bir kernel konfigürasyonu sonucunda oluşan thread bloklarının tamamına *grid* adı verilir. Geliştiriciler kernelleri blok başına düşen thread sayısı ve grid başına düşen blok sayısını konfigüre ederek, verinin hangi seviyede paralelleştirileceğini tam anlamıyla kontrol edebilmektedir.

3.2.2 Bellek Hiyerarşisi

Şekil 3.4'de gösterildiği gibi [1], CUDA thread'leri kernel çalışması esnasında çeşitli bellek alanlarından veri transferi gerçekleştirebilir.

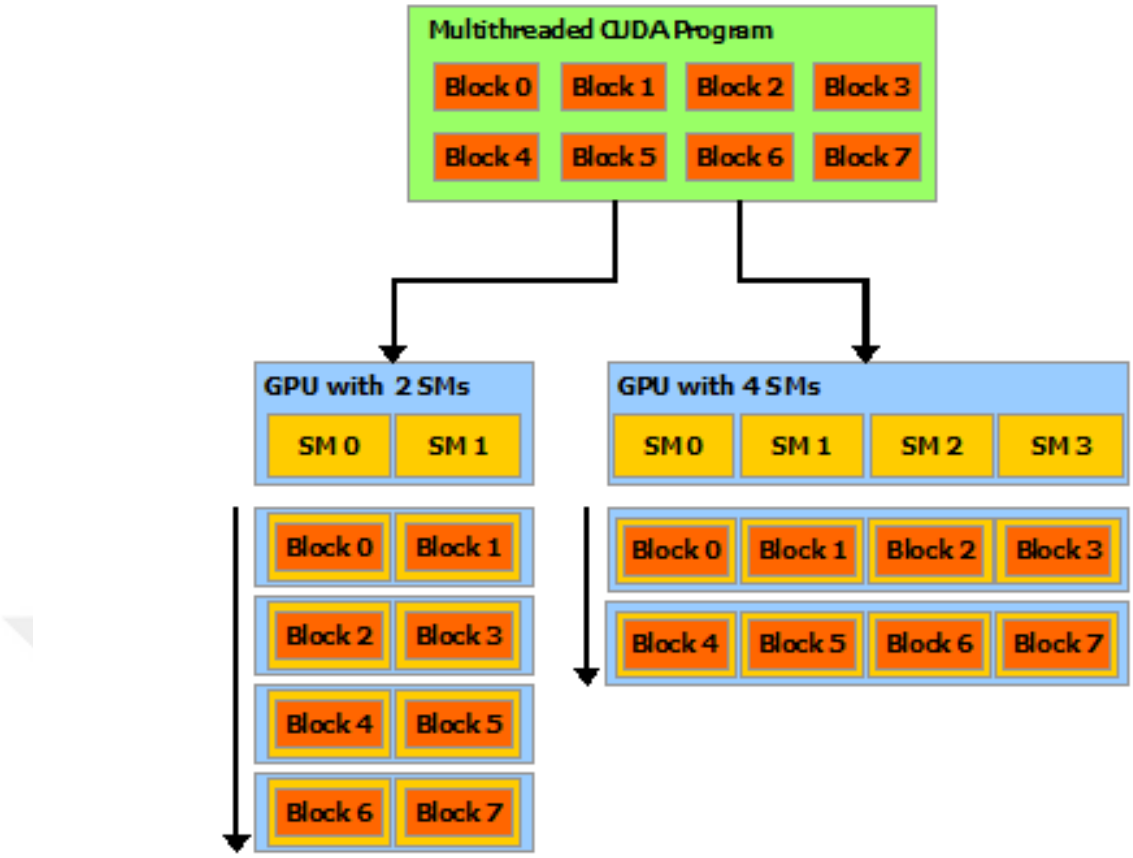
- **Lokal Bellek:** Her bir thread'in yanlızca kendi erişebildiği çok yüksek hızlı bir lokal belleği vardır.
- **Paylaşımlı Bellek:** Her bir thread bloğunun, blok içindeki tüm thread'ler tarafından erişilebilen ve yaşam döngüsü thread bloğuna bağlı olan bir paylaşımlı belleği vardır. Erişim hızı neredeyse lokal bellek ile aynıdır.
- **Global Bellek:** Global bellek, yaşam döngüsü kernellerden bağımsız olan, tüm thread'ler tarafından erişilebilir olan; ancak transfer hızı lokal ve paylaşımlı belleğe kıyasla çok daha düşük olan bir bellek alanıdır. Diğer iki bellek çeşidine kıyasla hacimce oldukça büyük olduğundan, veri çoğunlukla bu alanda tutulmakta ve ihtiyaç oldukça lokal ve paylaşımlı belleğe geçici olarak kopyalanmaktadır.



Şekil 3.4: CUDA bellek hiyerarşisi [1]

3.2.3 Donanım Modeli

NVIDIA GPU mimarisi, her bir cihaz için sayıca değişkenlik göstermek üzere, streaming multiprocessor (SM) adı verilen ve SIMT mimarisine sahip bir dizi çoklu işlemciden meydana gelmektedir. CUDA sayesinde spesifik bir GPU'dan bağımsız olarak otomatik ölçeklenebilir GPGPU uygulamaları geliştirmek mümkündür. Geliştiriciler eldeki paralel problemi blok ve thread'ler seviyesinde modeller ve Şekil 3.5'de gösterildiği gibi CUDA bu modelin mevcut GPU'daki SM'lere otomatik olarak dağıtılmasını sağlar.



Şekil 3.5: CUDA donanım modeli [1]

3.2.4 Çoklu GPU Uygulamaları

CUDA, bir makinede birden fazla NVIDIA GPU'nun birlikte kullanılmasına izin vermektedir. Böyle bir senaryoda, bellek tahsisleri ve kernel'ler istenilen cihazda çalışacak şekilde konfigüre edilebilmektedir. Böylelikle birçok uygulama dağıtık mimarilere gerek kalmadan tek bir sunucu veya iş istasyonunda performans anlamında ölçeklenebilmektedir.

Aynı zamanda CUDA, bir host'ta bulunan birden fazla GPU arasında bellek transferi yapmaya izin vermektedir. Üstelik, 64 bit uygulamalarda CC (compute capability) 2.0 ve üzeri Tesla serisi GPU'ların birlikte kullanılması durumunda bu cihazlar birbirlerine ait fiziksel belleklere doğrudan erişebilmektedir. Bu sayede çoklu GPU uygulamalarında cihazlar arası veri transferi oldukça esnek bir şekilde gerçekleştirilebilmektedir.

3.2.5 GPU Paralelliğinde Verimlilik Kriterleri

Bir görüntü işleme algoritmasının, GPU paralelliği aracılığıyla hızlandırılmaya ne kadar uygun olduğu aşağıdaki kriterlere göre değerlendirilebilir [27]:

- **Paralelleştirilebilir Oran:** Çoklu işlemci kullanarak elde edilebilecek teorik hızlanma Amdahl Yasası kullanılarak tahmin edilebilmektedir. Buna göre f'nin bir programın paralelleştirmeye açık ve $(1-f)$'nin paralelleştirmeye açık olmayan fraksiyonu olması halinde, N adet işlemci kullanılarak elde edilebilecek maksimum hızlanma S Denklem 3.1'e göre hesaplanabilir.

$$S \leq \frac{1}{1-f+\frac{f}{N}} \tag{3.1}$$

- **Kayar Nokta İşlemlerinin Bellek Erişimine Oranı:** Bellek erişimlerinden kaynaklanan gecikmeleri perdeleyebilmek açısından, bu oran ne kadar yüksekse SM verimliliği ve dolayısıyla işleme performansı o kadar yüksek olacaktır.

- **Piksel Başına Düşen Kayan Nokta Operasyonu ve Bellek Erişimi:** GPU'nun CPU'ya oranla daha yüksek performansa sahip olmasının başlıca nedeni, Şekil 3.3 ve Şekil 3.2'de görülebileceği gibi kayar nokta operasyonlarında ve bellek transferlerinde çok yüksek verimliliğe sahip olmasıdır. Dolayısıyla piksel başına düşen kayar nokta işlemlerinin ve bellek erişimlerinin yoğunluklu olduğu algoritmalar doğal olarak CPU'ya oranla daha fazla hızlandırılabilecektir.

- **Dallanma Çeşitliliği:** Warp içinde bulunan her bir thread SIMT mimarisi gereği aynı anda aynı buyruğu çalıştırmaktadır. Bu nedenle maksimum verimlilik warp içindeki tüm thread'lerin aynı mantık yolundan geçtiği durumda elde edilmektedir. `if`, `switch` vb. dallanma operatörleri warp içindeki thread'leri farklı mantık yollarından geçmeye zorlayarak verimliliğin olumsuz etkilenmesine sebep olabilmektedir.

- **Veri Bağımlılığı:** Bazı algoritmalarda, bir adımın ürettiği çıktı sıradaki adımda girdi olarak kullanılmaktadır. Böyle durumlarda bir bloktaki thread'lerin kendi içinde senkronize edilmesi, hatta bazen tüm thread'lerin global olarak senkronize edilmesi ve kernel'lerin sıralı bir şekilde çalıştırılması gerekebilmektedir. Böyle bir algoritmaya Bölüm 4.2.3'de detaylandırılan Inverse Discrete Wavelet Transform (IDWT) örnek olarak gösterilebilir. Bu tip bağımlılıklar doğal olarak GPU verimliliğin düşmesine sebep olmaktadır.

Bir sonraki bölümde, tez çalışmasında çeşitli ölçeklerde paralelleştirilmesi ve hızlandırılması hedeflenen JPEG 2000 kod çözücü algoritması hakkında bilgi verilmekte, algoritmayı oluşturan her bir adım genel hatlarıyla anlatılmaktadır.

4. JPEG 2000 STANDARDINA GENEL BAKIŞ

JPEG 2000 dalgacık teknolojisine dayanan, modern ve ölçeklendirmeye oldukça elverişli bir görüntü kodlama sistemidir. Yüksek sıkıştırma performansı, bit hatalarına karşı dayanımı ve kayıpsız modun yanısıra esnek kayıplı modları desteklemesi nedeniyle uydu görüntü işleme, uzaktan algılama ve diğer birçok alanda sıklıkla kullanılmaktadır. Görevlerine devam etmekte olan RASAT ve GÖKTÜRK-2 milli yer gözlem uydularımızın görüntü sıkıştırmasında kullanılan ve yakın gelecekte fırlatıması planlanan İMECE uydumuzda kullanılacak olan görüntü sıkıştırma algoritması yine JPEG 2000'dir. Bununla birlikte yüksek algoritmik karmaşıklığı ise JPEG 2000 standardının en büyük dezavantajlarından biridir.

Uydu operasyonlarında genellikle kayıpsız sıkıştırma modu tercih edilmektedir. Hem bu sebeple, hem de işleme performansı açısından en kötü durum sernaryosunu göz önünde bulundurabilmek adına, bu tez çalışması kapsamında kayıpsız sıkıştırma modu üzerinde yoğunlaşılmıştır.

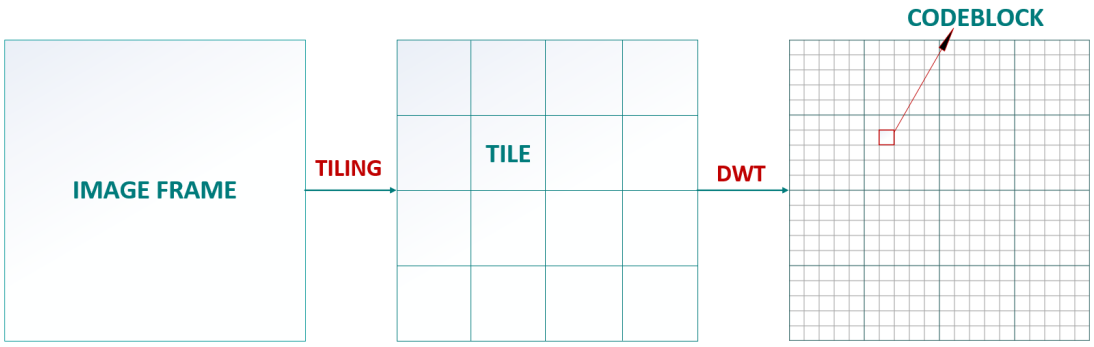
4.1 JPEG 2000 Temel Algoritmaları

JPEG 2000 standardı, birbiri üzerine eklenen birçok kısımdan oluşmaktadır. Bu çalışmada yalnızca 1. kısım, yani temel algoritmalar ele alınmaktadır. Bunun sebebi temel algoritmaların asıl veri sıkıştırma işleminin gerçekleştiği, yüksek hesapsal karmaşıklık içeren ve dolayısıyla optimize edilmesi gereken kısım oluşudur. Birinci kısmı meydana getiren temel algoritmalar aşağıda sıralanmakta ve bu bölümün devamında detaylandırılmaktadır.

1. DWT (Discrete Wavelet Transform)
2. BPC (Bit Plane Coding)
3. BAC (Binary Arithmetic Coding)

4.1.1 DWT

DWT dönüşümünden önce sıkıştırılacak görüntü kare şeklindeki karolara (tile) bölünür. Karoların özelliği, birbirlerinden tamamen bağımsız olarak işlenebilmeleridir. Ortaya çıkan her bir karo ise DWT sonrasında kodblok (code block) adı verilen daha küçük kare görüntü parçalarına bölünür. Bu kodbloklar da DWT'den sonraki aşamalarda birbirinden bağımsız işlenebilmektedirler. Bu dönüşüm Şekil 4.1 de gösterilmektedir.

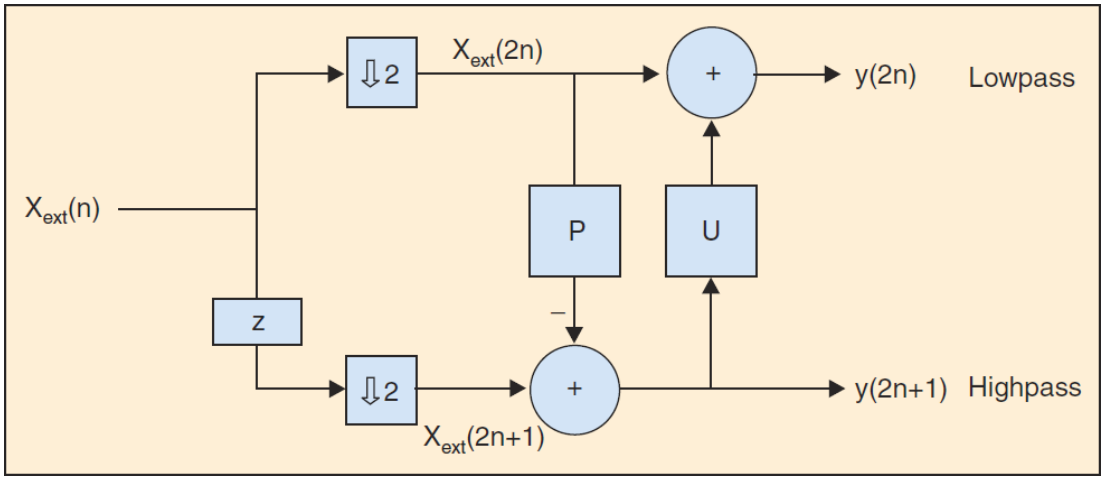


Şekil 4.1: Karo ve kodblok dönüşümü

Bu çalışmada geliştirilen algoritma tasarımına göre, 256×256 piksel boyutundaki karolara 3 seviye DWT uygulanmaktadır. Kodblok boyutu ise 32×32 piksel olarak seçilmiştir; böylece her bir karo toplamda 64 adet kodbloğa bölünmektedir. Kayıpsız sıkıştırmaya olanak sağladığından dolayı, DWT implementasyonu olarak Le Gall 5/3 filtresi seçilmiştir [2]. Şekil 4.2'de gösterildiği gibi, bu filtreye göre ortaya çıkan düşük ve yüksek frekanslı DWT katsayıları, sırasıyla Denklem 4.1 ve 4.2'e göre hesaplanmaktadır.

$$y(2n+1) = x_{ext}(2n+1) - \lfloor \frac{x_{ext}(2n) + x_{ext}(2n+2)}{2} \rfloor \tag{4.1}$$

$$y(2n) = x_{ext}(2n) + \lfloor \frac{y(2n-1) + y(2n+1) + 2}{4} \rfloor \tag{4.2}$$

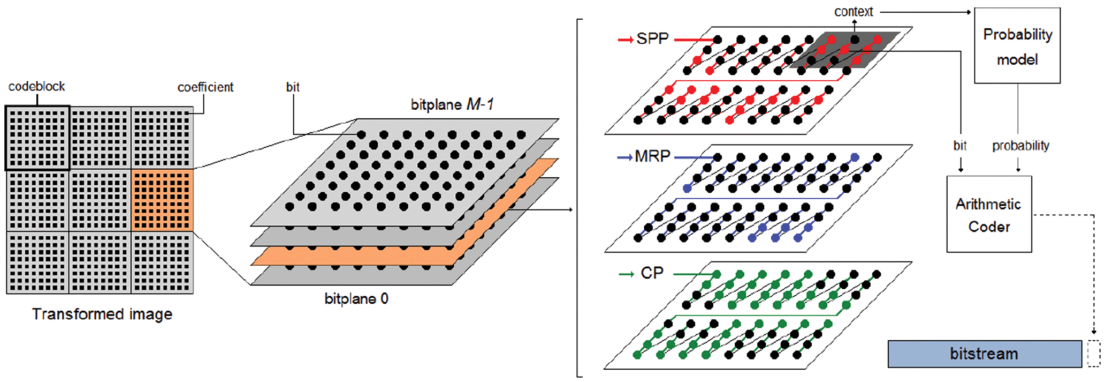


Şekil 4.2: Le Gall 5/3 DWT filtresi [2]

4.1.2 BPC

BPC algoritmasının ilk adımında, her bir kodblok kendi dinamik aralığı ile eşit sayıda bit düzlemine ayrıştırılır. Daha sonra bu bit düzlemleri 3 farklı geçiş ile taranarak kodbloğun olasılıksal modellemesi yapılır. Aşağıda listelenen bu geçişler, kodblok üzerinde Şekil 4.3'de gösterildiği gibi sırasıyla uygulanmaktadır.

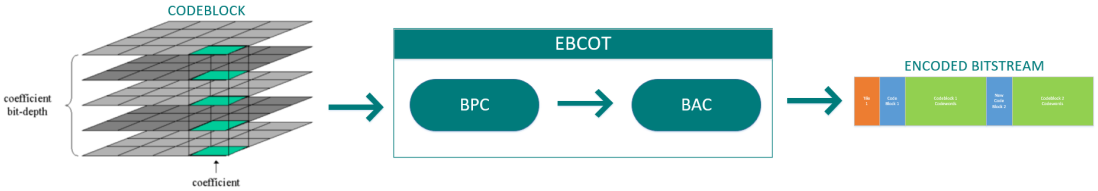
1. Significance Propagation Pass (SPP)
2. Magnitude Refinement Pass (MRP)
3. Clean-up Pass (CUP)



Şekil 4.3: BPC geçiş sıralaması

4.1.3 BAC

JPEG 2000 standardına göre, bir BAC türevi olan MQ-Coder kullanılmaktadır. BPC sonucunda ortaya çıkan olasılıksal modelin kullanılmasıyla, asıl sıkıştırma işlemi bu aşamada gerçekleşmektedir. Şekil 4.4'te de gösterildiği üzere, JPEG 2000'in entropi kodlama stratejisi olan EBCOT, BPC ve BAC algoritmalarının birleşiminden meydana gelmektedir.



Şekil 4.4: EBCOT akış diyagramı

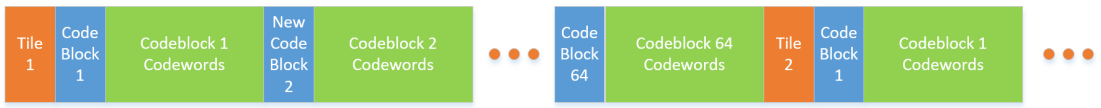
4.2 Sıkıştırma Açma Prosedürü

JPEG 2000 standardına göre sıkıştırılmış bir görüntünün geri kazanımı için, Şekil 4.5'de gösterilen adımlar uygulanmalıdır. Sıradaki bölümlerde bu 3 adım detaylı bir şekilde açıklanmaktadır.

Şekil 4.5: JPEG 2000 kod çözücü algoritma adımları

4.2.1 Kodblok Ayrıştırma

EBCOT, çıktı olarak kodlanmış bir bit dizisi üretmektedir. Şekil 4.6'de görülebileceği üzere, bu bit dizisinin içinde her bir karo ve kodbloğun başlangıç pozisyonlarını gösteren imleçler yer almaktadır.



Şekil 4.6: EBCOT sonucunda ortaya çıkan bit dizisi

Dolayısıyla, görüntünün geri kazanımı için öncelikle bu bit dizisi içindeki karo ve kodblokları ayrıştırılmalıdır. Bu adımın sonucunda, her bir sıkıştırılmış kodblok veri dizisine ayrı bir GPU thread'i atamak mümkün olacaktır. Tamamen sıralı bir işlem

olmasından dolayı, bu adımda yalnızca CPU kullanılmaktadır. Bununla birlikte birden fazla görüntünün sıkıştırılması sonucunda ortaya çıkan bit dizilerinin ayrıştırılması, birden fazla CPU thread'i kullanılarak paralelleştirilebilir.

4.2.2 EBCOT Çözme

EBCOT aşamasında kodbloklar (ve dolayısıyla karolar) birbirlerinden tamamen bağımsız olarak sıkıştırıldığı için, çözme sırasında da bunları bağımsız ve paralel bir şekilde işlemek mümkündür. Ancak sıralı işleyişi gereği; EBCOT çözme aşaması kodbloktan daha alt seviyede paralelleştirilememektedir [27]. Bu nedenle bir tasarım kısıtı olarak, her bir GPU thread'i bütün bir kodbloğun işlenmesinden sorumlu olmak durumundadır. İlerleyen bölümlerde de detaylandırılacağı üzere bu durum, yüksek dallanma çeşitliliği ile birlikte, JPEG 2000 algoritmasının GPU'da yüksek performanslı olarak işlenmesi önünde büyük bir engel teşkil etmektedir.

4.2.3 IDWT

Diğer iki adımın aksine IDWT adımı, neredeyse hiçbir sıralı operasyon içermediği için, GPU mimarilerine oldukça uyumlu ve dolayısıyla hızlandırılmaya çok daha elverişlidir. 3 seviyede tamamlanan bu adımdaki her bir seviyenin art arda gelme gerekliliği, IDWT'nin tam anlamıyla paralel bir algoritma olması hususundaki tek istisnadır. Yine de her seviyede 1 GPU thread'i başına 2 piksel atamak mümkün olduğundan, yüksek veri paralelliğinden faydalanmak mümkün hale gelmektedir. EBCOT çözme aşamasında thread başına en az 1024 piksel atanabildiği göz önünde bulundurulduğunda, bu iki adımın paralellik verimliliği arasındaki ciddi fark daha belirgin hale gelmektedir.

Tez kapsamında en yoğunluklu olarak yapılan çalışmaları içeren bir sonraki bölümde, JPEG 2000 kod çözücünün GPU'da hızlandırılması için uygulanan yöntemler ve elde edilen sonuçlar anlatılmaktadır.

5. JPEG 2000 KOD ÇÖZÜCÜNÜN GPU OPTİMİZASYONU

Önceki bölümlerde yer gözlem uydu operasyonları ve özellikle görüntü işleme süreçleri hakkında bilgi verilmiş, paralel ve dağıtık görüntü işleme için uygulanan güncel yöntemlerden bahsedilmiş ve JPEG 2000 kod çözücü algoritması anlatılmıştır. Bu bölümde ise, JPEG 2000 kod çözmenin birer parçası olan IDWT ve özellikle EBCOT adımı için tez kapsamında çalışılan, gerçeklenen, uygulanan ve test edilen GPU optimizasyon yöntemleri detaylandırılmaktadır.

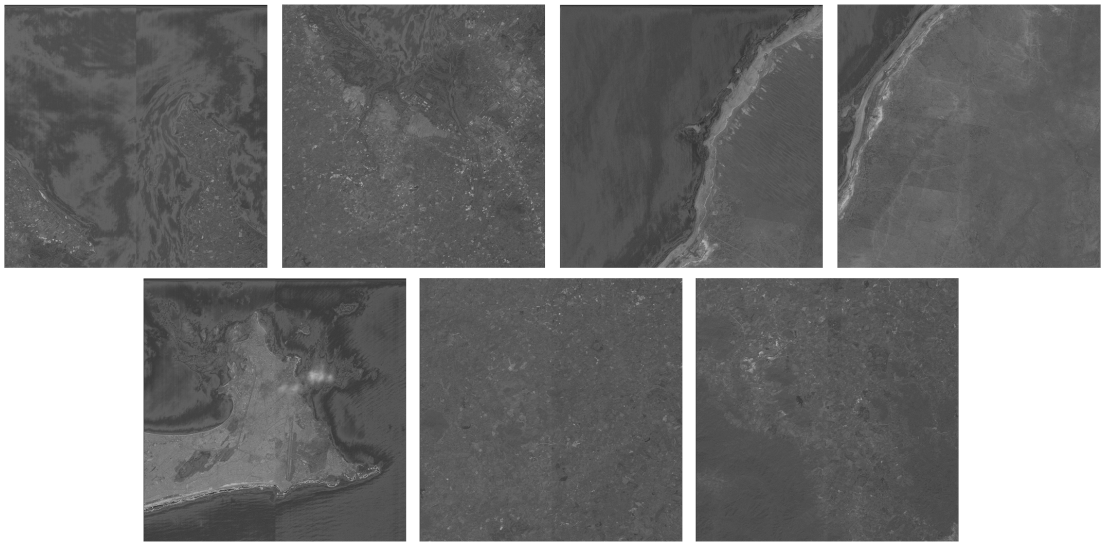
IDWT adımı piksel seviyesi paralelleştirmeye elverişli olması sebebiyle, EBCOT'un aksine sade bir GPU implementasyonuyla bile CPU'ya oranla oldukça hızlandırılabilmektedir. Üstelik ilerleyen kısımlarda da görülebileceği gibi, işleme zamanı anlamında EBCOT adımına kıyasla çok daha az zaman aldığı için IDWT adımında ek optimizasyonlar ile elde edilebilecek olası hızlanmanın genel performansa etkisi ihmal edilebilecek seviyede kalmaktadır. Bununla birlikte bu bölümde, EBCOT adımı kadar detaylı olmasa da, IDWT adımında uygulanan ek optimizasyon yöntemlerinden bahsedilmektedir.

5.1 Deney Konfigürasyonu

Bu çalışmada geliştirilen CPU ve GPU kod çözücülerin performans testleri, mekansal çözünürlüğü 8192×8192 piksel ve radyometrik çözünürlüğü 11 bit/piksel olan 7 adet pankromatik uydu görüntüsü üzerinde gerçekleştirilmiştir. Bu görüntülere ait küçük resimler Şekil 5.1'de gösterilmektedir.

Sıkıştırma-çözme testlerini gerçekleştirebilmek amacıyla, yalnızca CPU seviyesi paralelliği destekleyen bir kodlayıcı Java dilinde gerçeklenmiştir. Bu çalışmanın esas konusu olan kod çözücü ise CPU, GPU ve CPU-GPU hibrit paralellik modlarını desteklemektedir. Programlama dili olarak C++ ve CUDA kullanılmıştır.

Normal durumda her bir görüntü arka arkaya işlenmektedir. Buna ek olarak geliştirilen kod çözücünün GPU-paralel modu, tüm görüntüleri toplu (batch) halde işlemeyi mümkün kılacak şekilde tasarlanmıştır. İlerleyen bölümlerde, GPU-paralel modun normal ve toplu işlemedeki performansı birbiriyle ve diğer modlarla kıyaslanmaktadır.



Şekil 5.1: Test görüntülerinin küçük resimleri

Deneyler sırasında CUDA kernel çalışma süreleri ve GPU doluluk oranları NVIDIA Visual Profiler[6] aracı kullanılarak ölçülmüştür. Algoritma optimizasyon yöntemlerinin değerlendirilmesi aşamasında GPU donanımı olarak, 6 GB belleğe ve 1280 adet CUDA çekirdeğine sahip olan NVIDIA GeForce GTX 1060 kullanılmıştır.

5.2 CPU Kod Çözücü Tasarımı

Bu çalışmanın ana odağı GPU optimizasyonu olsa da, performans kıyaslaması yapılabilmesi ve Bölüm 6'nin konusu olan hibrit paralel kod çözücünün anlaşılabilmesi için, bu kısımda CPU kod çözücü tasarımından kısaca bahsedilmektedir.

CPU işlemcilerin çekirdek sayısı GPU'ya oranla çok daha az olduğundan, thread başına düşen iş miktarını azaltmak her durumda performansı artırmamaktadır. Bu nedenle EBCOT adımında GPU'da thread başına bir kodblok atanırken, geliştirilen CPU kod çözücünün hem IDWT hem de EBCOT adımlarında thread başına bir karo atanmıştır. C++ dilinde geliştirilen kod çözücüde karoların paralel bir şekilde işlenmesi OpenMP[7] aracılığıyla sağlanmıştır.

GPU kod çözücünün aksine, herhangi bir performans artışı sağlamadığından dolayı CPU kod çözücü toplu işleme senaryosunu desteklememektedir. CPU kod çözücü ile yapılan testlerde, bir görüntünün EBCOT açma adımı ortalama 7.2 saniye, IDWT adımı ise ortalama 507 milisaniye olarak ölçülmüştür.

[6] https://developer.nvidia.com/nvidia-visual-profiler
[7] https://www.openmp.org/

5.3 EBCOT Adımında Uygulanan Optimizasyon Yöntemleri

Bu kısımda, EBCOT çözme için GPU için uygulanan her bir CUDA optimizasyon tekniği detaylandırılmaktadır.

5.3.1 Yazmaç (Register) Sayısı

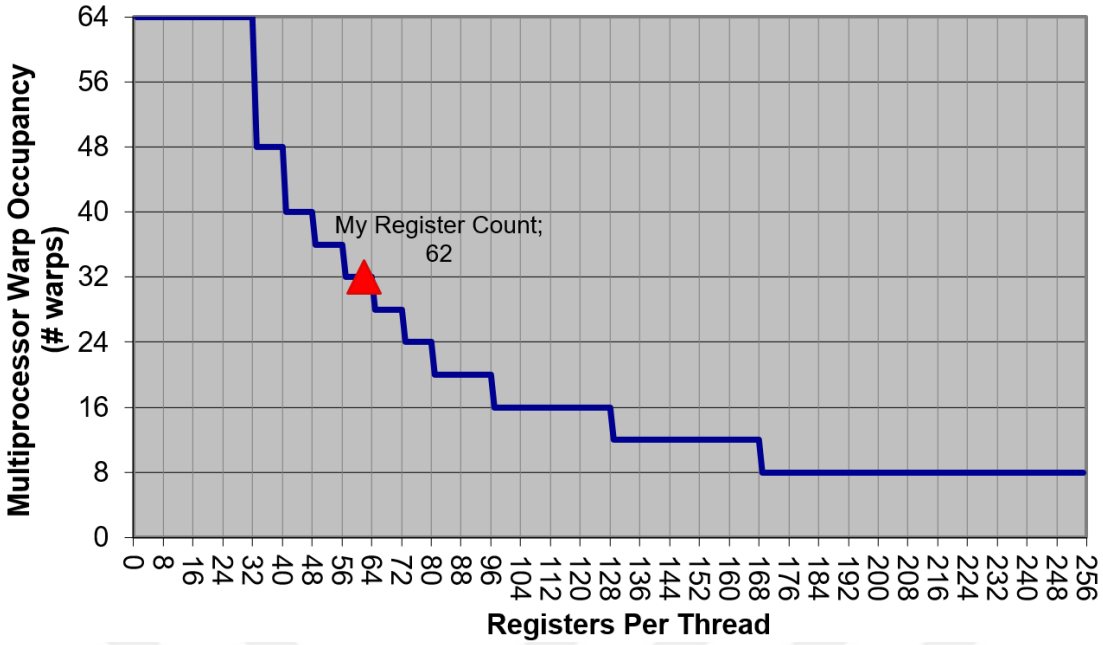
Hesaplamsal yoğunluğu yüksek CUDA kernellerinde, thread başına kullanılan yazmaç sayısı kolaylıkla GPU doluluk oranını kısıtlayan ana etken olabilmektedir. Önceden de belirtildiği gibi, EBCOT algoritmasında her bir thread bütün bir kodbloğun işlenmesinden sorumludur. Herhangi bir optimizasyon yapılmamış haliyle, geliştirilen kod çözücünün EBCOT aşamasında her bir thread'in 62 adet yazmaç kullanmakta olduğu gözlemlenmiştir. Bu durumda Şekil 5.2'de de görülebileceği gibi, teorik işlemci doluluk oranı warp cinsinden $32/64$, yüzde olarak ise %50'dir. Deneylerde ise gerçek doluluk oranı %40.7, kernel çalışma süresi ise 4.9 saniye olarak ölçülmüştür.

CUDA Occupancy Calculator[8] aracı sayesinde, yazmaç sayısı gibi kaynakların kullanımı ile teorik doluluk oranı arasındaki ilişki önceden hesaplanabilmektedir. Şekil 5.2 bu araç tarafından oluşturulmuş bir diyagramdır ve yazmaç sayısının 56'ya düşürülmesi durumunda teorik doluluk oranının $36/64$'e, yani %56.2'ye yükseleceğini göstermektedir. Bu analizden yola çıkarak, kernel kodlarında bir takım değişiklikler yapılarak yazmaç sayısı 56'ya düşürülmüş ve ölçümler tekrarlanmıştır. Optimize edilmiş kernel için gerçek doluluk oranı %46.4, çalışma süresi ise 4.7 saniye olarak ölçülmüştür. Dolayısıyla bu optimizasyon sonucunda %5'e yakın bir hızlanma sağlanmıştır.

5.3.2 Sabitlenmiş Bellek (Pinned Memory)

İŞletim sistemlerinin yapısı gereği, CPU bellek tahsisleri normalde sayfalanabilir (pageable) yapıdadır. Buradaki amaç bellekte az kullanılan sayfaların diske yazılmasıyla başka sayfalara yer açabilmektir. Ancak diske yazılmış olan sayfalara erişilmesi gerektiğinde, sayfa hatası dolayısıyla diske erişilmesi gerektiği için okuma verimliliği önemli ölçüde azalmaktadır. Görüntü işleme uygulamarında bellek çok yoğun bir şekilde kullanıldığından, tahsis edilen bellek sayfalarının diske yazılma olasılığı oldukça yüksektir. Bu durumun önüne geçmek için, CUDA aracılığıyla CPU belleğindeki bazı sayfaların sabitlenmesi sağlanarak, daima fiziksel bellek alanında bulunmaları garanti altına alınmıştır.

[8] https://developer.download.nvidia.com/compute/cuda/CUDA$_{O}ccupancy_{c}alculator.xls$



Şekil 5.2: Yazmaç kullanımının GPU doluluk oranına etkisi

Sabitlenmiş bellek kullanımının bir diğer faydası, CPU ve GPU arasındaki bellek transferlerini kernel çalıştırmayla eş zamanlı yapmaya olanak sağlamasıdır. İlerleyen kısımlarda bu konuya tekrar değinilecektir.

JPEG 2000 sıkıştırma açma sırasında CPU ve GPU arasında 2 çeşit bellek transferi yapılmaktadır:

- Kodblok ayrıştırma işleminden geçmiş bit dizisinin (ortalama 42.5 MB) EBCOT çözme öncesinde CPU belleğinden GPU belleğine kopyalanması.
- IDWT sonucunda tamamen açılmış olan görüntü verisinin (128 MB) GPU belleğinden CPU belleğine kopyalanması.

Yapılan deneylerde öncelikle herhangi bir optimizasyon yapılmamış ve işletim sisteminin yukarıdaki iki çeşit bellek tipini gerekli gördüğünde diske yazmasına izin verilmiştir. Bu senaryoda görüntü başına bit dizisinin kopyalanması ortalamada 9.5 milisaniye, açılmış görüntünün kopyalanması ise ortalamada 25.5 milisaniye sürmüştür.

Daha sonra bu iki bellek çeşidinin sabitlenmesi sağlanarak deneyler tekrarlanmıştır. Bu durumda kopyalama sürelerinin ortalamada 3.7 milisaniye ve 10.3 milisaniyeye düştüğü gözlemlenmiştir. Dolayısıyla bu optimizasyon sonucunda bellek transferlerinde yaklaşık 2.5 kat hızlanma elde edilmiştir.

5.3.3 Salt Okunur Önbellek (Read-Only Cache)

GPU belleğinden okuma hızını potansiyel olarak artıran salt okunur önbellek, CUDA CC 3.5 ile uygulamaya koyulmuştur. Bu teknolojinin kullanılabilmesi için ön şart olarak erişilmesi istenen bellek bloğunun yaşam döngüsü boyunca salt okunur olması gerekmektedir.

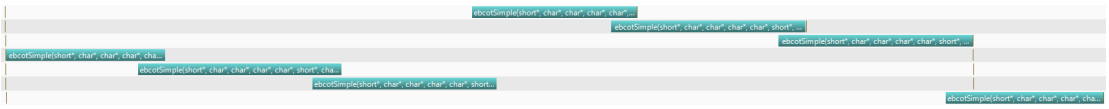
EBCOT kernelinin çalışması boyunca salt okunur olan yegane bellek alanı sıkıştırılmış bit dizisidir. Bu nedenle, yapılan deneylerde bit dizisinin varsayılan şekilde okunması ile salt okunur önbellek kullanılarak okunması ayrı ayrı test edilmiştir. Normalde ortalamada 4.7 saniye olan kernel çalışma süresi, salt okunur önbellek kullanıldığı durumda 4.5 saniyeye düşmüştür ve böylelikle %4'lük bir hızlanma sağlanmıştır.

5.3.4 Asenkron Kernel Çalışması ve Bellek Transferi

Geliştirilen kod çözücünün GPU-paralel modunda, görüntüler tek tek veya toplu halde işlenebilmektedir. Bu kısımda, toplu işlemeyi mümkün kılan optimizasyon tekniği konu alınmaktadır.

CUDA stream'leri kullanılarak, kernel çalıştırma ve bellek kopyalama işlemleri birbirlerinden bağımsız ve paralel bir şekilde gerçekleştirilebilmektedir. Birbirine bağlı ve sıralı olan işlemler aynı stream'e atandığı sürece, bağımsız operasyonların farklı ve çok sayıda stream'e dağıtılması mümkündür.

Yapılan deneylerde kullanılan 7 adet görüntünün her biri tamamen diğer görüntülerden bağımsız olarak işlenebilmektedir. Dolayısıyla, toplu işleme sırasında 7 adet stream oluşturulmakta ve bunların her birine bir adet görüntüye dair kernel ve kopyalama operasyonları atanmaktadır. Bu işlemin GPU'da çalışması NVVP[9] kullanılarak üretilen zaman çizelgesi Şekil 5.3'de gösterilmektedir.



Şekil 5.3: Toplu işleme GPU zaman çizelgesi

Tek tek ve toplu işleme senaryoları için ayrı ayrı yapılan ölçümlere göre, 7 görüntünün tek tek işlenmesi ortalama 33.5 saniye, toplu işlenmesi ise 32.4 saniye sürmektedir.

[9]https://developer.nvidia.com/nvidia-visual-profiler

5.3.5 Bit Düzlemlerinin Bellek Modellemesi ve Paylaşımlı Bellek

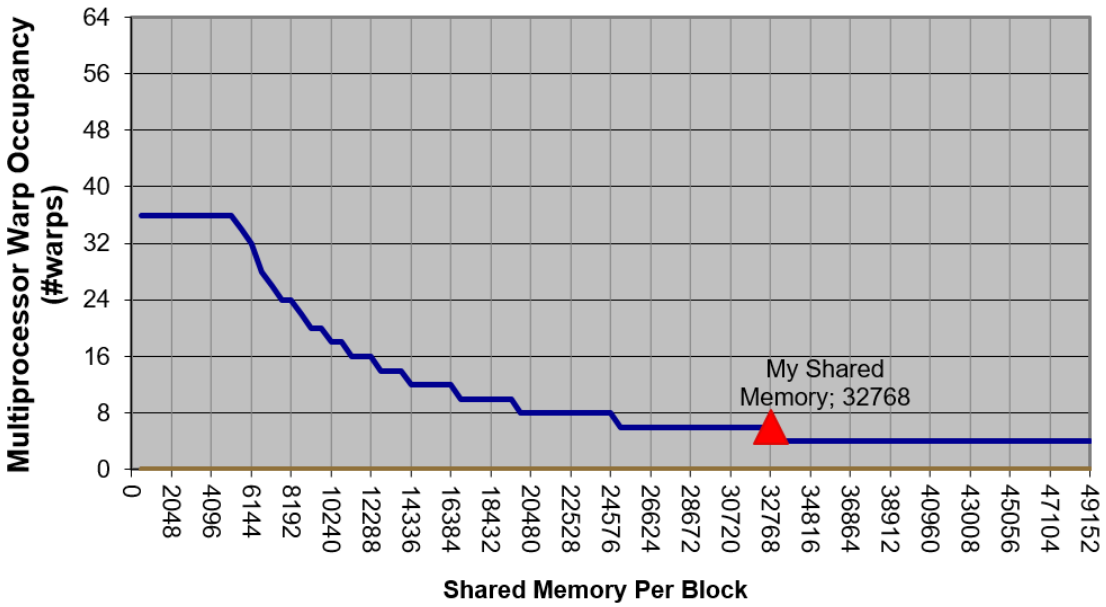
Bu kısımda EBCOT kernelinde ihtiyaç duyulan bayrak bit düzlemlerinin bellekteki reprezentasyonu ve bu verinin global bellekten paylaşımlı belleğe taşınmasının performans üzerindeki etkileri incelenmektedir.

Bölüm 3.2.2'te de belirtildiği gibi, GPU'larda bulunan paylaşımlı bellek (shared memory), global belleğe kıyasla daha düşük kapasiteye ve daha yüksek erişim hızına sahip bir bellek çeşididir. EBCOT adımında boyut itibariyle paylaşımlı bellekte tutulması mümkün olan tek veri tipi, χ, σ, σ' ve η sembolleriyle ifade edilen 4 adet bayrak bit düzlemidir. Kodblok boyutu 32×32 piksel olduğundan, bu bayrak bit düzlemleri toplamda 4096 bitlik veriden meydana gelmektedir. Bir thread bloğunda 64 adet thread (kodblok) olduğuna göre, bu verinin paylaşımlı bellekte kaplayacağı alan toplamda 32 KB'tır. CUDA programlama rehberinde [1] de belirtildiği üzere, CC değeri 6.1 olan GTX 1060 kartında thread bloğu başına düşen maksimum paylaşımlı bellek boyutu 48 KB'tır. Dolayısıyla deneylerde bu veri için paylaşımlı belleğin kullanılması mümkündür.

Çalışmanın ilk aşamalarındaki bellek modeline göre, veri GPU'da `boolean` tipindeki dizilerde tutulmuştur. Ancak NVVP ile yapılan analizler sonucunda, her bir `boolean` dizi elemanının GPU belleğinde 1 bayt yer kapladığı farkedilmiştir. Dolayısıyla gerçek paylaşımlı bellek ihtiyacı bu modele göre 32 KB yerine 256 KB olmakta ve dolayısıyla GPU kapasitesini aşmaktadır. Bu sorunun önüne geçip bellek ihtiyacını 32 KB'a indirebilmek için bellek modellemesi optimize edilerek bayrak bit düzlemleri için bayt dizileri kullanılmış, bayraklara erişim için ise bit maskeleme yöntemi uygulanmıştır.

Bu tasarım değişikliği, henüz paylaşımlı bellek kullanımı yapılmadan tek başına önemli ölçüde performans artışı sağlamıştır. Öyle ki, bir kare görüntü için EBCOT kernel çalşma süresi ortalama 4.5 saniyeden 4.2 saniyeye düşmüş, toplu işlemede ise 7 kare görüntünün toplam EBCOT çözme süresi 32.4 saniyeden 29.4 saniyeye düşmüştür.

Bir sonraki aşamada bayrak bit düzlemleri için paylaşımlı bellek kullanılarak deneyler tekrarlanmıştır. Ancak bu durumda bir thread bloğu başına 32 KB paylaşımlı bellek kullanılması GPU doluluk oranını ciddi ölçüde düşürmüş ve dolayısıyla performans genel olarak olumsuz etkilenmiştir. CUDA Occupancy Calculator aracılığıyla üretilen Şekil 5.4'te de görülebileceği üzere, doluluk oranı %56.2'den %9.4'e düşmektedir. Ortalama kernel çalışma zamanları ölçüldüğünde ise tek kare için EBCOT kernel süresinin 4.2 saniyeden 8.7 saniyeye çıktığı, toplu EBCOT işleme süresinin ise 29.4 saniyeden 60 saniyeye çıktığı görülmüştür.



Şekil 5.4: Paylaşımlı bellek kullanımının GPU doluluk oranına etkisi

5.3.6 Dinamik Paralellik

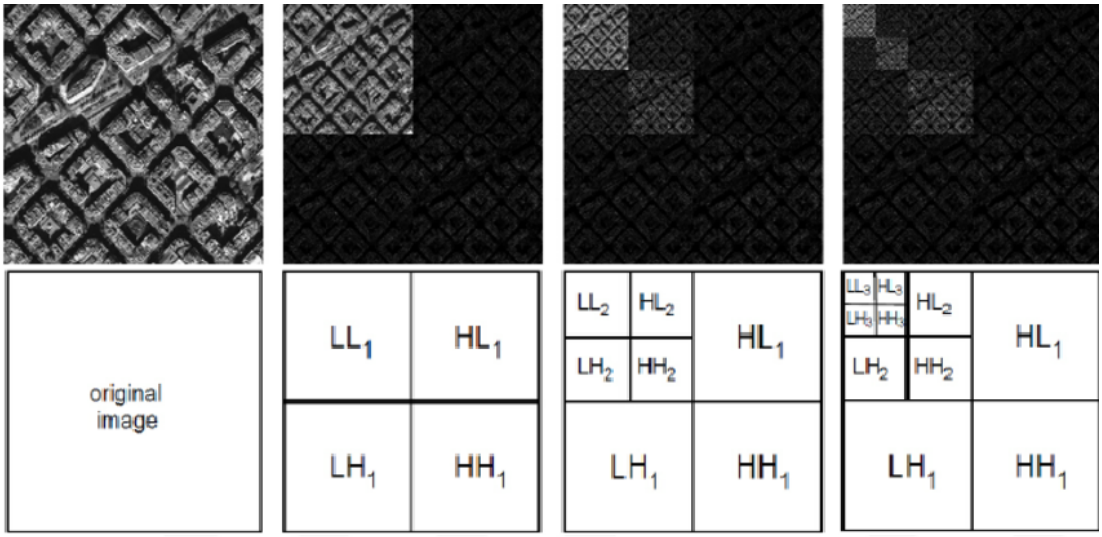
EBCOT adımının sonunda kodblok bellek alanında bulunan her bir pikselin öncelikle işaret-genlik formatından (signed-magnitude) ikiye tümleyen formata (2's complement) dönüştürülmesi, daha sonra da görüntü bellek alanına kopyalanması gerekmektedir.

Bu optimizasyon adımına kadar, EBCOT kernel konfigürasyonunda kodblok başına 1 thread atanmıştır. Ancak kernel sonundaki dönüşüm ve kopyalama işlemi teorik olarak paralelleştirmeye müsaittir. Bu paralelliği sağlama yöntemi olarak, bir kernel içinden başka kerneller çalıştırmaya olanak sağlayan dinamik paralellik özelliği değerlendirilmiş, test edilmiş ve performansa olan etkisi ölçülmüştür.

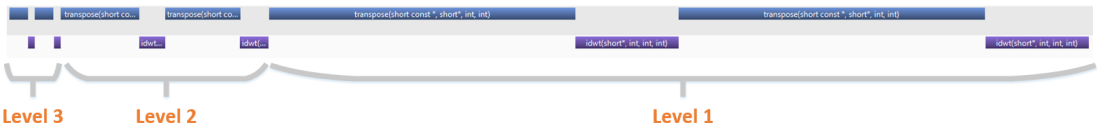
Öncelikle kernel sonunda geçekleşen 1024 pikselin tek thread tarafından sıralı dönüşüm ve kopyalama işlemi kaldırılmıştır. Bunun yerine, her bir thread'in 1024'er adet thread başlattığı yeni bir dinamik kernel tanımlanmıştır. Ancak yapılan testler, bu değişikliğin gerçekleşen GPU doluluk oranını ortalama %46.4'ten %38.5'e düşürdüğünü göstermiştir. Bunun bir sonucu olarak, bir görüntü için EBCOT kernel süresi ortalama 4.2 saniyeden 4.6 saniyeye çıkmış, toplu işlemede ise bu süre 29.4 saniyeden 40.2 saniyeye çıkmıştır. Performanstaki bu düşüşün temel sebebi, düşük aritmetik karmaşıklığa sahip olan yeni thread'leri dinamik olarak başlatmanın getirdiği ek yükün genel performans açısından daha ağırlıklı olmasıdır.

5.4 IDWT Adımında Uygulanan Optimizasyon Yöntemleri

Sıkıştırma esnasında önce yatay, sonra düşey eksende ardışık olarak 3 seviyeli bir DWT dönüşümü uygulanmaktadır. Bu işlem örnek bir görüntüye uygulandığında elde edilen çıktı ve alt-bantlar Şekil 5.5'te gösterilmektedir. Kod çözücüde ise bu işlemin tersi olarak 3 seviyeli bir IDWT dönüşümü uygulanmaktadır. Her bir seviyede farklı büyüklükte görüntü parçaları işlendiği için, Şekil 5.6'de gösterilen GPU zaman çizelgesinden de anlaşılabileceği üzere kernel çalışma zamanları her bir seviyede farklılık göstermektedir.



Şekil 5.5: Her bir DWT seviyesinde ortaya çıkan görüntü ve alt-bantlar



Şekil 5.6: Test görüntülerinin küçük resimleri

Bellek erişimi için harcanan zamanı en aza indirebilmek için her bir seviyede düşey eksende IDWT uygulamak yerine, görüntü önce transpoze edilip yatay IDWT uygulanmakta ve son olarak tekrar transpoze edilmektedir. Düşey dönüşümde transpoze işlemi yapılmaması halinde, her bir thread atandığı piksele ve bu pikselin bir alt ve bir üst satırındaki piksele erişmek durumunda kalacaktır. Transpoze işlemi yapıldığında ise her bir thread atandğı pikselin sağındaki ve solundaki sütunlarda bulunan piksellere, yani ardışık bellek adreslerine erişmektedir. Üstelik bu durumda bir warp içinde bulunan tüm thread'ler de birbirlerine göre ardışık bellek adreslerine erişmektedir. Global bellekte okuma ve yazmaların ardışık adreslerden yapılması kopyalama sürelerini en aza indirdiği için, düşey IDWT dönüşümlerinin önüne ve arkasına transpoze kernelleri eklemek toplam IDWT çalışma süresini azaltmaktadır.

Bölüm 5.3'te bahsi geçen optimizasyon yöntemleri arasından, IDWT ve transpoze kernellerinde paylaşımlı bellek ve transpoze kernelinde ek olarak salt okunur önbellek kullanılmıştır. IDWT kernelinde bir thread bloğu tarafından işlenecek pikseller global bellekten paylaşımlı belleğe kopyalanmakta, paylaşımlı bellekte işlenmekte ve işlenmiş veri tekrar global belleğe kopyalanmaktadır. Böylelikle Denklem 4.1 ve Denklem 4.2'deki dönüşüm işlemleri için ikişer defa global bellekte okuma ve yazma yapmak yerine yalnızca birer defa okuma ve yazma yapılmaktadır.

Transpoze kernelinde ise veri global girdi arabelleğinden paylaşımlı belleğe kopyalanmakta, paylaşımlı bellekte transpoze edilmekte ve ardından global çıktı arabelleğine kopyalanmaktadır. Böylelikle ilgili warp içinde bulunan thread'lerin hem okuma hem de yazma işlemlerini ardışık bellek adresleri için gerçekleştirmesi sağlanarak transfer süreleri en aza indirilmektedir.

Transpoze kerneli bir girdi bir de çıktı arabelleği ile çalışmakta ve girdi arabelleği kernel yaşam döngüsü boyunca salt okunur olarak kullanılmaktadır. Bu durumdan faydalanılarak, transpoze kernelinde ek olarak girdi arabelleği için salt okunur önbellek kullanılmış ve girdi verinin okunma süresi optimize edilmiştir.

Sonuç olarak, bir görüntünün 3 seviyeli IDWT dönüşümü için ardışık olarak çalıştırılan 12 CUDA kernelinin ortalama çalışma süresi 36 milisaniye olarak ölçülmüştür. EBCOT kernelinin en optimize edilmiş halinde işleme süresinin 4.2 saniye olduğu göz önünde bulundurulduğunda, SIMT mimarisine uyumlu olmayan EBCOT algoritmasının meydana getirdiği darboğaz çok daha net bir şekilde göze çarpmaktadır.

5.5 Performans Değerlendirmesi

Tez çalışması kapsamında geliştirilen CPU ve GPU kod çözücü performansları bu bölümde birbiriyle kıyaslanmaktadır. Ayrıca bu bölümde değinilen optimizasyon adımlarının yalnızca deneylerde kullanılan GPU modeli için değil, tüm cihazlar için geçerli olduğu tartışılmakta ve gerekçelendirilmektedir.

Bölüm 5'de detaylandırılan çalışmayı özetlemek gerekirse, GPU kod çözücünün tasarımı aşağıdaki gibidir:

- Thread başına kullanılan yazmaç sayısı 56 ile sınırlandırılmıştır.
- Sıkıştırılmış bit dizisi ve açılmış görüntü verisi için sabitlenmiş bellek kullanılmıştır.
- EBCOT kerneli içinden sıkıştırılmış bit dizisine erişim için salt okunur önbellek kullanılmıştır.

- Toplu işleme senaryosunda asenkron kernel çalıştırma ve bellek transferleri için CUDA stream'leri kullanılmıştır.
- Bayrak bit düzlemleri (χ, σ, σ' ve η) bellekte bayt dizileri olarak tutulmuş ve erişim için maskeleme kullanılmıştır.
- Paylaşımlı bellek IDWT kernelinde kullanılmış, EBCOT kernelinde kullanılmamıştır.
- EBCOT kernelinde dinamik paralellik kullanılmamış, kodbloktaki 1024 pikselin tamamı tek bir thread tarafından görüntü bellek alanına kopyalanmıştır.

Bu konfigürasyonda, tek bir görüntü için EBCOT adımı ortalama 4.2 saniye, IDWT adımı ise ortalama 33 milisayniye olarak ölçülmüştür. 7 adet görüntünün toplu işlenme süresi ise bellek transferleri, EBCOT ve IDWT adımları dahil olmak üzere ortalama 29.4 saniye olarak ölçülmüştür. EBCOT kerneli için [27]'de yapılan çalışmada elde edilen %17'lik GPU doluluk oranıyla, burada elde edilen %46.4'lık doluluk oranı karşılaştırıldığında bu optimizasyonların etkinliği açıkça görülebilmektedir.

5.5.1 CPU ve GPU Kod Çözücülerin Performans Kıyaslaması

Çizelge 5.1'de her bir adımın CPU ve GPU kod çözücüde ortalama ne kadar sürdüğü gösterilmiştir. Bölüm 5.2'deki ölçümlerle kıyaslandığında, GPU kod çözücünün CPU kod çözücüye oranla yaklaşık 1.8 kat daha hızlı olduğu görülmektedir. GPU tekli işleme toplu işleme senaryoları arasında ise, toplam çalışma zamanı baz alındığında yalnızca %1'lik bir performans farkı gözlemlenmektedir. Bu farkın ihmal edilebilecek kadar küçük olması, Bölüm 6.2'de detaylandırılacak olan hibrit paralel tasarımı etkileyen önemli bir faktördür.

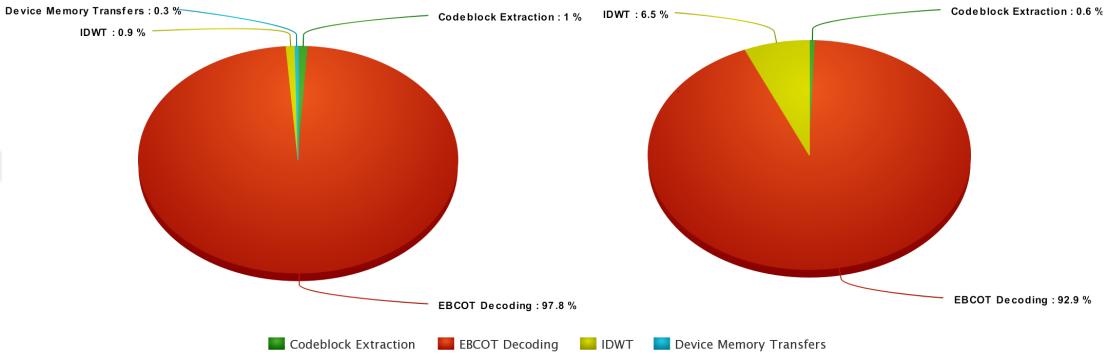
Şekil 5.7'de ise her bir adımın orantısal olarak CPU ve GPU kod çözücülerde aldığı süre görselleştirilmiştir. Kodblok ayrıştırma adımı tamamen sıralı bir işlem olduğundan, her durumda CPU kullanılarak yapılmakta, bu yüzden eşit zaman almaktadır.

5.5.2 Optimizasyon Yöntemlerinin Genel Geçerlilik Gerekçeleri

Her ne kadar bu bölümde yapılan deneylerde yalnızca GTX 1060 cihazı kullanılmış olsa da, aşağıdaki gerekçelerden dolayı farklı GPU modelleri kullanıldığında da benzer sonuçlar elde edilecektir:

Çizelge 5.1: Her bir adımın CPU ve GPU kod çözücüde ortalama aldığı süre

Kod Çözücü	Kodblok Ayrıştırma	EBCOT	IDWT	GPU-CPU Bellek Transferleri	Toplam
CPU	43 ms	7220 ms	507 ms	0	7770 ms
GPU (Tekli)	43 ms	4180 ms	36 ms	13 ms	4272 ms
GPU (Toplu)	43 ms	4140 ms	36 ms	13 ms	4232 ms



Şekil 5.7: Her bir adımın orantısal olarak aldığı süre (a) GPU (b) CPU

- Kullanılan yazmaç sayısı ve paylaşımlı bellek miktarı tüm GPU'lar için sınırlı birer kaynaktır. Her bir cihaz için sayısal değerler farklılık göstermekle birlikte, thread başına düşen yazmaç sayısının ve thread bloğu başına düşen paylaşımlı bellek miktarının artırılması kaçınılmaz olarak GPU doluluk oranını düşürecektir. EBCOT adımının karmaşıklığı nedeniyle, bu iki kaynağın kullanımı bilinçli olarak sınırlanmadığı durumda verimlilik olmusuz etkilenecektir. Paylaşımlı bellek kullanımının IDWT ve transpoze kernellerinde olumsuz bir etki yaratmamasının sebebi, paylaşımlı belleğe taşınan verinin en kötü durum senaryosunda bile en fazla 4 KB olmasıdır. Benzer şekilde, bu iki kernelde thread başına kullanılan maksimum yazmaç sayısı 14'tür. IDWT ve EBCOT adımlarının algoritmik karmaşıklıkları arasındaki bu fark, yazmaç ve paylaşımlı bellek kaynaklarının kullanımını doğrudan etkilemekte ve farklı yaklaşımlar gerektirmektedir.

- Sabitlemiş bellek ve salt-okunur önbellek kullanımı yine GPU modelinden bağımsız olarak bellek transferlerini hızlandıracaktır. Sabitlenmiş bellek kullanırken dikkat edilmesi gereken nokta, sabitlenen bellek boyutunun mevcut fiziksel CPU bellek kapasitesi göz önünde bulundurularak seçilmesi ve ihtiyaç sonlandığı anda serbest bırakılmasıdır. Benzer şekilde salt okunur önbellek de fiziksel GPU bellek kapasitesi göz önünde bulundurularak kullanılmalıdır. Bu hususlara dikkat edildiği

sürece, CPU-GPU bellek transfer hızları ve global bellekten salt okunur veriye erişim hızları kullanılan GPU modelinden bağımsız olarak artacaktır.

- CUDA stream'leri kullanılarak bağımsız görevlerin asenkron hale getirilmesi birçok CUDA uygulamasında kullanılan bir yöntemdir. Burada önemli olan, bu yöntemin yalnızca birbirinden bağımsız görevlerin mevcut olduğu durumlarda kullanılabileceğidir. Bu bölümde yapılan deneylerde 7 farklı görüntü birbirinden bağımsız olarak işleniyor olması buna örnek olarak gösterilebilir. Bu koşul sağlandığı sürece Bölüm 5.3.4'de bahsedilene benzer bir performans artışı, teorik olarak herhangi bir GPU kartı ile elde edilebilecektir.

- Son olarak dinamik paralellik, dinamik olarak yaratılan kernel'ler tarafından yoğunluklu olarak aritmetik operasyonların yapıldığı senaryolarda performans artışı sağlayabilen bir yöntemdir. Bölüm 5.3.6'te de bahsedildiği gibi EBCOT adımının son aşamasında yalnızca arabellekler arasında bir kopyalama işlemi yapılmakta, herhangi bir aritmetik operasyon yapılmamaktadır. Dolayısıyla EBCOT adımının son aşamasında denenmiş olan dinamik paralellik GTX 1060 özelinde performans kaybına yol açtığı gibi, bir başka GPU kullanıldığında da benzer bir etki yaratacaktır.

5.5.3 Ölçeklenebilirlik

Bu bölümde önerilmiş olan GPU optimizasyon yöntemleri bir önceki kısımda da belirtildiği gibi yalnızca deney konfigürasyonunda kullanılan GTX 1060 kartına özel olmayıp, genel olarak tüm NVIDIA GPU'larda benzer performans sonuçları sağlayacaktır. Dolayısıyla zorlu performans kısıtları ile karşılaşıldığı durumda bu bölümde önerilen yöntemlerin ölçeklenebilmesi için daha yüksek CC seviyesine sahip bir GPU kullanmak gerekmektedir. Örnek vermek gerekirse Bölüm 6'da önerilen hibrit kod çözücünün performans ölçümlerini gösteren Çizelge 6.1'e göre, CC seviyesi 6.1 olan GTX 1060 yerine CC seviyesi 7.5 olan RTX 2070 kullanıldığında GPU kod çözücünün performansı 2 kattan daha fazla artmaktadır.

Elbette yalnızca daha iyi bir GPU kullanmanın yeterli olamayacağı kadar zorlu performans kısıtlarıya karşılaşmak mümkündür. Böyle durumlarda Bölüm 6 ve Bölüm 7'de önerilen hibrit ve dağıtık paralelliğe dayalı yöntemlere başvurmak gerekecetir.

Bir sonraki bölümde, burada detaylandırılan GPU optimizasyon yöntemleri sonucunda hızlandırılan GPU kod çözücü ve Bölüm 5.2'de bahsedilen CPU kod çözücü bir arada kullanılarak geliştirilmiş olan hibrit-paralel kod çözücü tasarımı anlatılmaktadır.

6. JPEG 2000 KÖD ÇÖZÜCÜ İÇİN HİBRİT PARALELLİK

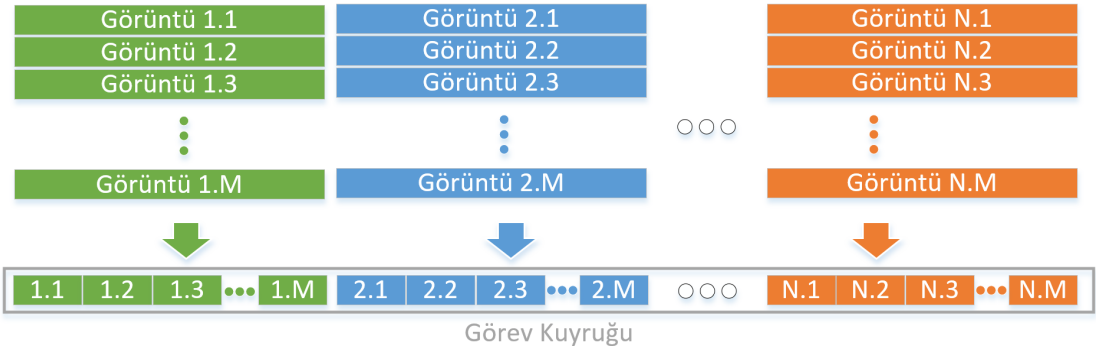
Bir önceki bölümde JPEG 2000 kod çözücü için önerilen GPU optimizasyon yöntemlerinden bahsedilmiş ve bunların uygulanması sonrasında, test görüntüleri üzerinde GPU ve GPU kod çözücüler arasında performans kıyaslaması yapılmıştır. Bu bölümde ise, uydu görüntülerinin işlenmesi sırasında her iki tip işlemciyi birden tam verimlilikle kullanmaya olanak sağlayan CPU-GPU hibrit bir tasarım önerilmektedir. Bir önceki bölümde detaylı olarak anlatılmış olan CPU ve GPU kod çözücüler, herhangi bir değişikliğe gerek olmadan bu bölümün konusu olan hibrit tasarımın bir parçası olarak kullanılmaktadır.

6.1 Yük Dengeleme Mekanizması

Hibrit kod çözücü tasarımının dayanak noktası, uydu görüntülerinin Bölüm 2.2'de bahsedildiği gibi parçalı halde indirilmesi ve kaydedilmesidir. Bir tek kareden oluşan uydu görüntülerinin bile çoğu durumda onlarca parçaya bölünmüş bir şekilde yer istasyonuna indirilmesinin yol açtığı granülerlik, GPU ve CPU arasındaki yük dengelemesinin hibrid kod çözücüde verimli bir şekilde yapılabilmesine izin vermektedir. Bu çalışma kapsamında geliştirilen hibrit kod çözücünün yük dengeleme mekanizması aşağıdaki adımlardan oluşmaktadır:

1. İndirilen görüntülerin her bir parçası için ayrık birer görev tanımlanır ve tüm görevler her iki kod çözücü tarafından thread-safe bir şekilde erişilebilen bir kuyruğa eklenir.
2. GPU ve CPU kod çözücülerin her biri, aynı anda kuyruktan birer görev alarak sorumlu olduğu görüntü parçasını işlemeye başlar.
3. Mevcut görevini tamamlayan kod çözücü, kuyruktan thread-safe bir şekilde sıradaki görevi alır.
4. Kuyruktaki görevlerin hepsi tamamlanana kadar 3. adım tekrarlanır.

Şekil 6.1'de her biri M adet parçadan oluşan N adet görüntünün görev kuyruğuna eklendiği örenek bir senaryo gösterilmektedir.



Şekil 6.1: Görüntü parçalarının görev kuyruğuna eklenmesi

Bu mekanizma sayesinde her bir kod çözücü kendi performansıyla doğru orantılı sayıda görüntü parçası işlemekte ve dolayısıyla teorik optimuma oldukça yakın bir yük paylaşımı gerçekleştirilebilmektedir. Teorik optimuma ulaşmanın önündeki en önemli engel, işlemcilerin son görevlerini tam olarak aynı anda bitirmeme durumlarıdır. Böyle durumlarda işlemcilerden biri son görüntü parçasını işlerken, diğeri ise bu görevin tamamlanması için gereken süre kadar boşta beklemek durumunda kalabilir. Her ne kadar görüntü parçalarının yeterince küçük olduğu durumlarda bu bekleme süreleri ihmal edilebilir seviyede kalsa da, görüntü parçalarının büyük olduğu durumlarda performans olumsuz etkilenebilmektedir. Bunun önüne geçebilmek için, uydudan halihazırda parçalı olarak gelen görüntüler yer istasyonunda işlenmeden önce daha küçük parçalara bölünebilir; çünkü JPEG 2000 bit dizisi Şekil 4.6'da gösterilen karo başlıklarından bölündüğü sürece bağımsız olarak işlenebilmektedir.

6.2 Tekli/Toplu İşleme Ödünleşimi

Bölüm 5.3.4'te bahsedilen GPU toplu işleme senaryosunun desteklenmemesi, geliştirilen hibrit kod çözücün tek limitasyonudur. Toplu işleme senaryosu, GPU kod çözücü özelinde birim iş parçası büyüklüğünü artırmakta ve bu sebeple granülerliği olumsuz yönde etkilemektedir. Hibrit kod çözücünün yük dengeleme performansı yüksek granülerliğe dayandığı için, toplu işleme senaryosu hibrit paralellikte faydadan çok zarara sebep olacaktır. Dolayısıyla hibrit kod çözücü tasarımında, Bölüm 5'de optimize edilen GPU kod çözücünün toplu işleme kabiliyeti haricindeki tüm özellikleri aynıdır.

Bununla beraber, Bölüm 6.3'de de açıkça görülebileceği üzere CPU ve GPU'dan aynı anda tam verimlilikte faydalanmanın sağladığı performans artışının yanında, GPU'da toplu işlemenin getirebileceği %1'lik kazanç (bkz. Bölüm 5.5) ihmal edilebilir seviyede kalmaktadır.

6.3 Performans Değerlendirmesi

Farklı CPU ve GPU özelliklerinin performansa etkisini görebilmek için, geliştirilen hibrit kod çözücü 3 farklı işlemci konfigürasyonunda test edilmiştir. Bu kısmın devamında bu işlemci konfigürasyonları, kullanılan test görüntüleri ve performans ölçümleri yer almaktadır.

6.3.1 Deney Konfigürasyonu

Ayrıca Bölüm 6.1'de belirtildiği gibi hibrit paralel kod çözücü performansı görüntünün yüksek granülerlikte olmasına dayandığı için, Bölüm 5.1'deki pankromatik test görüntülerine ek olarak, bu görüntülerin multispektral (kırmızı, yeşil, mavi ve kızılötesi) bantları da hibrit kod çözücünün performans testlerinde kullanılmıştır. Pankromatik görüntülere benzer şekilde bu multispektral bantların radyometrik çözünürlükleri 11 bit/piksel olmakla birlikte; mekansal çözünürlükleri 4096×4096 piksel, yani pankromatik görüntülerin dörtte biri kadardır. Dolayısıyla deneylerde kullanılan uydu görüntülerinin bir adet pankromatik bandı, büyüklük olarak diğer 4 multispektral bandın toplamına eşittir.

6.3.2 Performans Ölçümleri

Yapılan deneylerde 3 farklı işlemci konfigürasyonu için yalnızca CPU, yalnızca GPU ve hibrit kod kod çözücü kullanılarak 7 adet kare görüntü için toplam 5 adet bant (1 pankromatik ve 4 multispektral) işlenmiştir. Çizelge 6.1'de bu 3 işlemci konfigürasyonu için her bir kod çözücünün bu görüntüleri toplam ne kadar sürede işlediği gösterilmektedir.

Çizelge 6.1'in en sağdaki sütununda tüm görüntülerin işlenmesi için geçen toplam sürenin yanısıra, parantez içinde 50 MB büyüklüğündeki birim kare görüntü başına düşen işleme süresi de yer almaktadır. Bu sütunda, İMECE için Çizelge 1.1'de belirtilen yaklaşık 1.7 saniyelik birim görüntü indirme hızına eşit veya daha hızlı performans elde edilen ölçümler mavi renk ile gösterilmiştir. Burada da görülebileceği gibi hibrit kod çözücü, ikinci ve üçüncü işlemci konfigürasyonlarında İMECE görüntülerini gerçek zamanlı olarak işlemeye teorik olarak izin vermektedir. Üstelik RTX 2070'in kullanıldığı üçüncü konfigürasyonda yalnızca GPU kod çözücü kullanıldığında bile yeterli performansa ulaşmak mümkün olmuştur.

Çizelge 6.1: Kod çözücülerin farklı konfigürasyonlardaki işleme süreleri

Konfigürasyon	Kod Çözücü	İşleme Süresi (saniye)			
		Deney 1	Deney 2	Deney 3	Ortalama
Xeon E5-2637 v2 (4 core) GeForce GTX 1060	**CPU**	112.3	112.0	111.2	**111.8 (7.9)**
	GPU	63.1	62.7	62.2	**62.7 (4.5)**
	HİBRİT	47.9	47.3	47.7	**47.6 (3.4)**
Core i9-7940X (14 core) GeForce GTX 1060	**CPU**	26.8	27.0	26.8	**26.9 (1.9)**
	GPU	67.2	66.8	67.4	**67.2 (4.8)**
	HİBRİT	21.7	21.0	21.1	**21.3 (1.5)**
Xeon W-2145 (8 core) GeForce RTX 2070	**CPU**	39.4	39.8	39.8	**39.7 (2.8)**
	GPU	24.7	24.2	24.2	**24.4 (1.7)**
	HİBRİT	17.0	18.1	18.2	**17.8 (1.3)**

6.3.3 Ölçeklenebilirlik

Çizelge 6.1'de de görülebileceği üzere, sistemde kullanılan CPU ve GPU işlemcilerden birinin veya ikisinin birden iyileştirilmesiyle, herhangi bir yazılımsal konfigürasyona gerek kalmaksızın hibrid kod çözücü performansı doğrudan artırılabilmektedir. Bu durum hibrit kod çözücünün, Çizelge 1.1'de gösterilenden daha zorlu bir performans kısıtı ile karşılaşıldığında Bölüm 5.5.3'te değinildiği gibi daha yüksek CC seviyesine sahip bir GPU ve/veya daha iyi bir CPU kullanarak kolaylıkla ölçeklenebileceğini göstermektedir.

Tek bir sunucu veya iş istasyonunda daha iyi bir CPU veya GPU kullanmanın tek başına yeterli olamayacağı kadar zorlu performans kısıtlarıya karşılaşılması halinde, bu bölümde önerilen hibrit kod çözücünün bir sonraki bölümde önerilen dağıtık paralelliğe dayalı yöntemle birlikte kullanılması gerekecetir.

7. DAĞITIK JPEG 2000 KOD ÇÖZÜCÜ MİMARİSİ

Bundan önceki iki bölümde, JPEG 2000 kod çözücünün CPU, GPU ve hibrit paralel tasarımları ve ilgili optimizasyon yöntemleri konu alınmıştır. Bu bölümde ise, bu kod çözücülerden herhangi birinin her bir düğümde kullanılabileceği dağıtık bir mimari önerilmektedir. Bununla birlikte, bölümün devamında performans açısından daha avantajlı olması gerekçesiyle düğümlerde hibrit kod çözücünün kullanıldığı senaryo üzerinden gidilecektir.

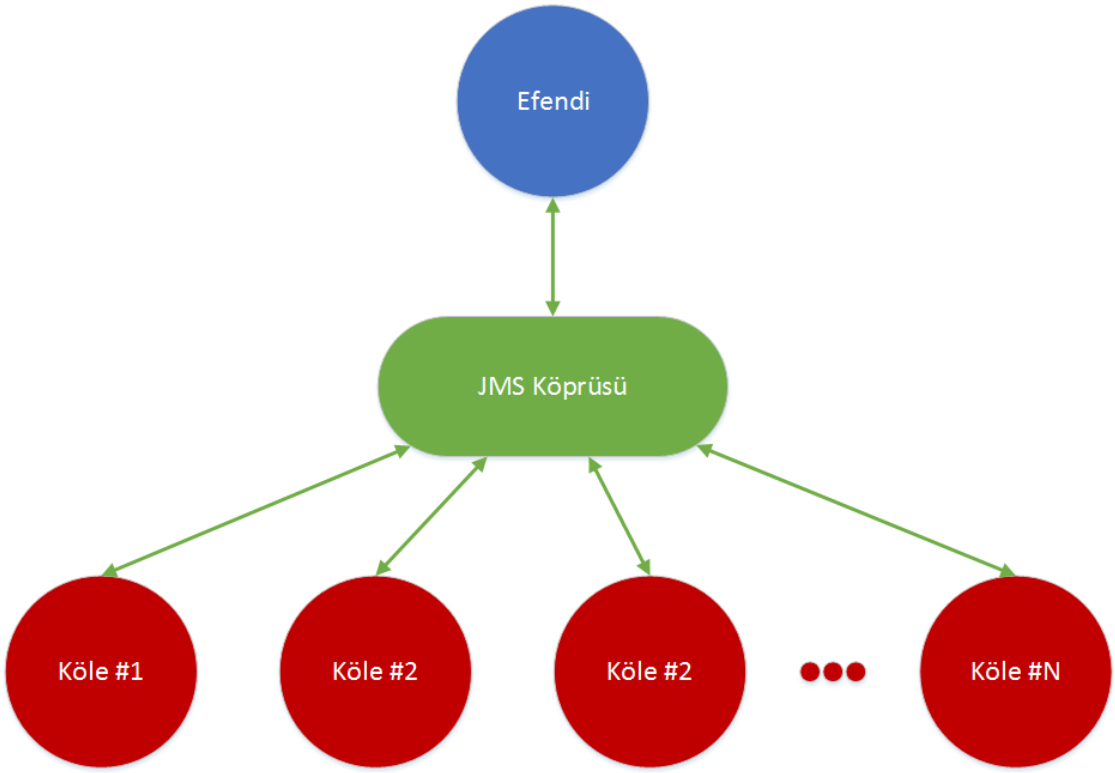
Bulut tabanlı dağıtık sistemlerin Bölüm 3.1.2'de bahsedildiği gibi birçok avantajı olsa da, GÖKTÜRK-2 ve İMECE gibi milli istihbari uydu projelerimizin güvenlik gereksinimlerine daha uyumlu olacağı düşünülerek, tez çalışması kapsamında lokal bir efendi-köle mimarisi önerilmiştir.

Bölüm 8.1'de ayrıca değinildiği gibi bu bölümde önerilen yöntemin kapsamı homojen düğümlerle sınırlı olmakla birlikte, gelecekte heterojen sistemleri de destekleyecek şekilde genişletilmeye açıktır. İlerleyen kısımlarda önerilen mimari topolojisi, kullanılan mesajlaşma altyapısı ve yük dengeleme mekanizması detaylandırılmaktadır.

7.1 Efendi-Köle Mimarisi

Önerilen efendi-köle mimarisinde, [28]'da tasarlanan tek seviyeli ağaç topolojisi baz alınmıştır. Baz alınan tasarımdan farklı olarak bu çalışmada Şekil 7.1'de gösterildiği gibi Java Messaging Service (JMS) tabanlı bir mesajlaşma altyapısı kullanılmıştır. Önerilen mimarinin tasarımı, aşağıda listelenen alt başlıklar halinde bu kısmın devamında detaylandırılmaktadır:

1. JMS Tabanlı Mesajlaşma Altapısı
2. Dağıtık Görüntü İşleme Prosedürü
3. Operasyon Esnekliği



Şekil 7.1: JMS tabanlı efendi-köle mimarisi

7.1.1 JMS Tabanlı Mesajlaşma Altyapısı

Java Message Service (JMS), uzak Java uygulamalarının birbirleri arasında TCP üzerinden kolaylıkla mesaj alışverişinde bulunmalarını sağlayan standart bir API sağlamaktadır [29]. Tez çalışması kapsamında JMS servis sağlayıcısı olarak Apache ActiveMQ Artemis[10] kullanılmıştır.

JMS yalın TCP haberleşmeye göre, özellikle eldeki mevcut dağıtık görüntü işleme problemi açısından bir takım avantajlar sağlamaktadır. JMS'in sağladığı avantajlar arasında en kayda değer olanlar aşağıdaki gibidir:

1. **Yayıncı-Abone ve Eşler Arası Haberleşme Olanağı:** Bazı uygulamalarda iki düğümün birbiriyle haberleşmesi gerekirken (eşler arası haberleşme), bazılarında bir düğümün ürettiği mesajları birden çok düğümün dinlemesi gerekebilmektedir (yayıncı-abone haberleşmesi). JMS bu iki durum için ayrı mesajlaşma paradigmaları sunmaktadır. Eşler arası haberleşmede düğümler kuyruk (queue) isimleri üzerinden haberleşirken, yayıncı-abone haberleşmesinde ise düğümler başlık (topic) isimleri üzerinden haberleşirler. Kuyruklara gönderilen mesajlar bir dinleyici tarafından alınana kadar korunmakta, alındığı zaman ise silinmektedir.

[10] https://activemq.apache.org/artemis/

Başlıklara gönderilen mesajlar ise birden fazla dinleyici tarafından alınabilmekte, ancak gönderildikten sonra korunmamaktadır. Böylelikle uygulamaların haberleşme gereksinimlerine göre düğümler birbirine kuyruk ve/veya başlık isimleriyle kolaylıkla bağlanabilmektedir. Bunun aksine yalın TCP'de yayıncı-abone haberleşmesi yapabilmek için, mesaj üreten düğümün tüm dinleyicilere tek tek mesaj göndermesi gerekmektedir.

2. **Gevşek Bağlaşım (Loose Coupling) ve Esneklik:** Birden fazla düğümün TCP üzerinden haberleşebilmesi için, normalde birbirlerine ait IP adreslerini bilmeleri gerekmektedir. Bu sıkı bağlaşım (tight coupling), düğümlerden birinin IP adresi değiştiği, yeni bir düğüm eklendiği veya düğümlerden birinin çıkarıldığı zaman konfigürasyon değişikliği yapmayı gerektirmektedir. JMS'te ise uygulamalar IP adresleri üzerinden değil; gevşek bağlaşımlı olarak kuyruk veya başlık isimleri üzerinden haberleşmektedirler. Sistemdeki JMS sağlayıcısı (örneğin Apache Artemis ActiveMQ) kuyruklara veya başlıklara mesaj gönderen ve bu kuyruklardan veya başlıklardan mesaj dinleyen düğümlerin IP adreslerini takip ederek, uygulamaların daha soyut adresler üzerinden haberleşmelerine izin vermektedir. Böylelikle kuyruk veya başlık isimleri sabit kaldığı sürece sisteme yeni düğüm eklemek, var olan bir düğümü çıkarmak veya bir düğümün IP adresini değiştirmek herhangi bir konfigürasyon değişikliğine sebep olmamakta, hatta bu değişiklikler dinamik olarak yapılabilmektedir.

3. **Java Nesnelerinin Serileştirilmesi:** Yalın TCP'nin aksine, JMS üzerinden Java nesneleri transfer edilebilmektedir. Uygulama seviyesinde kullanılan Java nesneleri otomatik olarak serileştirilerek, herhangi bir primitif veri tipine dönüştürmeye gerek kalmadan olduğu gibi gönderilip alınabilmektedir. JMS'in bu yeteneği, uygulamalar arası mesajlaşma arayüzünü önemli ölçüde basitleştirmekte ve özellikle nesne tabanlı uygulamaların birbirleriyle haberleşebilmesi için gereken fazladan programlama yükünü neredeyse sıfıra indirmektedir.

4. **Güvenilirlik:** Eşler arası JMS haberleşmede mesajlar kuyruklara gönderilmekte ve bir dinleyici tarafından alınana kadar korunmaktadır. Böylelikle olası bir haberleşme kopukluğundan dolayı dinleyici düğüm geçici olarak çevrim dışı kalsa bile tekrar çevrim içi olduğu zaman kendisine gelen mesajı alabilmektedir. JMS'in kendiliğinden sağladığı bu mesaj güvenilirliğini yalın TCP haberleşmede gerçeklemek fazladan efor gerektirmektedir.

7.1.2 Dağıtık Görüntü İşleme Prosedürü

Şekil 7.2'da akış diyagramı gösterilen dağıtık görüntü işleme süreci, aşağıdaki 4 adımdan meydana gelmektedir:

1. **Köle Keşfi:** Her bir görüntü işleme görevi için ilk adım, sistemdeki mevcut köle düğümlerini belirlemek ve bu düğümlere dair ilgili kuyruk isimlerini toplamaktır. Bu amaçla efendi düğüm ilk adım olarak tüm köle düğümlerin dinleyebileceği bir JMS başlığına keşif mesajı yayınlar. JMS altypısına bağlı ve hazır olan köle sunucular, bu mesaja cevaben efendi düğümün dinleyici kuyruğuna kendi kuyruk isimlerini gönderir. Böylelikle efendi düğüm toplam köle sayısını öğrenerek, mevcut görüntü işleme görevini Bölüm 7.2'de anlatılan yönteme göre sistemdeki köle düğümlere bölüştürür. Köle keşif mekanizmasının en önemli faydalarından biri, sisteme dinamik olarak köle düğümü ekleme ve çıkarmayı oldukça kolaylaştırmasıdır.

2. **Görüntü Okuma:** Uydudan indirlen ve işlenen görüntülerin, tüm düğümler tarafından ağ üzerinden ortak erişilebileceği bir yerde tutulacağı varsayılmaktadır. Bir önceki adımda gerçekleşen yük dağılımı sonucunda efendi düğüm tüm köle düğümlere birer görüntü okuma talep mesajı göndererek, sorumlu oldukları görüntü parçalarının lokasyonlarını bildirir. Köle düğümler talep edilen görüntü parçalarını okuduktan sonra efendi düğüme görüntü okuma onay mesajı gönderirler.

3. **JPEG 2000 Çözme:** Efendi düğüm, görüntü okuma onayı gönderen köle düğümlere JMS üzerinden görüntü işleme talebi gönderir. Talebi kabul eden köle düğümler, efendi düğüme görüntü işleme onay mesajı gönderir ve hibrit JPEG 2000 kod çözme işlemine başlar. Kod çözme işlemi tamamlandıktan sonra köle düğümler, efendi düğüme JMS üzerinden görüntü işleme cevap mesajı gönderirler.

4. **Görüntü Yazma:** Efendi düğüm, görüntü işleme cevap mesajı gönderen köle düğümlere görüntü yazma talebi gönderir. Bu talebi alan köle düğümler görüntüyü efendi düğüm tarafından talep edilen lokasyona yazar ve JMS üzerinden efendi düğüme görüntü yazma onay mesajı gönderir.

7.1.3 Operasyon Esnekliği

Uydu görüntü işleme sürecinde, görüntüler depolama birimine yazılmadan önce JPEG 2000 kod çözme adımına ek olarak Bölüm 2.1.4'de bahsi geçen diğer görüntü işleme

operasyonları sırasıyla gerçekleştirilmektedir. Dolayısıyla Bölüm 7.1.2'de anlatılan akış özellikle birden fazla adıma bölünmüş ve arka arkaya birden fazla görüntü işleme operasyonunu tek bir okuma ve yazma işlemi ile esnek olarak gerçekleştirmeye elverişli bir şekilde tasarlanmıştır.

Örnek vermek gerekirse, JPEG 2000 kod çözme adımından önce şifre çözmek için ayrı bir adım eklenebilir. Benzer şekilde kod çözme sonrasında radyometrik düzeltme için ayrı bir adım eklenebilir. Sonuç olarak köle düğümler tek bir dosya okuma işlemi yapmış olacak ve tüm görüntü işleme adımlarından sonra tek bir dosya yazma işlemi yapmış olacaktır. Bu sayede birden fazla dosya okuma ve yazmadan kaynaklanabilecek gereksiz kaynak israfının önüne geçilebilecektir.

Şekil 7.2: Dağıtık görüntü işleme sıralama diyagramı

7.2 Yük Dengeleme Mekanizması

Bölümün başında da belirtildiği gibi bu çalışmanın kapsamı, aynı CPU ve GPU işlemcilere sahip düğümler içeren homojen sistemlerle sınırlıdır ve bu nedenle heterojen bir sisteme kıyasla yük dengeleme problemi daha basittir.

Bölüm 6'de detaylandırılan hibrit kod çözücünün getirdiği hızlanmadan faydalanabilmek için, düğümlere çok sayıda görüntü parçası atamak gerekmektedir. Dolayısıyla efendi düğümün mevcut görüntüleri küçük parçalar halinde köle düğümlere dağıtması optimal bir çözüm sağlamayacaktır. Bunun yerine, bu çalışmada köle düğümlere en küçük birim olarak bir kareye dair tüm bant ve sensör verilerinden oluşan görüntü öbeklerinin dağıtıldığı bir metodoloji önerilmektedir. Bölüm 2.2'de verilen örnekten yola çıkıldığı durumda, bu metolojiye göre her bir düğüme 30 adet birim görüntü parçası dağıtılmış olacaktır. Köle düğümlere gönderilen görüntü öbekleri büyük bir şeridin parçaları olabileceği gibi, farkı görevlere ait birden fazla kare görüntüyü köle düğümlere dağıtmak da mümkündür. Böylelikle N adet köle düğüm içeren bir sistemde N adet görüntü öbeğini, hibrit kod çözücü tarafından yalnızca 1 görüntü öbeğinin işlenmesi için gereken sürede işlemek mümkün hale gelmektedir. Elbette mevcut öbek sayısının N'den az olduğu durumlarda bazı köle düğümler boşta beklemesi kaçınılmaz olacaktır.

7.3 Ölçeklenebilirlik

GPU kod çözücü için Bölüm 5.5.3'te ve hibrit kod çözücü için Bölüm 6.3.3'te anlatıldığı gibi, bu bölümde önerilen dağıtık kod çözme mimarisinin de yüksek performans gereksinimlerine uyumlu olarak ölçeklendirilmesi mümkündür. Dağıtık mimarilerde en temel ölçeklendirme yöntemi olan sisteme yeni düğümler ekleme yöntemine başvurulabileceği gibi, düğümlerdeki CPU ve GPU'ları iyileştirmek de genel performansı artıracaktır.

Daha yüksek performans elde etmek adına düğümleri güncellemek ile yeni düğümler eklemek arasındaki tercih, hedeflenen performans seviyesi, sistem bakım ve konfigürasyon maliyetleri ve işlemci maliyetleri göz önünde bulundurularak yapılmalıdır. Önerilen dağıtık mimarinin esnek bir şekilde düğüm eklemeye izin verir yapıda olması sayesinde, düğüm sayısının artırılması konfigürasyon maliyetini önemli ölçüde artırmayacaktır. Yedeklilik açısından da fayda sağlayacağı düşünülürse, düğümlerdeki işlemcilerin güncellenmesi yerine sisteme yeni düğümler eklemenin çoğu durumda daha kolay ve ucuz bir çözüm olacağı öngörülmektedir.

8. SONUÇ

Bu tez kapsamında yer gözlem uydu operasyonlarının verimliliği göz önünde bulundurularak, JPEG 2000 kod çözme sürecinin yer istasyonlarında çeşitli seviyelerde hızlandırılmasına yönelik yöntemler ve mimariler önerilmiştir. Hem sivil hem de istihbari uydu operasyonlarında potansiyel olarak karşılaşılabilecek koşullar ve kısıtlar göz önünde bulundurulmuştur. Aşağıdakiler bu gibi koşul ve kısıtlar karşısında, tez çalışması kapsamında sunulan çözümlere örnek olarak gösterilebilir:

- En az 5 yıllık görev ömrü hedeflenerek tasarlanan uyduların görüntü sıkıştırma modüllerinde fırlatmadan sonra değişiklik yapılamayacağı göz önünde bulundurularak JPEG 2000 standardına sadık kalınmıştır.
- Yine uzun görev ömrü boyunca gerçekleşebilecek yer istasyonu bakım ve yenileme faaliyetleri göz önünde bulundurularak;
  - Önerilen dağıtık mimari için standart TCP haberleşme yerine, spesifik IP adreslerine bağımlı olmayan ve düğüm ekleme-çıkarma işlemlerini dinamik ve esnek bir şekilde yapmayı kolaylaştıran bir JMS haberleşme altyapısı tercih edilmiştir.
  - Hibrit kod çözücünün yük dengeleme mekanizması, sistemdeki CPU ve/veya GPU işlemcilerin değişmesi nedeniyle herhangi bir konfigürasyona veya yazılımsal değişikliğe gerek kalmayacak şekilde tasarlanmıştır.
- Haberleşme hatalarından dolayı meydana gelebilecek veri kayıplarını minimize edebilmek için uydu görüntülerinin parçalı halde indirilmesinden faydalanılarak, önerilen hibrit ve dağıtık kod çözücüler yüksek granülerliğe dayalı bir şekilde tasarlanmıştır.
- Önerilen dağıtık mimari, kod çözme haricindeki uydu görüntü işleme aşamalarının herhangi bir verimlilik kaybına yol açmadan sisteme kolayca dahil edilmesine izin verecek şekilde tasarlanmıştır.
- Önerilen yöntem ve mimarilerin, yeni bir uydu olan İMECE'de teorik olarak gerçek zamanlı görüntü işlemeyi mümkün kıldıkları gibi, gelecekte üretilebilecek daha yüksek haberleşme hızına sahip uydular için de gerçek zamanlı görüntü işlemeye yönelik kolayca ölçeklenmeye elverişli olmaları sağlanmıştır.

Tez çalışmasının en önemli bölümünü JPEG 2000 kod çözücünün GPU için CUDA optimizasyon yöntemleriyle hızlandırılması olmuştur. JPEG 2000'in hesapsal karmaşıklığı en yüksek olan ve SIMT mimarisi ile tam anlamyıla uyumlu olmayan EBCOT adımı, GPU ile CPU'ya oranla çok yüksek oranlarda hızlanma elde edilmesine engel teşkil etmektedir. Bununla birlikte, önerilen optimizasyon yöntemleri sonucunda geliştirilen GPU kod çözücünün CPU'ya oranla 1.8 kat daha hızlı çalıştığı görülmüştür. CPU ve GPU işlemcilerin JPEG 2000 kod çözme performansları arasında muazzam bir fark olmayışından yola çıkarak geliştirilen hibrit kod çözücü, sistemdeki kaynakların boşta beklemesini engelleyerek, performans bakımından her iki kod çözücüyü de geçmiştir.

Sonuç olarak, tez kapsamında tipik bir yer istasyonu sistemi için JPEG 2000 kod çözme sürecini oldukça hızlandıracak yöntem ve mimariler önerilmiştir. Elbette genel operasyon verimliliği, ancak uydu görüntü işleme sürecindeki diğer aşamaların da hızlandırılmasıyla mümkün hale gelecektir. Bu tez çalışmasının, diğer uydu görüntü işleme adımlarının optimizasyonu konusunda bir referans ve bakış açısı kazandıracağı öngörülmektedir.

8.1 Gelecekteki Çalışmalar

Tez kapsamında önerilen üç ana modelin performansını, gelecekte yapılabilecek ek çalışmalarla iyileştirmek mümkündür. İlerleyen kısımlarda, her bir modele yönelik olarak yapılabilecek çalışmalardan ayrı ayrı bahsedilmektedir.

8.1.1 Heterojen Dağıtık Mimarilerin Desteklenmesi

Tez kapsamında öncelikli olarak çalışılan konular JPEG 2000 kod çözme algoritması için GPU optimizasyonları ve bu algoritmanın CPU-GPU heterojen paralelleştirmesidir. Tez çalışmasının üçüncü kısmı olan dağıtık yük dengeleme mekanizması ise ilk iki kısmı tamamlayıcı niteliktedir. Bu kısımda işlemci performanslarının değişiklik gösterebileceği heterojen sistemler değil; yalnızca homojen sistemler ele alınmıştır.

Literatürde dağıtık mimarilerin yük dengeleme mekanizmalarını optimize etmeye yönelik çeşitli çalışmalar mevcuttur. Divisible Load Theory (DLT), yüksek hacimlere sahip görüntü verilerinin dağıtık bir sistemdeki işlemcilere optimal bir şekilde dağıtılmasını modelleyen bir paradigmadır [30]. Ağdaki veri transferinden kaynaklanan gecikmelerin ve heterojen işlemcilerden oluşan sistemlerin göz önünde bulundurulduğu Partitioning and Scheduling Strategy [28] ve işlemci belleklerinin sınırlı olabildiği durumları optimize etmeye yönelik geliştirilen Incremental Balancing Strategy [31], DLT'ye dayanan

yük dengeleme metodolojilerine örnek olarak gösterilebilir. Gelecekteki çalışmalarda literatürde mevcut olan bu yaklaşımlardan faydalanarak, tez kapsamında geliştirilen dağıtık yük dengeleme sisteminin iyileştirebileceği öngörülmektedir.

8.1.2 Dağıtık Mimariye Diğer Görüntü İşleme Adımlarının Eklenmesi

Gelecekte konu hakkında yapılabilecek bir diğer çalışma, önerilen dağıtık sisteme JPEG 2000 kod çözme işleminin yanısıra Bölüm 2.1.4'te bahsi geçen diğer uydu görüntü işleme adımlarının da dahil edilmesi olabilir. Bölüm 7.1.3'de değinildiği gibi, görüntü işleme prosedürü esnek olarak görüntü işleme adımları eklemeye elverişli olacak şekilde tasarlanmıştır. Böylelikle bu yapıya bağlı kalınarak uydu görüntülerinin seviyelendirmesi tek bir okuma ve tek bir yazma işlemi ile sınırlandırılabilecektir.

8.1.3 Çoklu GPU Mimarilerinin Kullanımı

Bölüm 3.2.4'de bahsedildiği gibi tek bir host makinede birden fazla GPU'nun birlikte tam kapasiteyle kullanılması mümkündür. Bölüm 5'te önerilen GPU optimizasyon yöntemlerinin yeterli olmadığı durumlarda Bölüm 6 ve Bölüm 7'de önerilen hibrit ve dağıtık mimarilere başvurulabileceği gibi, alternatif olarak çoklu GPU mimarileri de performans ölçekleme için bir çözüm olarak kullanılabilir.

Buna ilave olarak hibrit ve dağıtık mimarilerin bir parçası olarak çoklu GPU içeren host makinelerin kullanılması da gelecekte yapılabilecek çalışmalar arasındadır. Böyle bir çalışma sonucunda hem tez kapsamında önerilen tüm yaklaşımların sağladığı mevcut performans kazancından, hem de çoklu GPU mimarisinin sağlayacağı ekstra performans artışından aynı anda faydalanmak mümkün hale gelecektir.

KAYNAKLAR

[1] **NVIDIA Corporation**. (2018). CUDA C Programming Guide. `https://docs.nvidia.com/cuda/cuda-c-programming-guide`.

[2] **Skodras, A., Christopoulos, C. and Ebrahimi, T.** (2011). The JPEG 2000 still image compression standard. *IEEE Signal processing magazine*, 18(5): 36–58.

[3] **McCool, M., Reinders, J. and Robison, A.** (2012). Structured parallel programming: patterns for efficient computation. Elsevier.

[4] **Andra, K., Chakrabarti, C. and Acharya, T.** (2003). A high-performance JPEG2000 architecture. *IEEE Transactions on Circuits and Systems for video technology*, 13(3):209–218.

[5] **Chiang, J.S., Lin, Y.S. and Hsieh, C.Y.** (2002). Efficient pass-parallel architecture for EBCOT in JPEG2000. In *IEEE International Symposium on Circuits and Systems*.

[6] **Fang, H.C., Chang, Y.W., Wang, T.C., Lian, C.J. and Chen, L.G.** (2005). Parallel embedded block coding architecture for JPEG 2000. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(9): 1086–1097.

[7] **Sarawadekar, K. and Banerjee, S.** (2011). An efficient pass-parallel architecture for embedded block coder in JPEG 2000. *Transactions on Circuits and Systems for Video Technology*, 21(6):825–836.

[8] **Matela, J., Rusnak, V. and Holub, P.** (2011). Efficient JPEG2000 EBCOT context modeling for massively parallel architectures. In *Data Compression Conference (DCC)*, pages 423–432. IEEE.

[9] **Matela, J., Šrom, M. and Holub, P.** (2011). Low GPU occupancy approach to fast arithmetic coding in JPEG2000. In *International Doctoral*

Workshop on Mathematical and Engineering Methods in Computer Science, pages 136–145. Springer.

[10] **Lee, J.W., Kim, B. and Yoon, K.S.** (2014). CUDA-based JPEG2000 encoding scheme. In *16th International Conference on Advanced Communication Technology (ICACT)*, pages 671–674. IEEE.

[11] **Le, R., Bahar, I.R. and Mundy, J.L.** (2011). A novel parallel Tier-1 coder for JPEG2000 using GPUs. In *9th Symposium on Application Specific Processors (SASP)*, pages 129–136. IEEE.

[12] **Auli-Llinas, F., Enfedaque, P., Moure, J.C. and Sanchez, V.** (2016). Bitplane image coding with parallel coefficient processing. *Transactions on Image Processing*, 25(1):209–219.

[13] **Enfedaque, P., Aulí-Llinàs, F. and Moure, J.C.** (2017). GPU Implementation of bitplane coding with parallel coefficient processing for high performance image compression. *Transactions on Parallel and Distributed Systems*, 28(8):2272–2284.

[14] **Le R, Mundy JL, Bahar RI**. (2012). High performance parallel JPEG2000 streaming decoder using GPGPU-CPU heterogeneous system. In *23rd International Conference on Application-Specific Systems, Architectures and Processors*, pages 16–23. IEEE.

[15] **Ciznicki, M., Kurowski, K. and Plaza, A.J.** (2012). Graphics processing unit implementation of JPEG2000 for hyperspectral image compression. *Journal of Applied Remote Sensing*, 6(1):061507.

[16] **Ciżnicki, M., Kierzynka, M., Kopta, P., Kurowski, K. and Gepner, P.** (2014). Benchmarking JPEG 2000 implementations on modern CPU and GPU architectures. *Journal of Computational Science*, 5(2):90–98.

[17] **Wu, X., Li, Y., Liu, K., Wang, K. and Wang, L.** (2014). Massive parallel implementation of JPEG2000 decoding algorithm with multi-GPUs. In *Satellite Data Compression, Communications, and Processing X*, page 91240S. International Society for Optics and Photonics.

[18] **Teke, M.** (2016). Satellite image processing workflow for RASAT and Göktürk-2. *Journal of Aeronautics and Space Technologies*, 9(1):1–13.

[19] **Murthy, K., Shearn, M., Smiley, B.D., Chau, A.H., Levine, J. & Robinson, M.D.** (2014). SkySat-1: very high-resolution imagery from a small

satellite. In *Sensors, Systems, and Next-Generation Satellites XVIII,* page 92411E. International Society for Optics and Photonics.

[20] **Blanchet, G., Lebegue, L., Fourest, S., Latry, C., Porez-Nadal F., Lacherade, S. and Thıebaut, C.** (2012). Pleiades-HR innovative techniques for radiometric image quality commissioning. In *XXII ISPRS Congress*, volume 25.

[21] **Harrison, D.L.** (2011). A fast 2D image reconstruction algorithm from 1D data for the Gaia mission. *Experimental Astronomy*, 31(2-3):157.

[22] **Nurvitadhi, E., Sim, J., Sheffield, D., Mishra, A., Krishnan, S. and Marr, D.** (2016). Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC. In *26th International Conference on Field Programmable Logic and Applications (FPL)*. pages 1–4. IEEE.

[23] **Nurvitadhi, E., Sheffield, D., Sim, J., Mishra, A., Venkatesh, G. and Marr, D.** (2016). Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC. In *International Conference on Field-Programmable Technology (FPT)*, pages 77–84. IEEE.

[24] **Pauwels, K., Tomasi, M., Alonso, J.D., Ros, E. and Van Hulle, M.M**. (2012). A comparison of FPGA and GPU for real-time phase-based optical flow, stereo, and local image features. *Transactions on Computers*, 61(7): 999–1012.

[25] **Prajapati, H.B. and Vij, S.K.** (2011). Analytical study of parallel and distributed image processing. In *International Conference on Image Information Processing*, pages 1–6. IEEE.

[26] **Pereira, R., Azambuja, M., Breitman, K. and Endler, M.** (2010). An architecture for distributed high performance video processing in the cloud. In *3rd international conference on cloud computing*, pages 482–489. IEEE.

[27] **Park, I.K., Singhal, N., Lee, M.H., Cho, S. and Kim, C.** (2011). Design and performance evaluation of image processing algorithms on GPUs. *Transactions on Parallel and Distributed Systems*, 22(1):91–104.

[28] **Li, X.L., Veeravalli, B. and Ko, C.C.** (2003). distributed image processing on a network of workstations. *International Journal of Computers and Applications*, 25(2):136–145.

[29] **Hapner, M., Burridge, R., Sharma, R., Fialli, J. and Stout, K.** (2002). Java message service. *Sun Microsystems Inc., Santa Clara, CA*, 9.

[30] **Bharadwaj, V., Ghose, D. and Robertazzi, T.G.** (2003). Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6(1):7–17.

[31] **Li, X., Bharadwaj, V. and Ko, C.C.** (2000). Scheduling divisible tasks on heterogeneous single-level tree networks with finite-size buffers. In *Proceedings of the Joint Conference on Information Sciences*, volume 5, pages 285–288.

ÖZGEÇMİŞ

Ad-Soyad : Derviş Utku UFUK

Uyruğu : T.C.

Doğum Tarihi ve Yeri : 02.08.1990 Çankaya / Ankara

E-posta : utkuufuk@gmail.com

ÖĞRENİM DURUMU:

• **Lisans** : 2012, Hacettepe Üniversitesi, Elektrik-Elektronik Mihendisliği

MESLEKİ DENEYİM VE ÖDÜLLER:

Yıl	Yer	Görev
2019-Halen	TÜBİTAK UZAY	Uzman Araştırmacı
2014-2019	TÜBİTAK UZAY	Araştırmacı

YABANCI DİL: İngilizce, İspanyolca, Arapça

TEZDEN TÜRETİLEN YAYINLAR, SUNUMLAR VE PATENTLER:

- **Ufuk, D. U.**, Temizel, A., Özbayoglu, A. M. (2018, October). Optimized GPU Implementation of JPEG 2000 for Satellite Image Decompression. In 2018 IEEE International Conference on Computational Science and Engineering (CSE) (pp. 56-61). IEEE.

DİĞER YAYINLAR, SUNUMLAR VE PATENTLER:

- **Ufuk, D. U.**, Demirpolat, C., Demirci, M. F. (2017, May). Fast cloud detection using low-frequency components of satellite imagery. In Signal Processing and Communications Applications Conference (SIU), 2017 25th (pp. 1-4). IEEE.
- **Ufuk, D. U.**, Açikgöz, İ. S., Teke, M., Özbayoğlu, A. M. (2018, May). Satellite image band registration with Dynamic Time Warping and Discrete Wavelet Transform. In 2018 26th Signal Processing and Communications Applications Conference (SIU) (pp. 1-4). IEEE.