ONDOKUZ MAYIS ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

PARALEL PROGRAMLAMA İLE GÖRÜNTÜ İYİLEŞTİRME

SEMİNER RAPORU

Kadir Emre ÖZER

Danışman

Dr. Öğr. Üyesi Sercan DEMİRCİ

OCAK,2023

T E Ş E K K Ü R

C++ gibi bir dil tasarlayarak her noktalı virgülde bir tutam gülümsemeyle birlikte Dennis Ritchie'yi anımsatmasıyla ve *"It's easy to win forgiveness for being wrong; being right is what gets you into real trouble."* sözleriyle bilgisayar bilimleri tarihine ufak bir iz bırakan Bjarne Stroustrup'a teşekkürlerimi sunarım.

ÖZET

Bu çalışmada paralel programlamanın tanımı, koşutzamanlı programlama ile paralel programlamanın farkı, paralel programlama yaparken CPU-GPU arasındaki verim farkı ve bir görüntünün paralel şekilde nasıl işlenebileceği gibi konularda bilgi verilmiştir. Devamında ise bir paralel işleme teknolojisi olan Nvidia CUDA'dan ve açık kaynak kodlu bilgisayarlı görü kütüphanesi olan OpenCV'den bahsedilmiştir. Son olarak ise histogram eşitliği ve gamma düzeltmesi algoritmaları anlatılmış ve mimaride bu algoritmaların nasıl kullanılacağından bahsedilmiştir.

Anahtar kelimeler: Görüntü İşleme, Paralel Programlama, GPGPU, CUDA

İÇİNDEKİLER

ŞEKİLLER LİSTESİ

ÇİZELGELER LİSTESİ

BÖLÜM: I

GİRİŞ

1

AMAÇ

Bu çalışma kapsamında Grafik İşleme Birimi(GPU) üzerinde bulunan çok sayıda çekirdek kullanılarak Merkezi İşlemci Birimi(CPU)'ne kıyasla daha hızlı sonuç üreten bir görüntü iyileştirme algoritması için CUDA mimarisi tasarlamak amaçlanmaktadır.

2

LİTERATÜR ÖZETİ

2007 yılında Svetlin A. Manavski AES-128 ve AES-256 kriptolojik şifreleme algoritmalarını CUDA üzerinde gerçekleştirmeye çalışmış ve AES-128'de 19.6,AES-256'da 15.76 kat hızlanma olduğunu ölçeklemiştir. Bu ölçeklemelerde GPU ve CPU arasındaki veri aktarması için geçen süreler dahil edilmemiştir. Bu gecikmeler dahil edildiğinde hızlanmalar sırasıyla 5.92 ve 5.41 kat olarak ölçülmüştür[5].

2008 yılında Zhiyi Yang, Yating Zhu ve Yong Pu uçak ve uydu görüntüleri üzerinde histogram eşitliği, ayrık kosinüs dönüşümü, kenar çıkartma algoritmalarının paralel şekilde işlenebilmesi için CUDA ile çalışmış ve sırasıyla 37, 8 ve 79.4 kat hızlanmalar elde etmişlerdir. Bu ölçümleri yaparken verilerin GPU ve CPU arasındaki taşınması için geçen süreyi göz ardı etmişler ve hangi kenar çıkartma algoritmasını kullandıklarından bahsetmemişlerdir[6].

2010 yılında Daniel Strigl ve arkadaşları Çok Katmanlı Algılayıcı (Multi Layer Perceptron) bir standartı olan Evrişimsel Sinir Ağları (Convolutional Neural Networks)'nın yoğun hesaplamaya dayalı yönlerini GPU üzerinde CUDA ile paralel gerçekleştirmiştir[7]. Ağ topolojisinin eğitimine ve sınıflandırılmasına bağlı olarak 2-24 kat arası hızlanmalar olduğunu ölçmüşlerdir.

2011 yılında In Kyu Park ve arkadaşları çok görüntülü stereo eşleştirmesi, doğrusal özellik çıkartma, JPEG 2000 kodlama ve fotogerçekçi olmayan işleme (Nonphotographic Coding -

NPR) konuları üzerinde CUDA ile paralel programlama çalışmaları yürütmüşlerdir. Stereo eşleştirmesinde 160 kat, doğrusal özellik çıkartmada 3 kata kadar, JPEG2000 kodlamada 12.92 kata kadar, karikatür stili NPR'de 219 kata kadar, yağlı stil NPR'de 159 kata kadar hızlanmalar elde etmişlerdir[8].

2014 yılında Çekmez ve arkadaşları evrimsel algoritma çeşidi Genetik Algoritmalar (Genetic - GA) ve Karınca Kolonisi En İyilemesi (Ant Colony Optimisation) algoritmalarını CUDA üzerinde paralelleştirilerek bir insansız hava aracının rota planlamasında kullanılmasını amaçlamışlardır[9]. Yapılan denemelerde 113-1525 kat arası gibi çok yüksek hızlanmalar elde etmişlerdir. Deneme senaryolarından birinde 100 nokta ve 1024 karınca değerleri ile 1525 kat hızlanma ölçmüşlerdir.

2016 yılında Hattori ve arkadaşları bir metasezgisel algoritma olan Standart Parçacık Sürü En İyilemesi (Standard Particle Swarm Optimisation) algoritmasını CUDA ile paralelleştirmeye çalışmışlardır. Çalışmalar sonucu en yüksek 46.2 kat hızlanma elde edebilmişlerdir. Algoritma nın parametresi olan boyut sayısının sabit tutulup parçacık sayısının artırıldığı ve parçacık sayısının sabit tutulup boyut sayısının artırıldığı senaryolarda hızlanmanın azaldığı ölçmüşlerdir [10].

2016 yılında Xueqin Zhang ve arkadaşları 2012 yılında MIT tarafından geliştirilen Seyrek Hızlı Fourier Dönüşümü (Sparse Fourier Transform) algoritmasını CUDA ile paralelleştirmiş lerdir. Algoritmada 10 kat, CPU üzerinde paralel çalışan Fast Fourier Transform in The West(FFTW) algoritmasından 28 kat hızlanma elde edildiği raporlanmıştır[11].

2017 yılında Sayed ve Hossein Işın İzleme(Ray Tracing) algoritmasını CUDA ile paralel programlamaya çalışmışlardır. 1080p'lik bir görüntünün oluşturulmasında kullanılan bu algoritmanın GPU'da gerçeklenmiş haliyle CPU'ya göre 1.62 kat hızlanma elde etmişlerdir. 720p'lik bir görüntüde ise 1.48 kat hızlanma elde etmişlerdir[12].

2018 yılında Çınar ve Kıran popülasyon tabanlı bir metasezgisel algoritma olan Ağaç-Tohum Algoritması(Tree-Seed Algorithm-TSA)'nı paralel şekilde CUDA üzerinde işlemeye çalışmışlardır. Kıyas işlevi olarak Ackley kullandıklarında 123 kat, Griewank kullandıklarında 149 kat, Schwefel kullandıklarında 184 kat hızlanma ölçmüşlerdir[13].

2020 yılında Abir Al Sideiri ve arkadaşları fraktal sıkıştırma algoritmasının kodlama süresini indirgemek için CUDA ile paralel bir Fisher sınıflandırma şeması algoritması üretmeye çalışmış ve bazı görüntüler üzerinde 6.4 kata kadar hızlanma elde etmişlerdir[14].

2021 yılında Zhongju Wang ve arkadaşları rüzgar türbinlerinde, türbinin güç üretme performansının ölçülmesi için kullanılan Rüzgar Güç Eğrisi(Wind Power Curve-WPC) modelinin veri temizleme aşamasını CUDA ile paralelleştirmişlerdir. Denemeler birden çok ekran kartı üzerinde denenmiş ve en düşük 45 kat hızlanma elde edilmiştir[15].

3

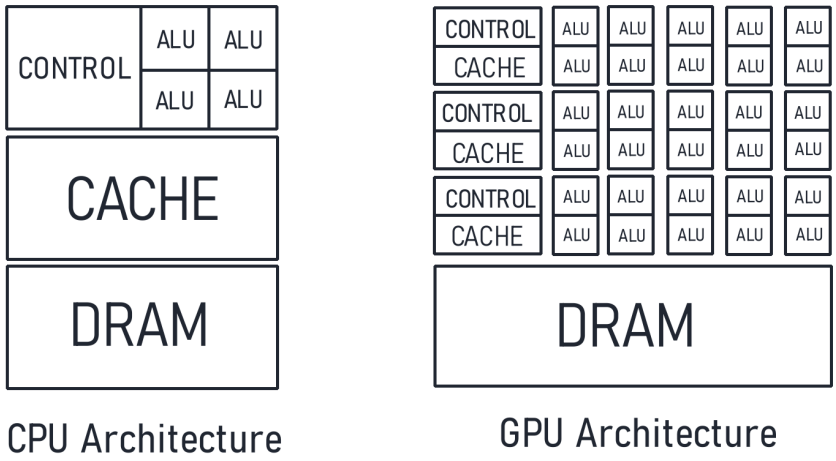
PARALEL PROGRAMLAMA

Paralellik en basit haliyle, birden fazla işin aynı anda icra edilmesi olarak tanımlanabilir. Paralel programlama ise büyük bir problemin kesin sınırlar ile küçük parçalara ayrılarak bu küçük parçaların paralel şekilde işlenmesine dayanır. Günümüzde paralel programlama oldukça yaygın bir yaklaşım olmakla beraber çokça da ihtiyaç duyulmaktadır. Gerek yüksek çözünürlüklü görüntülerin işlenmesi gerekse büyük verinin işlenmesi gibi yüksek iş gücü gerektiren işlemler için bir vazgeçilmeze dönüşmüştür. Bunların yanı sıra sinyal işleme, şifreleme ve sıralama algoritmaları için de oldukça kullanılabilir bir programlama yaklaşımıdır.

Paralellik yaklaşımı genellikle koşutzamanlılık (concurrency) ile karıştırılmaktadır. Bu iki yaklaşımın arasındaki temel fark mimarideki işlemci sayısıdır. Koşutzamanlılık, birden fazla işlemin tek bir işlemci üzerinde zaman paylaşımlı olarak yürütülmesi prensibine dayanır. Öte yandan paralellik ise aynı anda birden fazla işlemin birden fazla işlemci üzerinde yürütülebilmesidir. Günümüzde yaygın olarak tek işlemcili, çok çekirdekli mimariler tercih edilmektedir. Bu mimarilerde paralellik işlem (process) düzeyinde değil izlek (thread) düzeyinde sağlanabilmektedir. Yani bir iş, birçok alt işe bölünerek bu alt işler paralel şekilde izlekler halinde çekirdekler üzerinde icra edilebilmektedir.

3.1 *Paralel Programlamada GPU Kullanımı*

CPU çekirdekleri her ne kadar paralelleştirme desteği sağlasa da CPU'ların aynı anda çalıştırabileceği izlek sayısı GPU'lara göre oldukça düşük olmakla birlikte geliştirilme amaçları da paralellikten daha çok seri,denetimli ve düşük gecikmeli işlemeye yönelik olduğundan denetim bölütleri ve ön bellek miktarları GPU'ya göre bir hayli büyüktür. Öte yandan Şekil 1'de görüldüğü üzere GPU mimarileri CPU'ya kıyasla daha yüksek aritmetik işlem gücü, daha büyük bellek miktarı, daha küçük denetim birimleri ve daha küçük fakat çok sayıda ön bellekleri ile göze çarpmaktadır. CPU'nun düşük çekirdek miktarı ve seri işleme yeteneğine karşın GPU'nun yüksek aritmetik gücü ve yüsek çekirdek miktarı GPU'yu paralelleştirme için bir adım öne taşımıştır. Bu sebeplerden ötürü 2006'lı yıllardan başlayarak GPU'lar teknolojide daha fazla ihtiyaç duyulur birimler haline gelmiş ve CPU'lara kıyasla daha hızlı güç evrimi geçirmişlerdir.



Şekil 1: CPU ve GPU Mimarisi

3.2 *Paralel Programlanın Uygulanabilmesi*

Bir işlemin paralel şekilde uygulanabilmesi için en önemli koşul işlemin ayrıştırılabilmesidir. Yani bir işlemin parçaları birbirinden bağımsız bir şekilde yürütülebiliyorsa işlem, paralel işlemeye uygundur. Buna örnek olarak boyutu küçük olan bir dizinin elemanlarını toplama işlemi verilebilir. Belirli sayıda görevlendirilecek izlekler kendilerine ayrılan dizi elemanlarını toplar, tüm izlekler işini bitirdikten sonra elde edilen toplam değerleri birleştirilerek son değere ulaşılır. Bu örnekte olduğu gibi işlem paralelleştirmeye uygun olsa da her zaman bunu yapmak verimli değildir. Çünkü izlek oluşturmak maliyetli bir süreçtir. Eğer bu paralelleştirme işlemi GPU üzerinde yapılacaksa verilerin CPU'dan GPU'ya taşınması ve işlem bittikten sonra CPU'ya tekrar taşınması gibi zaman alan süreçler de mevcuttur. Bu sebeplerden dolayı paralel programlama her zaman olumlu sonuçlar doğuramayabilmektedir. Genellikle yüksek işlem gücü gerektiren veriler üzerinde paralel programlama yapılmaktadır.

3.3 *Paralelleştirme Yaklaşımları*

Temel olarak iki adet paralel programlama yaklaşımı vardır. Birincisi, işlemin birbirinden bağımsız altişlemlerinin paralel şekilde işlenmesi, diğeri ise veri üzerinde sıralı olarak çalışacak bir işleme verinin paralel olarak verilmesidir. Verinin paralel verilmesi yaklaşımının uygulanması bir bakıma daha kolay bir işlemdir.

3.4 *Paralel Görüntü İşleme*

Görüntü,küçük görüntü noktalarının (piksel-imgecik) iki boyutlu dağılımıdır. Bu tanımı biraz daha derinleştirirsek görüntü, x ve y parametrelerinin koordinatları temsil ettiği iki boyutlu bir $f(x,y)$ fonksiyonudur. Bu fonksiyonun geri döndereceği değer ise o noktadaki yoğunluktur ya da başka bir deyişle grilik seviyesidir[16]. Birçok görüntü işleme algoritması da bu imgeciklerin bir matris ile ya da skaler bir değerle çarpılmasından ibarettir. Böylelikle yapılmak istenen birçok işlem türlü sayıda izlek tarafından imgecik kümeleri üzerinde gerçekleştirilebilir. Bu ayrık işlenebilme yeteneğinden ve yoğun aritmetik işlem gereksinimin den dolayı görüntü işleme algoritmalarını GPU üzerinde paralel yürütmek daha mantıklı bir harekettir[17].

BÖLÜM: II

MATERYAL

4

NVIDIA CUDA MİMARİSİ

4.1 *Nvidia CUDA*

CUDA (Compute Unified Device Architecture), Nvidia tarafından ilk sürümü 2006 Kasım'da tanıtılmış birçok karmaşık hesaplama problemini Nvidia GPU'ları üzerinde gerçek leştirmek için tasarlanmış genel amaçlı paralel programlama ortamıdır. CUDA ortamı beraberinde yüksek seviyeli C, C++, Fortran ve Python gibi dillerin de içerisinde kullanılmasına izin vermektedir[1].

4.2 *CUDA Programlama Modeli*

4.2.1 *Çekirdekler(Kernels)*

CUDA C++ modeli, C/C++ özelliklerine GPU'yu yönetmek için birtakım eklentilerdir. Kullanıcının çekirdek denilen C++ işlevleri yazmasına izin verir. Bu çekirdek işlev, kullanılan parametrelere göre değişik sayıda bölüt içerisinde değişik sayıda CUDA izleği tarafından aynı anda yürütülebilir[1]. CUDA C++ modelinde bir işlevin GPU tarafından icra edileceğini

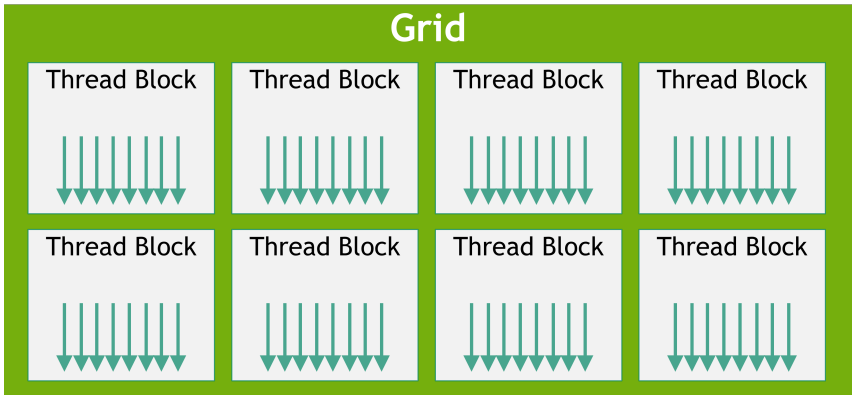
bildirmek için işlevin geri döndüreceği değerden önce *__global__* bildirimi yapılmalıdır. Bu çekirdek işlev <<< B,T>>> şeklinde bir ayarla çağrıldığı zaman B sayıda bölüt ile herbirinin içerisinde T sayıda izlek tarafından, toplamda B*T kez yürütülecektir.

4.2.2 *İzlek Sıradüzeni(Thread Hierarchy)*

Bir çekirdek işlev çağırıldığında bu işlevin parametrelerinin oluşturduğu ızgara, bölüt ve izlek sıradüzeninin programcı tarafından nasıl öğrenilebileceği hakkında birtakım değişkenler bulunmaktadır.

threadIDx : Bölüt içerisinde izleğin sıra numarasıdır. Üç boyutlu bir değişkendir. Böylelikle izlekler istenirse bir, iki ve üç boyutlu olarak tanımlanabilir. Bu üç boyutlu yapı vektör, matris ve 3B yapılar üzerinde işlem yaparken esneklik ve kolaylık sağlar. Boyutlarına threadIDx.x, threadIDx.y, threadIDx.z şeklinde erişilebilir[1].

Bölütler bir boyutlu veya iki boyutlu olarak tanımlanabilmektedir[1]. Bir bölütün destek sağlayabileceği izlek sayısı günümüz Nvidia GPU'larında 1024 ile sınırlıdır. Eğer programcı tarafından bu sınır ihlal edilirse bölüt çalışmayacaktır. Şekil 2'de iki boyutlu bir **ızgara** yapısı gösterilmiştir.



Şekil 2: 2 Boyutlu Bölüt[1]

blockIDx : Izgara içerisinde bölütün sıra numarasıdır. Üç boyutlu bir değişkendir. Boyutları na blockIDx.x, blockIDx.y ve blockIDx.z şeklinde erişilebilir[1].

blockDim : Bölütün derinliğini(barındırdığı izlek sayısı) tanımlar. Üç boyutlu bir değişkendir. İstenilen boyuttaki izlek sayısına blockDim.x, blockDim.y ve blockDim.z şeklinde erişilebilir[1]. Bölütleri kapsayan yapı ise ızgaradır. Her çekirdek işlev çağrılışında bir ızgara oluşturulur. Devamında gelen parametrelere göre bölütler ve bölüt başına izlekler oluşturulur[1].

gridDim : Izgara derinliğini (barındırdığı bölüt sayısı) tanımlar. Üç boyutlu bir değişkendir. Boyutlarına gridDim.x, gridDim.y, gridDim.z şeklinde erişilebilir. Izgara yapısı üç boyutlu olmadığı için gridDim.z şeklinde kullanım yapılmamaktadır[1].

4.2.3 *Eşsiz İzlek Sıra Numarası*

Bir bölüt içerisinde oluşturulan izlekler eğer tek boyutlu iseler burada threadIDx değeri izlekleri numaralandırmak için yeterli olacaktır. Eğer bölüt iki boyutlu ise burada sadece izlek sıra numaraları kullanmak yeterli olmayacaktır. Bu sorunu çözmek için Eşitlik 1'deki gibi bölüt özellikleri de kullanılmalıdır. Buradaki x, y, D_x değerleri sırasıyla izleğin satır numarası, izleğin sütun numarası ve blockDim.x'e karşılık gelmektedir[1].

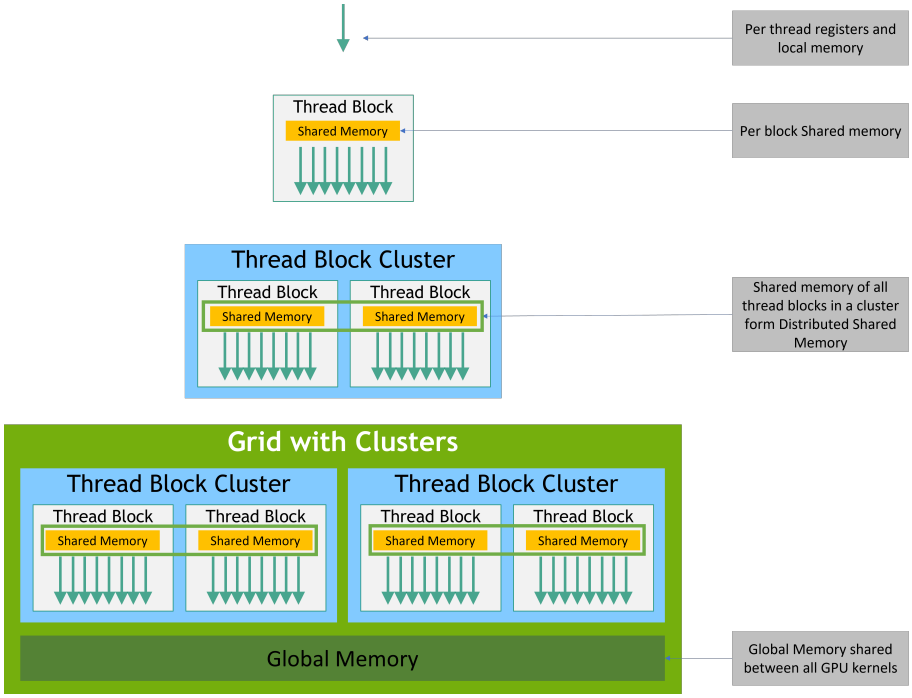
$$2BIzlekSiraNumarasi = x + y * D_x \tag{1}$$

Eğer üç boyutlu bir bölüt üzerinde numaralandırma yapılacaksa Eşitlik 2 kullanılmalıdır. x, y, z, D_x, D_y değerleri sırasıyla izleğin birinci boyut sıra numarası, izleğin ikinci boyut sıra numarası, izleğin üçüncü boyut sıra numarası, blockDim.x ve blockDim.y'ye karşılık gelmektedir[1].

$$3BIzlekSiraNumarasi = x + y * D_x + z * D_x * D_y \tag{2}$$

4.2.4 *Cuda Bellek Sıradüzeni*

Şekil 3'de görüldüğü üzere CUDA izleklerinin veriye birçok bellekten erişmesi mümkündür. Her bir izleğin kendisine ait yerel belleği ve yazmaçları; her bölütün,içerisindeki tüm izleklerin erişebileceği bir yerel belleği; çekirdekdeki tüm izleklerin erişebileceği evrensel (global) bellek vardır. Bölütün sahip olduğu bu belleğin yaşam süresi bölütün ömrü ile sınırlıdır. Bölüt kümelerindeki izlekler birbirlerinin bölüt belleklerine erişebilirler. Ayrıca tüm izleklerin erişebileceği doku (texture) ve sabit (constant) bellekleri bulunmaktadır[1].

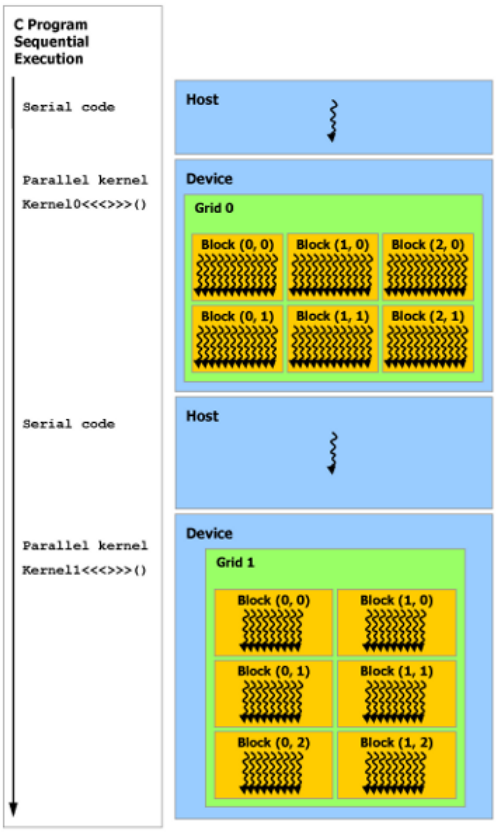


Şekil 3: CUDA Bellek Sıradüzeni[1]

4.2.5 *Ayrışık Programlama(Heterogeneous Programming)*

CUDA programlama modeli CUDA izleklerinin CPU'dan farklı, CPU'ya yardımcı bir birimde yürütüleceğini varsayar. CUDA programlama modeli ayrıca CPU'nun ve GPU'nun

kendilerine ait ayrı bir bellek alanları olduğunu varsayar. Böylelikle bir uygulama CUDA fonksiyonlarını kullanarak evrensel, sabit ve doku belleklerine erişebilir, CPU'da bellek ayırabilir, GPU'da bellek ayırabilir, GPU ve CPU arasında bellek aktarması yapabilir[1]. Ayrışık programlama mantığında seri kodların CPU'da, paralel kodların ise GPU'da çalıştırılması beklenmektedir.



Şekil 4: Ayrışık Programlama[1]

4.2.6 *Hesaplama Yeteneği(Compute Capability)*

Hesaplama yeteneği ya da hesaplama sürümü, GPU'nun işlem yeteneğini belirten sürüm numarasıdır.Akış Çokluişlemcisi(Streaming Multiprocessor - SM)'nin sürüm numarası olarak da bilinmektedir[1]. Örneğin bir Turing mimarisine sahip Nvidia GTX 1660TI ekran kartının

hesaplama kapasitesi 7.5'dir. Bu sürüm numarasındaki noktadan önceki sayı (7) ana sürümü (major version), noktadan sonraki sayı (5) ise küçük sürümü (minor version) tanımlar. Aynı ana sürüm numarasına sahip olan kartlar aynı çekirdek yapısına sahiptirler. Küçük sürümler ise çekirdek mimarisinde olası yeni özellikler ile birtakım iyileşmeleri temsil eder. Hesaplama Yeteneği 7.5'a ait CUDA özellikleri Çizelge 1'de belirtilmiştir.

Warp Başına İzlek	32
SM Başına En Fazla Warp	32
SM Başına En Fazla Bölüt	16
SM Başına En Fazla İzlek	1024
En Büyük Bölüt Boyutu	1024
SM Başına Yazmaç	65536
Bölüt Başına En Fazla Yazmaç	65536
İzlek Başına En Fazla Yazmaç	255
SM Başına Paylaşımlı Bellek (bayt)	65536
Bölüt Başına En Fazla Paylaşımlı Bellek	65536
Yazmaç Ayırma Birim Boyutu	256
Yazmaç Ayırma Düzeyi	warp
Paylaşımlı Bellek Ayırma Birim Boyutu	256
Warp Ayırma Düzeyi	4
Bölüt Başına Paylaşımlı Bellek Boyutu (bayt) (CUDA Runtime Gerekli)	0

Çizelge 1: Hesaplama Yeteneği 7.5 Özellikleri

4.3 *CUDA Programlama Arayüzü*

CUDA C++, geleneksel C/C++ programcılarına GPU üzerinde çekirdek işlevler yazabilme leri için kolay bir kullanım sağlar. Küçük bir C++ eklentisi ve CUDA Runtime kütüphanesi desteği ile programcının C++ işlevleri gibi çekirdek işlev yazmasına, çekirdek işlevi çağırırken basit bir bölüt ve izlek sayısı belirtimesine izin verir. Bu tarz CUDA eklentileri içeren kaynak kodlar Nvidia CUDA Derleyicisi (Nvidia CUDA Compiler - nvcc) ile derlenmek zorundadır[1].

4.3.1 *NVCC ile Derleme*

CUDA çekirdek işlevleri hem C/C++ ile hem de PTX (Parallel Thread Execution) denilen CUDA buyruk kümesi mimarisi kullanılarak yazılabilir. CUDA programlama yapılırken genellikle PTX yerine daha etkili ve güvenilir bir seçenek olan C/C++ gibi yüksek seviyeli bir dil tercih edilmektedir. Her iki seçenekte de çekirdek işlev nvcc tarafından ikilik koda dönüştürülmektedir[1].

5

OPENCV

OpenCV (Open Source Computer Vision) ilk sürümü Haziran 2000'de BSD lisansı altında yayınlanmış olan açık kaynak kodlu bir bilgisayarlı görü kütüphanesidir. BSD lisansından ötürü akademik ve ticari kullanımında herhangi bir sınır yoktur. Kütüphanesi C/C++ ile yazılmıştır. Öncelikli olarak C++ devamında ise Python, Java, Matlab gibi dillere arayüz sağlamaktadır. Linux, Windows ve Mac OS X işletim sistemlerinde çalışmaktadır[18].

OpenCV kütüphanesi yüksek verimlilik ve gerçek zamanlı uygulama desteği sağlamayı amaçlayarak tasarlandığı için kaynak kodu oldukça iyilenmiş ve çok çekirdekli işlemciler üzerinde çalışmaya uygun tasarlanmıştır[18].

OpenCV'nin bir amacı da basit bir kullanım sunarak gelişmiş görüntü işleme uygulamala rının hızlı bir şekilde oluşturulmasına imkan sağlamaktır. Tıbbi görüntüleme, nesne yakalama, video yakalama, özellik çıkartma, kullanıcı arayüzü vb. gibi alanlarda 2500'den fazla işlev sunar[18].

BÖLÜM: III

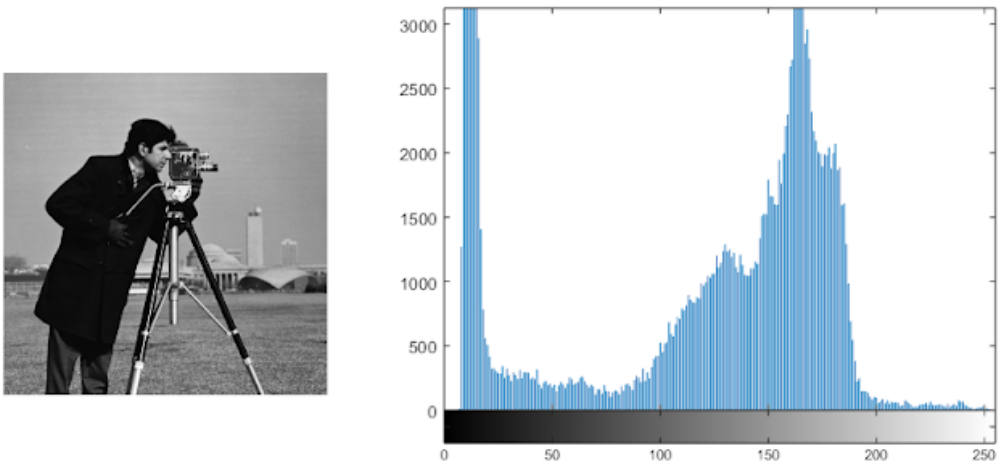
YÖNTEM

6

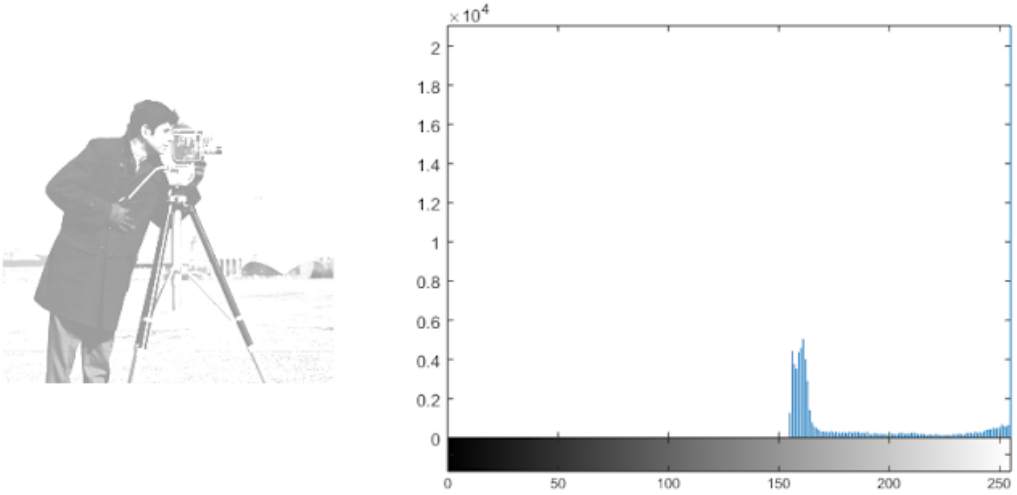
KARANLIK GÖRÜNTÜNÜN İYILEŞTIRILMESI

6.1 *Histogram Hesaplama*

Histogram, bir işlem sonucunda elde edilen sonuçların veya verilerin bir dağılımını gösteren çizgedir. Görüntü işlemede kullanımı ise bir görüntüdeki herhangi bir yoğunluk (renk) dağılımının ölçülmesidir. Başka bir deyişle herhangi bir renk değerini taşıyan kaç tane imgecik olduğu hesaplanır. Şekil 5'te ve Şekil 6'da aynı görüntünün farklı parlaklık değerleri ile histogram çizgesi görülmektedir.



Şekil 5: Normal Görüntü ve Histogram Çizgesi[2]



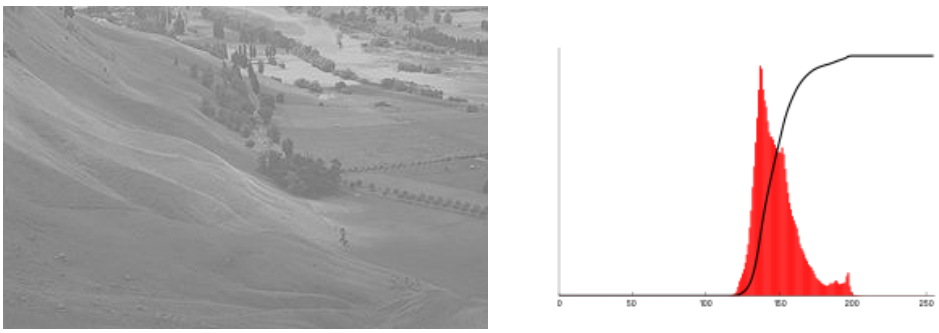
Şekil 6: Aydınlık Görüntü ve Histogram Çizgesi[2]

Yukarıda görüldüğü üzere,parlak olan Şekil 6'da histogram değerleri 255 (beyaz) değerine oldukça yakınken Şekil 5'te paltodaki katı siyah tondan 0 (siyah), resmin genelindeki gri tonlardan dolayı 160 değerlerinde yoğun bir dağılım görülmektedir. Burada kullanılan fotoğraflar tek kanallı (siyah-beyaz) fotoğraflardır. Eğer üç kanallı (RGB) bir görüntü üzerinde histogram ölçümü yapılmak isteniyorsa herbir kanal için ayrı ayrı histogram ölçümü yapılmalıdır.

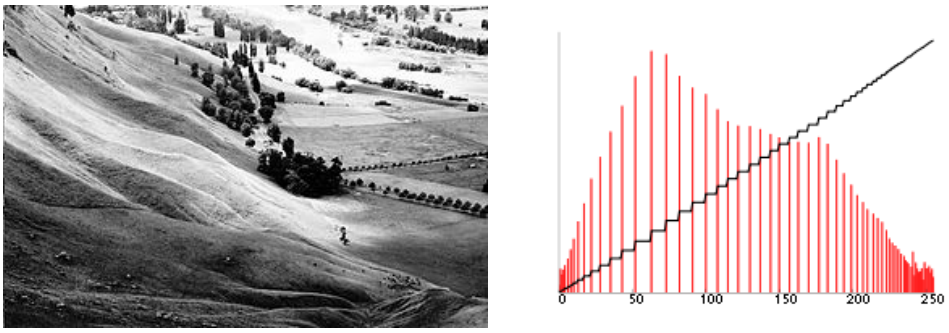
6.2 *Histogram Eşitleme*

Histogram eşitliği en basit ve en bilinen karşıtlık iyileştirme yöntemlerinden biridir[16]. Genellikle bir görüntü işleme algoritması uygulanacağı zaman ilk adımda görüntüyü iyileştir mek için kullanılır. Histogram eşitliği yapılabilmesi için önce uygulanmak istenen kanal için histogram ölçümü yapılmalıdır. Sonrasında ölçülen bu değer görüntü üzerine dağıtılır. Böylece görüntüde renk dağılımları eşitlenir. Eğer eşitlenen görüntüde parlak imgecik sayısı karanlık imgecik sayısından oldukça fazla ise aydınlatma işlevi, aksi durumda ise karartma

işlevi görür. RGB görüntülerde her bir kanal için ayrı ayrı histogram hesabı ve histogram eşitlemesi yapılmalıdır.



Şekil 7: Normal Görüntü ve Histogram Çizgesi[3]



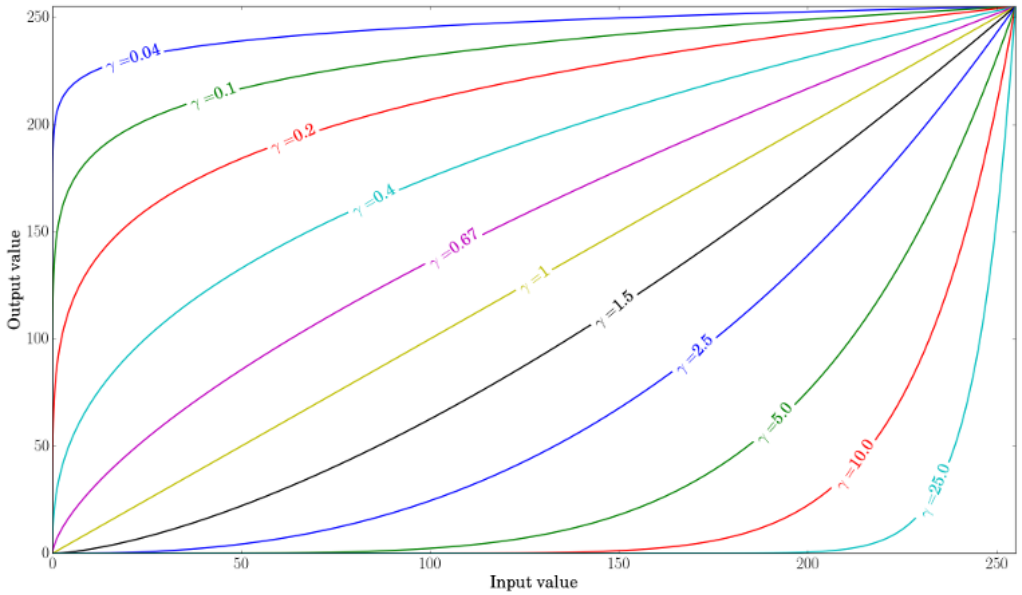
Şekil 8: Eşitlenmiş Görüntü ve Histogram Çizgesi[3]

6.3 *Gamma Düzeltmesi(Gamma Correction)*

Gamma düzeltmesinin histogram eşitliğinden farkı görüntü imgecikleri dışında bir paramet reye ihtiyaç duymasıdır. Bu parametre gamma katsayısıdır. Eşitlik 3'te gamma düzeltmesinin denklemi verilmiştir. Burada O değeri çıktı imgeciğine, I değeri uygulanacak imgeye, γ ise gamma katsayısına denk gelmektedir. Şekil 9'ta gamma katsayısının seçimine göre çıkış imgeciklerinin değerindeki değişim görülmektedir. Görüleceği üzere karanlık bir görüntüye

(imgecik değeri sıfıra yakın) küçük bir gamma katsayısı ile gamma düzeltmesi uygulandığında imgeciğin parlak bir değere yakınlaştığı görülmektedir.

$$O = \left(\frac{I}{255}\right)^{\frac{1}{\gamma}} * 255 \tag{3}$$

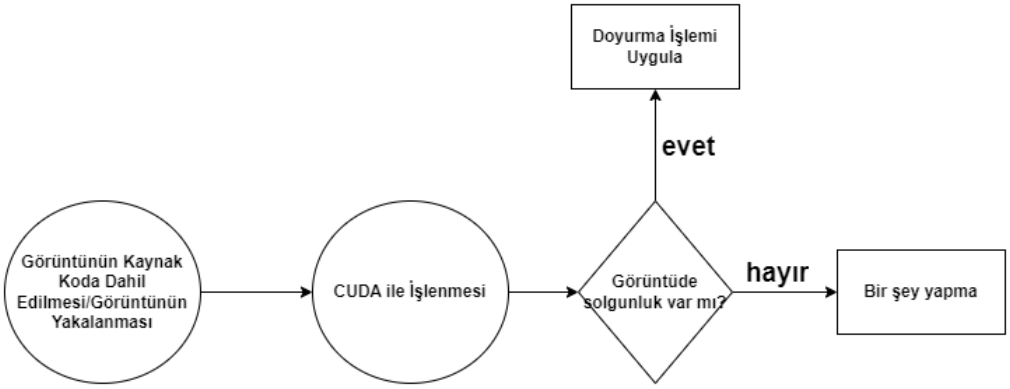


Şekil 9: Gamma Düzeltmesi Çizgesi[4]

Şekil 10: Gamma Düzeltmesi Uygulanmış Karanlık Görüntü[4]

6.4 *CUDA ile Histogram Eşitliği ve Gamma Düzeltmesi*

Taslak Program Akışı : Şekil 11'de görüldüğü üzere ilk adımda görüntü OpenCV işleviyle C++ kaynak koduna dahil edilmelidir. Okunan görüntü uygun oranlarda, uygun sayıda izlek tarafından işlenecek şekilde kuramsal olarak paylaştırılmalıdır. Burada kullanılacak GPU kartının da önemi vardır. Kartın hesaplama yeteneğine uygun şekilde bölüt ve izlek dağılımı yapılmalıdır. Görüntü ve bölüt-izlek tasarımı hazırlandıktan sonra CUDA işlevleri



Şekil 11: Uygulama Akış Çizeneği

ile görüntünün bulunduğu bellek alanı GPU belleğine gönderilmelidir. Sonrasında görüntü üzerinde çalışacak histogram eşitliği veya gamma düzeltme CUDA çekirdek işlevleri çalıştırıl malıdır. Çekirdek işlevler tamamlandıktan sonra görüntü CPU belleğine gönderilmelidir. Karşıtlık ayarı yapılan görüntüde solgun renkler varsa renk doyurma işlemi uygulanması tasarlanmaktadır.

Paralellik Yaklaşımı : Tasarlanan paralel programlama yaklaşımı verinin paralel işlenmesidir. Görüntü alt görüntü parçacıklarına bölünüp, bu alt parçacıklar üzerinde eş zamanlı olarak çekirdek işlevler çalıştırılacaktır.

Histogram Eşitliği : Algoritmanın ilk aşaması olan histogram hesaplama adımı çok maliyetli bir işlem olmadığı ve bu değerin GPU tarafından hesaplanması bir olasılık verim düşüklülüğüne

neden olabileceği için bu adımın CPU üzerinde gerçekleştirilmesi, histogram değerinin ise görüntü ile GPU belleğine gönderilmesi tasarlanmaktadır. Eşitlik uygulaması için ise her bir imgeciğe erişip imgeciğin kanal değerlerine eşitlik işlemi uygulanması tasarlanmaktadır.

Gamma Düzeltmesi : Görüntü CPU süzgecinden geçmeden doğrudan GPU'ya gönderilecek tir. Görüntünün her bir imgeciğinin kanal değerleri, γ katsayısının 1 olmadığı her durumda gamma düzeltmesi denkleminden geçirilecektir. Görüntü aydınlatılmaya çalışıldığı için katsayı 0.3 civarında seçilecektir.

CUDA Mimarisi : Görüntü CPU'dan GPU'ya gönderildikten sonra yürütülecek olan çekirdek işlevin bölüt ve izlek oranları oldukça önemlidir. Görüntü iki boyutlu bir veri olduğu için konuşlandırılacak bölütler iki boyuta yayılacaktır. Bölütler ise artık alt görüntü parçacıklarını temsil etmektedir. Bölütler de içerdikleri görüntü parçaları içerisinde iki boyutlu izleklerle tüm imgeciklere erişeceklerdir. Çalışmada Nvidia GTX 1660Ti kartı kullanılacaktır. Bu kart 24 SM içermekte ve her SM aynı anda 2 tane warp açabilmektedir. Yani her SM aynı anda 64 (2*32) izlek çalıştırabilmektedir. Bu, her SM sadece 64 tane izlek çalıştırabilir demek değildir. SM'den 64'den fazla izlek çalıştırması istenebilir. Sadece aynı anda yapamaz. Toplamda ise 1536 (24*64) izleğe aynı anda destek verebilmektedir. Buradaki en önemli nokta açılacak izlek sayısı ve izleklerin yüklenmesidir. Programcı en iyi verime ulaşmak için kartı %100 kullanmak zorunda değildir. Bu bir bakıma en verimli yük dengesini belirli oranlarda arayıştır. Bazı durumlarda SM'ler daha fazla yük ile daha iyi sonuç verebilmektedir. İlk adımda 1536 izlek üzerine yük eşit dağıtılarak; ikinci adımda ise izleklerin yarısı, iki kat yük ile kullanılacaktır. Aradaki verim farkları ölçülecektir.

BÖLÜM: IV

SONUÇ

7

SONUÇ

Görüntü işlemenin temel işlevleri olan birtakım algoritmaların paralel programlamasında, küçük boyutlu veriler işlenirken GPU-CPU arasındaki veri aktarımının algoritmayı işlemekten daha uzun sürdüğü görülmüştür. Buna karşın tasarlanması amaçlanan algoritmanın da küçük veriler üzerinde CPU ile neredeyse eşdeğer bir verim sağlayacağı düşünülmektedir. Veri boyutunun büyüdüğü her senaryoda daha yüksek oranda hızlanacağı düşünülmektedir. Fakat bu hızlanmanın, benzeri algoritmaların paralel işlenmesine bakılarak en iyi durumda 3-4 katı aşamayacağı tahmin edilmektedir.

K A Y N A K Ç A

[1] “A general-purpose parallel computing platform and programming model.” https://docs.nvidia.com/cuda/archive/11.8.0/cuda-c-programming guide/index.html. Accessed: 2023-01-08.

[2] “Image histogram bright image dark image low contrast image high contrast image.” https://www.gofastresearch.com/2020/05/Image-Histogram.html. Accessed: 2023-01-11.

[3] “Histogram equalization.” https://en.wikipedia.org/wiki/Histogram_equalization. Accessed: 2023-01-11.

[4] “Gamma corection in opencv.” https://docs.opencv.org/3.4/d3/dc1/tutorial_basic_linear_transform.html. Accessed: 2023-01-08.

[5] S. A. Manavski, “Cuda compatible gpu as an efficient hardware accelerator for aes cryptography,” in *2007 IEEE International Conference on Signal Processing and Communications*, 2007.

[6] Z. Yang, Y. Zhu, and Y. Pu, “Parallel image processing based on cuda,” in *2008 International Conference on Computer Science and Software Engineering*, vol. 3, 2008.

[7] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of gpu-based convolutional neural networks," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2010.

[8] I. K. Park, N. Singhal, M. H. Lee, S. Cho, and C. Kim, "Design and performance evaluation of image processing algorithms on gpus," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 1, 2011.

[9] U. Cekmez, M. Ozsiginan, and O. K. Sahingoz, "A uav path planning with parallel aco algorithm on cuda platform," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, 2014.

[10] M. M. Hussain, H. Hattori, and N. Fujimoto, "A cuda implementation of the standard particle swarm optimization," in *2016 18th international symposium on symbolic and numeric algorithms for scientific computing (SYNASC)*, IEEE, 2016.

[11] C. Wang, S. Chandrasekaran, and B. Chapman, "cusfft: A high-performance sparse fast fourier transform algorithm on gpus," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2016.

[12] S. A. Razian and H. MahvashMohammadi, "Optimizing raytracing algorithm using cuda," *Emerging Science Journal*, vol. 1, no. 3, 2017.

[13] A. C. Cinar and M. S. Kiran, "A parallel implementation of tree-seed algorithm on cuda-supported graphical processing unit," *Journal of the Faculty of Engineering and Architecture of Gazi University*, vol. 33, no. 4, 2018.

[14] A. Al Sideiri, N. Alzeidi, M. Al Hammoshi, M. S. Chauhan, and G. AlFarsi, "Cuda implementation of fractal image compression," Jul 2019.

[15] Z. Wang, L. Wang, and C. Huang, “A fast abnormal data cleaning algorithm for performance evaluation of wind turbine,” *IEEE Transactions on Instrumentation and Measurement*, vol. 70, 2021.

[16] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*. USA: Prentice-Hall, Inc., 2006.

[17] N. Zhang, Y. shan Chen, and J. li Wang, “Image parallel processing based on gpu,” in *2010 2nd International Conference on Advanced Computer Control*, vol. 3, 2010.

[18] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. O’Reilly Media, Inc., 2008.