GPGPU Yöntemi ile Görüntülerin Gerçek Zamanlı Ortorektifikasyonu
(Real Time Orthorectification Of Images By GPGPU Method)

Hakan ŞAHİN[1], Mehmet Sıtkı KÜLÜR[2]
[1] Harita Genel Komutanlığı, 06100 Dikimevi, Ankara,
[2] İTÜ, İnşaat Fakültesi Geomatik Mühendisliği Bölümü , 34469 Maslak, İstanbul
hakan.sahin@hgk.msb.gov.tr

ÖZET

Bilgisayarların grafik kartları üzerindeki grafik işlemci birimleri (Graphic Processing Units – GPU) on sene öncesine göre, özellikle performans ve yeteneklerinin artışı doğrultusunda büyük gelişme göstermiştir. Modern GPU'lar sadece çok güçlü grafik motoru olmaktan çıkarak, bilgisayar işlemcilerine (Central Processing Unit-CPU) göre aritmetik işlem yapabilme hızı ve hafıza band genişliği hızı çok daha yüksek olan ve üst seviyede paralel programlanabilir işlemciler halini almışlardır. GPU'ların programlanabilirliğindeki ve yeteneklerindeki hızlı gelişme, yüksek seviyede hesap yapma ihtiyacı olan karmaşık problemlerle uğraşan araştırmacıların ilgisini çekmiştir. Bu ilgi "grafik işlemci birimi üzerinde genel amaçlı hesaplama (General Purpose Computation on Graphic Processing Units - GPGPU)" ve "akış işleme (stream processing)" kavramlarını ortaya çıkarmıştır. Grafik işlemcilerin bilgisayar işlemcilerine bir alternatif olarak gündeme gelmesinin asıl nedeni ise; çok güçlü ve bunun yanında ucuz temin edilebilir donanım olmalarıdır. Bu grafik çipler, sabit uygulama donanımları olmaktan çıkarak günümüzde modern, güçlü ve programlanabilir genel ihtiyaçları karşılayabilecek işlemcilere dönüşmüşlerdir.

Yapılan çalışma içerisinde, çeşitli fotogrametrik uygulamalar ve özellikle ortorektifikasyon işlemi GPGPU yöntemi ile CUDA (Compute Unified Device Architecture-Birleşik Hesaplama Aygıt Mimarisi) programlama dili kullanılarak yeniden programlanmıştır. Böylelikle daha kısa sürede ve daha ucuz maliyetli donanımlarla ortorektifiye edilmiş görüntülerin nasıl elde edilebileceği ortaya konulmaya çalışılmıştır.

Bu amaca yönelik olarak yapılan uygulamalarda elde edilen sonuçlar değerlendirildiğinde, yöntemin fotogrametrinin görüntü işlemeyi gerektirdiği ve aynı işlem adımlarının her bir piksel için tekrarlandığı durumlarda ve ayrıca hesap yoğun işlem adımlarında çok etkili ve hızlı sonuçlar verdiği görülmüştür. Özellikle ortorektifikasyon amacıyla yapılan uygulamalarda aynı donanımla CPU'ya oranla 7 kat hız farklarına ulaşılmıştır.

Anahtar Kelimeler: *GPGPU, CUDA, akış işleme, programlama, ortorektifikasyon, doğrudan yöneltme, görüş analizi, görüntü filtreleme.*

ABSTRACT

The graphic processing units (GPU) on the graphic cards integral parts of computers are really developed today according to the last ten years. The development was the increase of the GPUs performance and capabilities. The modern GPUs are not only became powerful graphic engines and also they are high level parallel programmable processors with very fast computing capabilities and high memory bandwidth speed compared to central processing units (CPU). The rapid development of GPUs programmability and capabilities attracted the researchers dealing with complex problems which need highly level calculation. This interest has revealed the concepts of "General Purpose Computation on Graphics Processing Units (GPGPU)" and "stream processing". The graphic processors are powerful hardware which is really cheap and affordable. So the graphic processors became an alternative to computer processors. The graphic chips which were standard application hardware have been transformed into modern, powerful and programmable processors to meet the overall needs.

In this study, some photogrammetric applictions and especially orthorectification process were coded by using GPGPU method and CUDA (Compute Unified Device Architecture) programming language. So we can orthorectify images with cheaper hardware in a short time.

The results obtained are evaluated; the method is really suitable for image processing and photogrammetry especially if we do the same calculations to per image pixels. Also it is suitable for intensive calculation procedures. Especially with orthorectification procedure with GPU is 7 times faster than CPU implementation and speedup is 7 times.

Keywords: *GPGPU, CUDA, stream processing, programming, orthorectification, direct georeferencing, viewshed analysis, image filtering,*

1. GİRİŞ

Grafik işlemci birimleri (Graphical Processing Unit-GPU) olarak bilinen ve bilgisayarlar ile ekran arayüzüne sahip olan birçok elektronik donanımın ayrılmaz bir parçası olarak kullanılmaktadır. Bilgisayar ve elektronik dünyasında son yıllarda yaşanan hızlı gelişmelerden grafik kartları da son derece etkilenmişlerdir. Bu gelişme GPU'ların hızlarının, performanslarının ve yeteneklerinin artışı doğrultusunda olumlu yönde olmuştur. Bilgisayarlarda kullanılan modern GPU'lar sadece çok güçlü grafik motoru olmaktan çıkarak bilgisayar işlemcilerine (Central Processing Unit-CPU) göre aritmetik işlem yapabilme hızı ve hafıza band genişliği çok daha yüksek olan ve üst

seviyede paralel programlanabilir işlemciler halini almışlardır. GPU'ların paralel programlanabilir hale gelmesi, üzerlerinde çalışabilecek uygulamaların geliştirilmesi ve yeteneklerindeki hızlı gelişme, yüksek seviyede hesap yapma ihtiyacı olan karmaşık problemlerle uğraşan araştırmacıların yoğun ilgisini çekmiştir. Bu ilgi "grafik işlemci birimi üzerinde genel amaçlı hesaplama (General Purpose Computation on Graphic Processing Units - GPGPU)" ve "akış işleme (stream processing)" kavramlarını ortaya çıkarmıştır. Grafik işlemcilerin bilgisayar işlemcilerine bir alternatif olarak gündeme gelmesinin asıl nedeni ise; çok güçlü ve bunun yanında ucuz temin edilebilir donanım olmalarıdır. Bu grafik çipler, sabit uygulama donanımları olmaktan çıkarak günümüzde modern, güçlü ve programlanabilir genel ihtiyaçları karşılayabilecek işlemcilere dönüşmüşlerdir.

Çeşitli platformlardan elde edilen görüntüler harita üretiminde, özellikle görüntülerin değerlendirme sonuçlarının hızla elde edilmesi ve karar vericilere aktarılmasının gerektiği felaketlerde, orman yangınları ve depremler gibi doğal afetlerde yaşanan kriz durumlarında, askeri açıdan hedef istihbaratı ve hedef konumunun hızlıca tespitinde sıklıkla kullanılmaktadırlar. Fakat bu görüntülerin anlamlı hale gelmesi ve belirtilen uygulamalarda kullanılabilmesi için öncelikli olarak yapılması gereken işlem adımı; bu görüntülerin ortorektifikasyonudur. Özellikle son zamanlarda yaygınlaşarak çeşitli kurum ve kuruluşlar tarafından birçok uygulamada kullanılmaya başlanan insansız hava araçları (İHA) yardımıyla elde edilen görüntüler üzerinden çok hızlı karar vermek, çıkarımlar yapmak ve çeşitli hedef tespiti yaparak bu hedefleri de doğru koordinatlarla tarifleyebilmek için elde edilen görüntülerin yöneltilmesi yani ortorektifiye edilmesi ihtiyacı bulunmaktadır.

Sayısal hava kameraları ve insansız hava araçları gibi çeşitli sensör ve platformlardan elde edilen görüntülerin dosya boyutları oldukça yüksektir. Haliyle bu görüntülerin ortorektifikasyonu için çok güçlü bilgisayarlara ihtiyaç vardır. Thomas ve diğ. (2008) tarafından yapılan bir çalışma incelendiğinde görüntülerin anlık ortorektifikasyonu için toplam beş adet çok güçlü donanıma sahip bilgisayarın kullanıldığı görülmektedir. Böyle bir sistemin normal uçak platformları için oluşturulabileceği, maliyet göz önüne alınmazsa mümkün olabilir. Fakat insansız hava araçları gibi yapı itibariyle normal uçaklardan çok daha küçük olan ve görüntü alımı gerçekleştirerek çok hızlı ve doğru sonuç elde edilmesi gereken platformlarda ise böyle çoklu bilgisayar sistemlerinin kurulabilmesi pek olanaklı değildir. Aynı zamanda bu sistemlerde platform üzerindeki kamera, navigasyon sistemi gibi faydalı yük olarak ifade edilen donanımın hafif olması da son derece önemli bir faktördür. İnsansız hava araçları üzerine eklenen her bir parça sisteme ağırlık olarak etki etmekte ve bu da sistemi olumsuz yönde etkilemektedir.

İlk deneysel çalışmalara Ohio State Üniversitesi'nde 1980'li yılların sonunda başlanarak, yerde hareket halindeki araç üzerinde bulunan algılama sistemi yardımıyla doğrudan görüntü elde edilmesi üzerinde çalışmalar yapılmıştır. Ardından 1990'lı yılların başında Calgary Üniversitesi Coğrafi Bilgi Sistemi uygulamaları için karada hareketli ölçme sistemi VISAT (Video cameras, an Inertial System, and SATellite GPS receivers) 'ı geliştirmiştir. Bu deneysel çalışmalarla birlikte doğrudan yöneltme yaklaşımı, hareket halindeki algılayıcının konum ve dönüklük verilerinde oluşan dış yöneltme elemanlarının belirlenmesi üzerine yoğunlaşmıştır (Schwarz ve diğ, 1993; Schwarz 1993). Günümüze kadar yapılan deneysel çalışmalar ve test projeleri, GPS/IMU verilerinin sinyal işleme teknikleri kullanılarak iyileştirilmesi ve bu verilerin Kalman Filtreleme yöntemi ile birleştirilmesi, klasik fotogrametrik kameralar kullanılarak doğrudan yöneltme yaklaşımı, sayısal kameralar kullanılarak doğrudan yöneltme yaklaşımı ve uçakta kullanılan tarayıcı sistemler ile doğrudan yöneltme yaklaşımı konularında yapılmıştır (Yastıklı, 2003).

Amerika Birleşik Devletlerinde 11 Eylül 2001 yılında meydana gelen terörist saldırı sonrası, arama-kurtarma çalışmalarını yönlendirmek amacıyla hızlıca bölgenin haritalarının üretimi için bu yöntemden faydalanılmıştır (Yastıklı, 2003). Yapılan bu çalışma; özellikle hızlı ve acil müdahalelerin gerektiği afet durumunda, afet bölgesi ile ilgili hızlıca yorum yapabilmek ve karar verme sürecini hızlandırmak amacıyla, farklı sensör ve platformlardan elde edilen veri ve görüntülerin hiç yer kontrol noktası olmadan GPS/IMU verileriyle doğrudan yöneltilerek kullanılması ve böylelikle yöntemin geçerliliği açısından çok iyi bir örnek teşkil etmektedir.

GPU'lar aynı hesaplama işlem adımının birçok veri elemanına, özellikle yüksek aritmetik doğrulukla uygulanmasının gerektiği durumlarda çok etkili ve hızlı sonuçlar ortaya koymaktadırlar. Böylelikle yapılan hesaplama işleminin daha hızlı ve doğru olması sağlanmaktadır. Bilgisayar CPU'ları bir akış kontrolü içerisinde ve belli bir sıra ile her seferinde sadece tek bir hesaplama

yaptıkları için GPU'lar ile kıyaslandığında daha yavaş işlem yapmaktadırlar. Bu yapı bilgisayar teknolojisinin kullanıldığı çok çeşitli uygulamalar için değerlendirilebilmektedir. Görüntü işleme, matris hesaplamaları gibi GPU tabanlı grafik olmayan hesaplamalar paralel veri işlemeye çok uygun yapıdadırlar.

Bu çalışma kapsamında; GPU'ların genel amaçlı paralel programlama ve hesaplama gücünü kullanarak GPGPU yöntemiyle sayısal hava kameraları, insansız hava araçları gibi çok çeşitli platformlardan elde edilebilecek görüntülerin hızlı bir şekilde gerçek zamanlı ortorektifikasyonunun yapılabilmesi için yöntem detaylı bir şekilde incelenerek, bir program algoritması ortaya çıkarılmış ve uygulanabilirliği değerlendirilmiştir.

2. GPGPU VE AKIŞ İŞLEME

GPU'lar basit OpenGL ve DirectX desteği olan grafik donanımları olmalarından çok programlanabilir olmaları açısından birçok araştırmacının ilgisini çekmiştir. GPU hesaplamanın ilk zamanlarındaki genel yaklaşım olağanüstü karmaşıktır. Çünkü OpenGL ve DirectX gibi API'ler (uygulama programlama arayüzü - application programming interface) halen GPU üzerinde hesaplama yapabilmek için kullanılabilecek, çeşitli kısıtlamaları olan ve aslında grafik programlama için tasarlanan araçlardır. Bu nedenle de araştırmacılar problemlerini çözmek için genel amaçlı hesaplamaları grafik API'nin anlayacağı şekilde geleneksel grafik işleme problemi olan renderlama işlemine benzetmenin yöntemlerini aramışlardır.

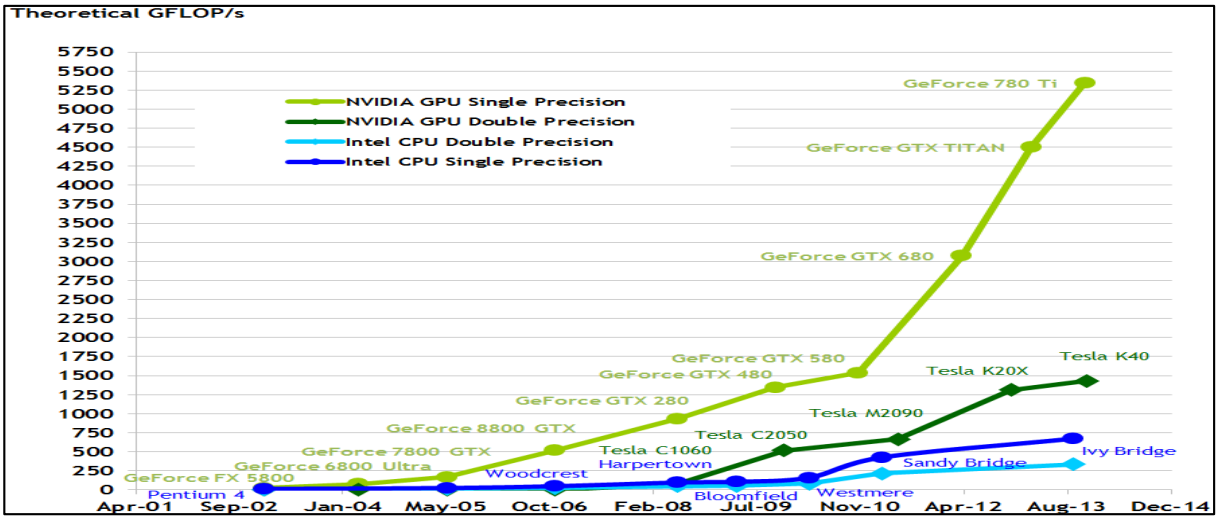
Aslında, 2000'lerin başındaki GPU'lar, piksel gölgelendirici (piksel shader) olarak bilinen programlanabilir aritmetik birimleri kullanarak ekran üzerindeki her bir pikselin rengini üretmek için tasarlanmışlardır. Genel olarak bir piksel gölgelendirici pikselin ekran üzerindeki (x, y) konumunu ve ek bazı bilgileri girdi olarak birleştirip hesap yaparak pikselin sonuç rengini üretmektedir. Bahsedilen ek bilgiler; girdi renkleri, doku (texture) koordinatları veya çalıştırıldığında gölgelendirmeye etki edecek diğer özellikler olabilmektedir. Fakat girdi renkleri ve dokularda uygulanan aritmetik tümüyle programcı tarafından kontrol edilebilmektedir. Bu noktada araştırmacılar "girdi renklerinin" aslında herhangi bir "veri" olabileceğini keşfetmişlerdir. Gelişme bu şekilde başlayarak devam etmiştir.

Böylelikle girdiler renkten farklı olarak sayısal bir veri olarak nitelendirildiğinden, programcılar piksel gölgelendiricileri bu veri üzerinde istenilen hesaplamaları yapmayı programlayabilmişlerdir. Elde edilen sonuçlar aynı sonuç piksel renkleri gibi GPU'ya geri dönmüş ve böylelikle bu renkler basitçe programcının girdilerini vererek GPU'ya yaptırdığı hesaplama işleminin sonuçları olmuşlardır. Bu veri araştırmacılar tarafından geri okunabilmiştir. Özünde GPU, bilgisayarda çizilen ham modelin yazılım yardımıyla resime dönüştürülmesi işlemi olarak açıklanan renderlama işlemi dışındaki hesaplama işlemlerini de sanki renderlama işlemiymiş gibi değerlendirerek, bu işlemi yapıyormuş gibi yapmıştır.

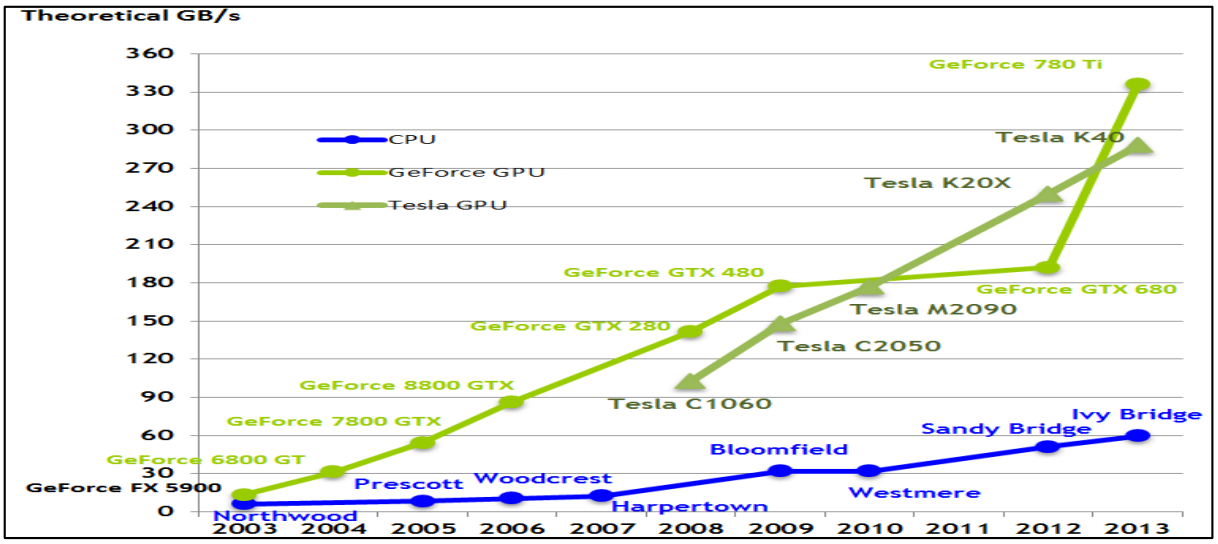
Buradaki en büyük problem, grafik işlemci birimlerinin mevcut programlama yöntemlerinden farklı bir programlama modelini kullanıyor olmasıdır. Bu nedenle etkili bir GPU programlama, mevcut program algoritmasının, donanımın yapısını ve sınırlamalarını da dikkate alan grafik terimlerini kullanarak tekrardan yazılmasını gerektirmektedir. Mevcut çift çekirdekli işlemcilerin programlanabilmesi, geleneksel programlama yöntemleriyle gerçekleştirilememekte ve tipik olay yordamlı programlama yönteminin birden fazla çekirdekli işlemcilerin programlanması için kullanılması mümkün olamamaktadır.

GPGPU için uygulanacak programlama modeli akış hesaplama (stream computing-processing) terimiyle değişmiştir. Bu model içerisinde, akış içerisindeki her bir elemana uygulanan yoğun hesaplama işlemlerini (kernel functions-çekirdek fonksiyonları) tanımlamak için girdi verileri ve çıktı verileri birer akış olarak nitelendirilmektedir. Grafik kartları üzerinde ise bu akışları hesaplayan ve işleyen çok sayıda işlemci bulunmaktadır. Örneğin günümüzde kullanılan grafik kartlarından birisi olan Nvidia GTX780 serisi grafik kartı üzerinde 2304 adet akış işlemcisi bulunmaktadır. Bu da yan yana sıralanmış, birlikte işlem yapabilme kapasitesi olan 2304 adet tek çekirdekli işlemcisi olan bilgisayarlar kümesi gibi düşünülebilir. Bu akış işlemcileri sayesinde grafik kartları aynı anda birden fazla yoğun işlemi yapabilmektedirler.

GPU'lar CPU'lara göre çok daha fazla paralel hesap yapabilmektedirler. Bu durum Şekil 1'de bir grafikle gösterilmektedir. Burada işlemci hızını "flop" kavramı belirlemektedir. Flop; "saniye başına kayan nokta işlemi" anlamına gelmektedir. Grafik incelendiğinde Nvidia firmasının grafik işlemcilerinin, Intel firmasının bilgisayar işlemcilerine oranla 2014 yılındaki değerlere göre ortalama on kat daha hızlı olduğu görülmektedir.



Şekil 1. CPU ve GPU'nun saniyede kayan nokta işlem miktarı yıllara göre gelişimi (Nvidia, 2015).



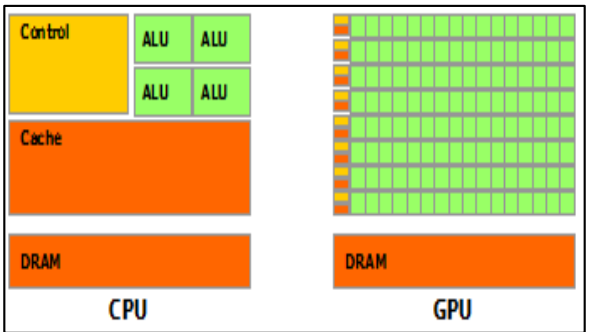
Şekil 2. GPU ve CPU'ların band genişlikleri (Nvidia, 2015).

Hafıza band genişliği kavramı, ekran kartının işlemcisi ile hafızası arasında saniyede aktarılabilen toplam veri miktarı boyutu anlamına gelmektedir. Hafıza veriyolu genişliğinin byte cinsinden değeri ile efektif frekansın çarpılmış hali olarak ifade edilir. Bellekle grafik işlemcisinin haberleşmesinin hızlı olması da grafik kartının performansını artıran bir etkendir. Saniye başına kayan nokta işlem kapasitesinin yıllar dikkate alındığında artış göstermesi paralelinde, hafıza band genişliğinde de bir gelişme olmuştur. Şekil 2 incelendiğinde, GPU'ların hafıza band genişliklerinin, CPU'lara oranla 4 kat daha fazla bir orana ulaştıkları görülmektedir.

CPU ve GPU'ların kayan nokta işlem kapasiteleri arasındaki farklılığın arkasında yatan asıl neden, GPU'ların yüksek kapasitede hassas paralel hesap yapma ihtiyacı olan grafik renderlama (doku kaplama) işi için özel olarak tasarlanmış olmalarıdır.

Tasarımdaki bu fark Şekil 3'de gösterilmeye çalışılmıştır. Bu şekil incelendiğinde, GPU'ların CPU'lardan farklı olarak; veriyi önbellekte tutmak ve bir akış kontrolü yapmaktan çok sadece veriyi işlemek için tasarlanmış daha fazla sayıda transistöre sahip olduğu görülmektedir.

Böylelikle GPU'ların CPU'lardan farklı olarak birçok aritmetik işlemlerle dolu paralel hesaplamaları yapmak için tasarlandığı anlaşılmaktadır. GPU'lar veri dizinlerini, akış kontrolü yerine, çeşitli sıralı hesaplama iş parçaları şeklinde işlerler.



Şekil 3. CPU ve GPU'ların genel yapısı ve transistör sayısındaki farklılık (Nvidia, 2015).

3. CUDA TEKNOLOJİSİ

Yapılacak bir hesaplamada, tekrarlanan hesaplama işlem adımının birçok veri elemanına, yüksek aritmetik doğrulukla uygulanmasının gerektiği hallerde GPU'lar çok etkili sonuçlar vermektedir. Bunun en önemli nedeni aynı programın her bir veri elemanı için çalıştırılması ve akış kontrolüne çok az ihtiyaç duyulmasıdır.

Çok sayıdaki işlemci, yani transistörlerin yardımıyla küçük veri kümelerinin her biri için program çalıştırılarak hesap yapılmakta, sonuçta da daha yüksek aritmetik doğruluk elde edilerek çok büyük verinin önbellekte tutulmasının önüne geçilip, hafızaya ulaşmadaki gecikmeler de önlenerek işlem süresinden kazanılmaktadır. Verinin paralel işlenmesinde, veri elemanları paralel iş parçalarına ayrılır. Birçok uygulama için bu yöntem işlem ve hesaplama hızını artırmak için kullanılabilir. Örneğin üç boyutlu (3B) renderlama işleminde, çok büyük boyutlardaki pikseller ve verteksler birer paralel iş parçası haline getirilir ve bu şekilde GPU'lar tarafından hesaplanır ve işlenirler (Yılmaz, 2010). Benzer şekilde görüntü ölçekleme, stereo görüş ve görüntüden şekil tanıma gibi renderlanmış görüntülerin ve videoların işlenmesinde görüntü bloklar ve pikseller şeklinde iş parçaları haline getirilebilirler. Aslında görüntü işleme dışında sinyal işlemeden fiziksel simülasyona, finansal hesaplamadan biyolojik hesaplamalara kadar birçok algoritma paralel programlama ve paralel işlemeyi kullanarak hızlanmışlardır.

Uzun zaman önce geliştiriciler paralel hesaplama işleri için GPU'ları kullanmayı denemişlerdir. İlk başlardaki bu kullanım girişimleri (rasterizing ve Z-buffering gibi) çok ilkeldir ve donanım fonksiyonlarını tam anlamıyla kullanabilmek için sınırlı kalmıştır. Fakat gölgelendirme işlemleri matris hesaplamalarını hızlandırmıştır.

2003 yılında yapılan SIGGRAPH konferansında hemen hemen hiç katılım olmadan geçen ve "GPU computing" için ayrılmış "GPGPU" adında bir oturum olmuştur. Burada en iyi bilinen başlık "BrookGPU" adıyla anılan akış programlama dili olmuştur. Bu programlama dilinin yayınlanmasından önce popüler olan iki yazılım geliştirme uygulaması vardır; Direct3D ve OpenGL. Fakat bunlarla sınırlı sayıda GPU uygulamaları geliştirilebilmektedir. Sonrasında Brook projesi ile GPU'ların paralel işleyiciler olarak kullanılabilmelerini ve C diliyle programlanabilmelerini olanaklı kılmıştır. Stanford Üniversitesi tarafından geliştirilen bu proje, iki farklı grafik kartı tasarımcısı ve üreticisi olan NVIDIA ve ATI firmalarının dikkatlerini çekmiştir. Sonrasında ise Brook'u geliştiren bazı insanlar NVIDIA firmasına katılmışlar ve paralel hesaplama stratejisini yeni bir pazarlama birimi olarak sunmaya başlamışlardır. Böylelikle grafik donanımı doğrudan kullanabilen bir yapı ortaya çıkmıştır ve adına NVIDIA CUDA (Birleşik Hesaplama Aygıt Mimarisi – Compute Unified Device Architecture) denilmiştir.

2006 yılının Kasım ayında NVIDIA, endüstrideki ilk DirectX10 destekli GPU'su GeForce8800 GTX'i duyurdu. Bu ekran kartı aynı zamanda NVIDIA CUDA teknolojisine sahip ilk ekran kartı idi. Fakat duyuruları daha önceden yapılmış olmasına rağmen NVIDIA, yeni teknolojisi CUDA'yı ancak 2007 yılının Şubat ayında kamuoyuna açıklayarak tanıtmıştır. Bu teknoloji geniş bir kullanıcı kitlesinin gereksinimlerini karşılamak için geliştirilmiştir. En önemli ihtiyaç GPU'ların kolayca programlanabilirliğinin sağlanmasıdır. Basitlik ve sadelik, GPU ile paralel programlamanın kolaylığını ve daha fazla disiplin içerisinde kullanılabilirliğini sağlamak için gerekli görülmüştür. CUDA'dan önce, GPU paralel programlama grafik API(Application Programming Interface-Uygulama Programlama Arayüzü)'lerinin gölgelendirme modelleri ile sınırlıydı. Böylelikle sadece vertex ve parça gölgelendirmesinin doğasına uygun olan problemler GPU paralel programlama yöntemi kullanılarak hesaplanabilmiştir. Ayrıca, dokulandırma ve GPU gereken 3B işlemler için sadece kayan noktalı sayıların kullanılması GPU hesaplama konusunun popülaritesini sınırlandırmıştır. NVIDIA firması GPU paralel programlamayı kolay ve pratik hale getirebilmek için "C" programlama dilinin minimum eklentileri ile kullanımını önermiştir.

Diğer önemli bir konu ise CPU ve GPU kaynaklarının birarada kullanılabilirliğini sağlayan heterojen bir hesaplama modelinin ortaya konulması olmuştur. CUDA, programcıların kod ve veriyi alt parçalara bölmesine olanak tanıyarak, CPU/GPU mimarisi ve ilgili programlama tekniklerine uygun hale getirebilmesini sağlamıştır. Böyle bir ayrım aslında GPU ve CPU'nun her ikisinin de kendi hafızasına sahip olmasından kaynaklanmaktadır. Bu anlamda aynı zamanda mevcut tanımlamaların CPU'dan GPU'ya aktarımı mümkün olabilmektedir (Yılmaz, 2010). Kısaca CUDA teknolojisi C programlama diline dayanan, paralel hesap yapabilmek için GPU komutlarını ve video hafızayı kontrol edebilen, NVIDIA tarafından geliştirilen yazılım-donanım hesaplama mimarisidir. CUDA ile GPU programlama, daha önceki GPGPU çözümlerine göre oldukça esnek ve kolaydır.

CUDA mimarisi, NVIDIA GeForce 8x serisi ve GeForce, Quadro, Tesla gibi daha yeni model tüm grafik kartlarında kullanılabilmektedir. Veri-paralel ve iş parçası-paralel mimari ölçeklenebilir bir yapıya sahiptir. Yeniden fazladan bir çaba sarfetmeden mevcut çözümü GPU üzerinde daha fazla iş parçacığı işleme kapasitesiyle uygulamak ve çalıştırmak mümkündür. Başka bir ifadeyle NVIDIA 8 serisi için yazılan bir kod, başka bir kodlamaya gerek duymadan NVIDIA Quadro için de çalışabilecek ve olduğu gibi kullanılarak daha hızlı sonuç alınabilecektir.

İyi veri paralelleştirmesi ve iş parçacığı paralelleştirmesi için NVIDIA tarafından üç özet çıkarım yapılmıştır. CUDA programlama dilini kullanarak kod yazan programcıların hayatını kolaylaştıran bu çıkarımların listesi aşağıdaki gibi verilebilir:

- İş Parçası Grup Hiyerarşisi (Thread Group Hierarchy): İş parçaları bloklar halinde paketlenir ve bu paketlenen bloklar da tek bir grid olarak paketlenir.

- Paylaşılan Hafızalar (Shared Memories): CUDA iş parçalarının çok farklı ihtiyaçları görmek için tasarlanmış altı farklı hafızayı kullanmasına olanak tanır.

- Bariyer Senkronizasyonu (Barrier Synchronization): İş parçalarının küçük bir blok halinde eşitlenmesini ve ilgili hesaplamanın diğer parçalarının daha ileriye gitmeden bitmesini bekleyen bir iş parçacığı olmasını sağlamaktadır.

CUDA için C programlama dili, GPU üzerinde çalışabilen fonksiyonların C dilini kullanarak yazılabilmesini sağlamaktadır. Bu fonksiyonlar "çekirdekler-kernels" olarak ifade edilmektedir ve bu kerneller her bir iş parçacığı için paralel olarak yeniden üretilerek işleme alınmaktadırlar. Bu yönüyle bir defa çalışan geleneksel seri programlama fonksiyonlarından farklıdırlar. Bunu açıklayan bir örnek Şekil 4'de NxN boyutunda A ve B matrisinin çarpımını yapıp sonuçları C matrisine yazan bir kod parçası ile gösterilmektedir. Şekil 4'de de görüldüğü üzere iş parçacıkları için bloklar oluşturulmuştur. Bu bloklar bir, iki ya da üç boyutlu tanımlanabilmektedir.

CUDA mimarisi aşağıdaki gibi baştan sona bir iş parçacığı hiyerarşisi içerisinde çalışmaktadır:

- Grid: Bir grid bir veya iki boyutlu blokları içermektedir.

- Bloklar: Bir blok bir, iki veya üç boyutlu iş parçalarını içermektedir. Mevcut GPU'lar bir blokta 1024 iş parçasını içerebilmektedirler. Bloklar bağımsız yürütülmekte ve ölçeklenebilir bir şekilde uygun olan işlemcilere yönlendirilebilmektedir.

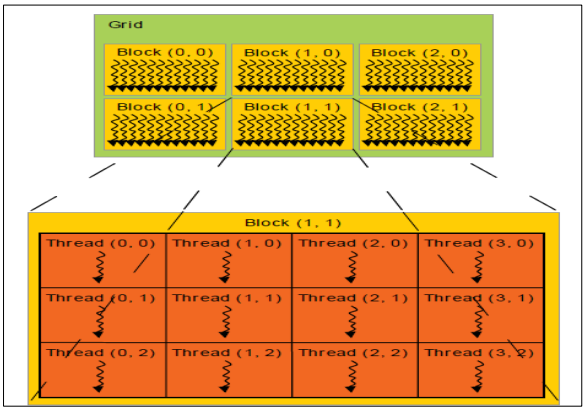
- İş Parçası (Thread): Bir iş parçası temel uygulama elemanıdır.

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N],
                       float C[N][N])
{
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    ...
    // Kernel invocation with one block of N * N * 1 threads
    int numBlocks = 1;
    dim3 threadsPerBlock(N, N);
    MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
    ...
}
```

Şekil 4. Kernel iş parçacığının tanımlanması.

Yukarıda bahsedilen bu hiyerarşi ve yapısı Şekil 5'de gösterilmiştir. Örneğin 1048576 pikselden oluşan bir görüntü, birbirinden bağımsız paralel olarak 512 blok boyutunda, 2048 grid ile işlenebilmektedir. Her bir bloktaki iş parçacığı sayısı ve her bir griddeki blok sayısı <<<…>>> yazım şekli ve int ya da dim3 tipinde tanımlanabilmektedir. İki boyutlu bloklar veya gridlere örnek Şekil 4'de yer alan matris toplamı örneğinde olduğu gibi verilebilir. Gridde yer alan her bir blok, bir, iki ya da üç boyutlu olacak şekilde tanımlanabilir ve blockIdx değişkeni ile kernelden bu indeks değerine erişilebilir. İş parçacığı bloğunun boyutuna da blockDim değişkeni ile kernelden ulaşılabilir.



Şekil 5. CUDA iş parçacığı hiyerarşisi (Nvidia, 2015).

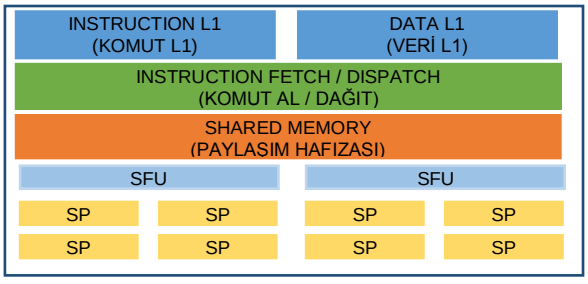
Bir önceki matris toplamı işlemini çoklu bloklara ayırdığımızda kodlama ifadesi Şekil 6'da gösterildiği gibi olacaktır.

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N],
float C[N][N])
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N)
        C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    ...
    // Kernel invocation
    dim3 threadsPerBlock(16, 16);
    dim3 numBlocks(N / threadsPerBlock.x, N / threadsPerBlock.y);
    MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
    ...
}
```

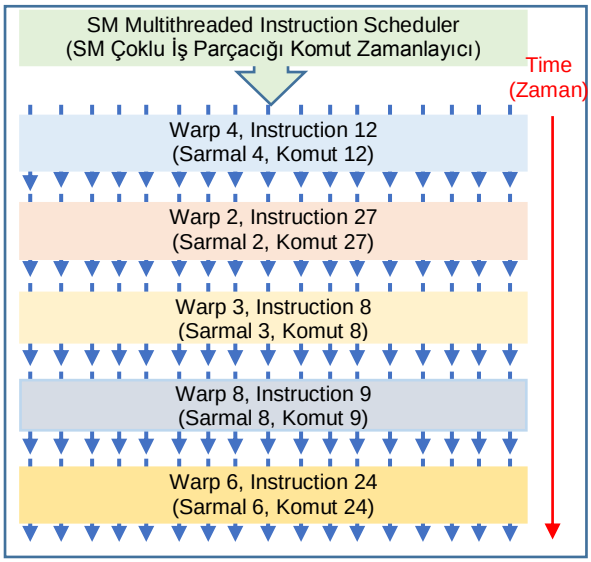
Şekil 6. Bloklara ayrılan kernel tanımlaması.

CUDA işlemcisine sahip GPU'lar, birçok çoklu iş parçacığı akış çoklu işlemcisine (streaming multiprocessors-SM) sahiptir. Bir SM'nin yapısı Şekil 7'de verilmektedir. Örneğin GTX295 GPU'su üzerinde 60 adet SM barındırmaktadır. Her bir SM de sekiz skaler işlemciye (Scalar Processors-SP) sahiptir. Şekil 7 incelenecek olursa SM'in aynı zamanda iki özel fonksiyon birimine sahip olduğu görülür: bir çoklu işlem tanımlama birimi ve çip üzerinde paylaşım hafızası.



Şekil 7. Akış çoklu işlemcisi (Streaming Multiprocessor).

Her bir SP, 32 adet iş parçacığını içeren tekil sarmal paketi eş zamanlı çalıştırabilir. Burada yer alan SFU (Special Functions Unit-Özel Fonksiyonlar Birimi) içerisinde sin, cosine, karekök ve ters alma gibi fonksiyonları içermekte ve bu fonksiyonları yerine getirmektedir.



Şekil 8. Tek komutlu çoklu iş parçacığı mimarisi.

SIMT (Single Instruction Multiple Thread -Tek Komutlu Çoklu İş Parçacığı) mimarisi eş zamanlı çok sayıda iş parçacığını yönetebilmektedir. Her bir tekil sarmal içindeki her türlü konuyu işleyen SIMT birimi, bütün SM'lerin herbiri için ayrı ayrı mevcuttur. SIMT'nin açıklamalı bir gösterimi Şekil 8'de verilmiştir. Her seferinde genel komut her bir adımda aktif iş parçacığına uygulanmaktadır. Böylelikle, her iş parçacığı aynı uygulama yolunu takip ederek performansın artmasını sağlamaktadır. Fakat dallanma nedeniyle sarmal içerisindeki iş parçacıkları genel komut gelene kadar beklemektedir, bu da bir miktar gecikmeye sebep olmaktadır. Bunun önüne geçmek ve en iyi performansı yakalayabilmek için, benzer paralel iş parçacıkları sıralı olarak düzenlenmelidir. Bu da ancak algoritmanın ve programın tasarım aşamasında yapılmalıdır.

Bu aşamadan sonra karşımıza diğer bir etken olarak GPU hafızası çıkmaktadır. GPU paralel programlamanın ne kadar etkin olduğu, GPU hafızasının ne kadar iyi kullanıldığına bağlıdır.

Yukarıdaki anlatımlar ışığında genel olarak CUDA işlem akışı dört adımdan oluşmaktadır ve şu şekilde özetlenebilir:

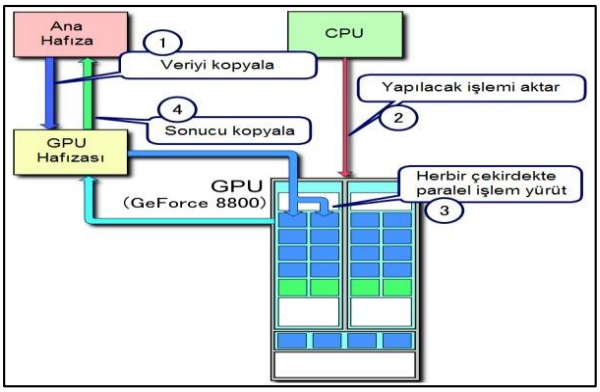
• Veri ana hafızadan GPU hafızasına kopyalanır.

• CPU yapılacak işlemi GPU ya aktarır.

• GPU her bir çekirdek için paralel işlemi yürütür.

• Sonuçlar GPU hafızasından ana hafızaya kopyalanır.

Bu adımlar bütün uygulamalar için geçerlidir. Bu adımların nasıl işlediği Şekil 9'da gösterilmeye çalışılmıştır.



Şekil 9. CUDA işlem akışı.

4. CUDA İLE PROJEKTİF REKTİFİKASYON

Ortorektifikasyon amacıyla kullanılabilecek rektifikasyon yöntemlerinden biri olan projektif dönüşüm yöntemi, çalışma içerisinde örnek bir uygulama için kullanılmıştır. Projektif dönüşümün uygulanabilmesi için resim düzlemi ile izdüşüm düzlemi arasındaki geometrik dönüşüme ihtiyaç vardır. Projektif dönüşüm denklemindeki sekiz bilinmeyenin çözülebilmesi için obje düzleminde koordinatları bilinen en az dört kontrol noktası gereklidir. Projektif dönüşüm daha çok, özellikle arazi yüzeyinin düz olduğu alanlar, bina cepheleri gibi detayların yer aldığı hava resimlerinin yöneltilmesi için uygulanabilirdir ve rölyef etkisini ortadan kaldırmamaktadır (Novak, 1992).

Yapılan bu örnek uygulamada bir düzlem alanda çekilen 4096x4096 piksellik bir örnek görüntü değerlendirilmiştir. Bu uygulama içerisinde kodlanacak program için işlem adımları aşağıdaki gibi özetlenebilir:

• Görüntü Koordinatları ve Obje Koordinatları girdi olarak kullanılır.

• Matris yapıda denklem sistemi yazılır ve bu denklem sistemi çözülerek katsayılar hesaplanır.

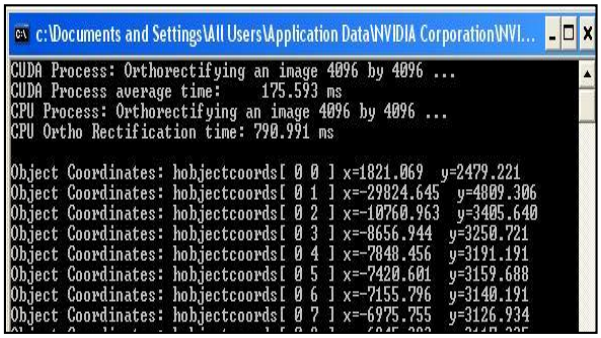
• Katsayıları hesaplanan denklemler ile bilinmeyen obje koordinatları, görüntü üzerindeki her bir pikselin görüntü koordinatlarından hesaplanır.

• Hesaplanan obje koordinatları bir dosyaya yazdırılır.

• Üçüncü ve dördüncü sıradaki işlem adımları her bir piksel için tekrarlanır.

Yukarıda verilen işlem adımları da dikkate alınarak örnek görüntü üzerinde obje ve resim koordinatları bilinen dört adet nokta seçilmiştir. Bu noktalar yardımı ile denklem sistemi çözülmüş ve denklem katsayıları hesaplanmıştır. Bu aşamanın ardından katsayıları belirlenen denklem yardımı ile görüntü üzerindeki her bir pikselin obje uzayındaki koordinatları hesaplanmıştır. Bu yapılan işlemler için geliştirilen algoritma CUDA programlama dili kullanılarak kodlanmış ve programın hem CPU, hem de GPU'yu kullanarak aynı işlemleri yapması sağlanarak, işlem süreleri arasındaki farklar karşılaştırılmıştır. Kodlanan program çalıştırıldığında elde edilen ekran görüntüsü Şekil 10'daki gibidir.

Yapılan çalışmada uygulanan projektif dönüşüm ile ortorektifikasyon yöntemi, görüntüyü oluşturan her bir piksel için tekrarlanmaktadır. Böyle bir durumda CUDA programlama dili kullanılarak problem iş parçalarına bölünüp dağıtık bir şekilde, birçok grafik işlemciye aynı anda verilip, hesaplama sonuçları elde edilmektedir.



Şekil 10. Projektif rektifikasyon programı.

Sonuçta program hem CUDA mimarisini kullanarak grafik kartı üzerindeki işlemcileri kullanarak işlem yapmakta, hem de mevcut CPU'yu kullanarak aynı işlemi tekrarlamaktadır. Her iki farklı yöntem için de işlem süresi Şekil 10'da görülmektedir.

Projektif dönüşüm ile ortorektifikasyon uygulamasının hesaplama sonuçları incelendiğinde, yapılan hesaplamanın GPU süresi 175,593 milisaniye iken, aynı işlem CPU üzerinde yapıldığında 790,991 milisaniye olmaktadır. Aradaki bu fark hesaplandığında, GPU ile yapılan işlemin CPU ile yapılan işleme göre yaklaşık 4,5 kat hızlı olduğu görülmektedir. Bu yapılan uygulama içerisinde kullanılan 4096x4096 boyutunda bir görüntü yerine daha küçük boyutlarda görüntülere aynı işlem uygulandığında CPU'nun performansının iyileştiği Tablo 1'de görülmektedir. Bu da çok yoğun ve tekrarlı hesap gerektiren problemlerde GPU'nun daha hızlı sonuçlar ürettiğini ortaya koymaktadır. Aynı uygulama içerisinde 14430 x 9420 boyutundaki örnek bir görüntü de uygulamaya sokulmuş ve Tablo 1'de gösterilen hız farkına ulaşılmıştır.

Tablo 1. GPU ve CPU ile yapılan uygulamanın hız testi sonuçları.

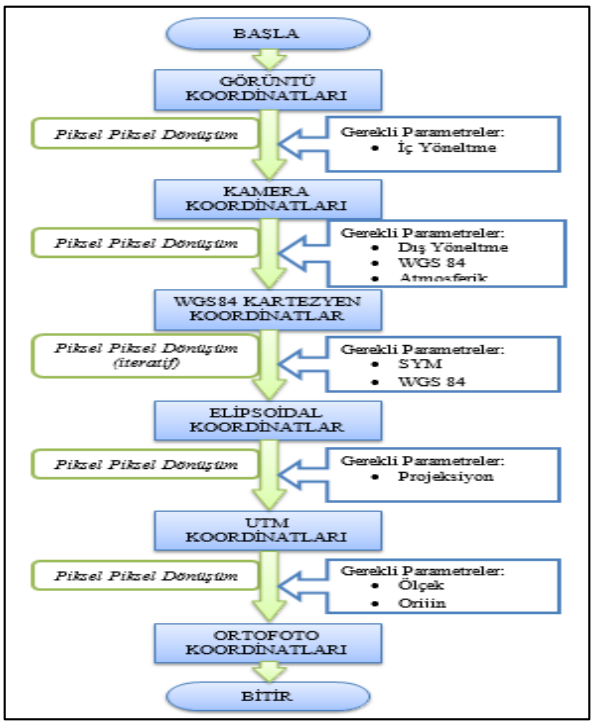
Görüntü Boyutu (piksel)	GPU Zamanı (milisaniye)	CPU Zamanı (milisaniye)	CPU / GPU (oran)
1024 x 1024	14,090	48,917	3,47
2048 x 2048	75,994	190,626	2,51
4096 x 4096	175,593	790,991	4,50
14430 x 9420	414,238	2823,654	6,82

5. CUDA İLE DİFERANSİYEL ORTOREKTİFİKASYON

Diferansiyel ortorektifikasyon, kaynak görüntünün ortogonal bir projeksiyonla piksel bazında referans düzlemine izdüşümü olarak kısaca ifade edilebilir. Bu noktada gerçek zamanlı ortorektifikasyona ihtiyaç duyulan çalışmalarda, görüntülerin doğrudan ortorektifikasyonu aşamasında diferansiyel ortorektifikasyonun kullanılabileceği değerlendirilerek, çalışma içerisinde diferansiyel ortorektifikasyon yöntemi de kullanılmıştır.

Uygulama, projektif rektifikasyonda olduğu gibi, görüntüyü oluşturan her bir piksel için tekrarlanmaktadır. Böylelikle de CUDA programlama dili kullanılarak problem iş parçalarına bölünüp dağıtık bir şekilde, birçok grafik işlemciye aynı anda verilip, hesaplama sonuçları elde edilebilecek yapıdadır.

Bu noktadan hareketle, diferansiyel ortorektifikasyon için ortaya konulan algoritma, CUDA programlama dili ile kodlanmış ve diferansiyel ortorektifikasyonu gerçekleştiren bir uygulama ortaya çıkarılmıştır. Uygulama ile diferansiyel ortorektifikasyon hem grafik kartı üzerindeki işlemciler kullanılarak yapılabilmekte, hem de mevcut CPU kullanılarak yapılabilmektedir. Her iki farklı yöntem için de işlem süreleri Tablo 2'de verilmiştir.



Şekil 11. Diferansiyel ortorektifikasyon iş akışı (Karslıoğlu vd., 2005).

Tablo 2. GPU ve CPU ile yapılan uygulamanın hız testi sonuçları.

Görüntü Boyutu (piksel)	GPU Zamanı (milisaniye)	CPU Zamanı (milisaniye)	CPU / GPU (oran)
1024 x 1024	84675	324563	3,83
2048 x 2048	342634	934547	2,73
4096 x 4096	1328234	6827282	5,14
14430 x 9420	3824302	28129128	7,35

6. SONUÇ VE ÖNERİLER

Yapılan bu çalışmada grafik kartlarının yani GPU'ların GPGPU yönteminin genel amaçlı paralel programlama ve hesaplama gücünün fotogrametrik çalışmalar amacıyla ve özellikle ortorektifikasyon uygulamalarında nasıl kullanılabildiği üzerinde durulmuş, uygulamalar yapılmış ve çeşitli sonuçlar elde edilmiştir.

Çalışma içerisinde yapılan uygulamalardan da görüleceği gibi, etkili bir GPU programlama, mevcut program algoritmasının, donanımın yapısını ve sınırlamalarını da dikkate alan grafik terimlerini kullanarak tekrardan yazılmasını

gerektirmektedir. Halen çok çekirdekli işlemcilerin programlanabilmesi, geleneksel programlama yöntemleriyle gerçekleştirilememektedir. Tipik olay yordamlı programlama yönteminin birden fazla çekirdekli işlemcilerin programlanması için kullanılması mümkün olamamaktadır.

Yapılan uygulamalarda elde edilen sonuçlar değerlendirildiğinde, yöntemin fotogrametrinin görüntü işlemeyi gerektirdiği ve aynı işlem adımlarının her bir piksel için tekrarlandığı durumlarda ve ayrıca hesap yoğun işlem adımlarında çok etkili ve hızlı sonuçlar verdiği görülmektedir. Özellikle ortorektifikasyon amacıyla yapılan uygulamalarda aynı donanımla CPU'ya oranla 7 kat hız farklarına ulaşılmıştır.

Sonuçlar biraz daha detaylı incelendiğinde görüntü boyutu büyüdükçe CPU'nun performansının düştüğü, GPU'nun ve GPGPU yönteminin performansının arttığı görülmektedir. Yapılan bu uygulamaların sayısı ve çeşitliliği arttırılabilir. Hızın ve hızlı karar verme sürecinin etkili olduğu, hesaplama işleminin çok yoğun olarak kullanıldığı problemler için benzer yöntemlerin uygulanabilir olduğu görülmektedir.

Günümüzde harita üretiminde, özellikle görüntüler üzerinden değerlendirme sonuçlarının hızla elde edilmesi ve karar vericilere aktarılmasının gerektiği felaket senaryolarında, orman yangınları ve depremler gibi yaşanan doğal afetlerde ve kriz durumlarında, özellikle askeri açıdan hedef istihbaratı ve hedef konumunun hızlıca tespitinde çok çeşitli platformlardan elde edilen görüntüler sıklıkla kullanılmaktadırlar. Bu görüntülerin anlamlı hale gelmesi ve belirtilen uygulamalarda kullanılabilmesi için öncelikli olarak yapılması gereken işlem adımı; bu görüntülerin ortorektifikasyonudur.

Özellikle son zamanlarda yaygınlaşarak çeşitli kurum ve kuruluşlar tarafından birçok uygulamada kullanılmaya başlanan İHA'lar yardımıyla elde edilen görüntüler üzerinden çok hızlı karar vermek, çıkarımlar yapmak ve çeşitli hedef tespiti yaparak bu hedefleri de doğru koordinatlarla tarifleyebilmek için elde edilen görüntülerin yöneltilmesi yani ortorektifiye edilmesi ihtiyacı bulunmaktadır. Bu amaca yönelik olarak çekilen görüntüler bir depolama ünitesinde depolanmakta ve ardından elde edilen görüntüler İHA yere indirildikten sonra yerdeki istasyonlarda yer alan güçlü işlemcilere ve donanıma sahip bilgisayarlar yardımıyla ortorektifikasyon işlemine tabi tutularak koordinatlandırılmakta ve bu aşamadan sonra da koordinatlı olarak kullanılabilir hale gelmektedir. Bu kapsamda uygulama içerisinde ortaya konulan yöntemle; GPU'ların genel amaçlı paralel programlama ve hesaplama gücünü kullanarak GPGPU yöntemiyle sayısal hava kameraları ile elde edilen görüntülerin, insansız hava araçları gibi çok çeşitli platformlardan elde edilebilecek görüntülerin hızlı bir şekilde gerçek zamanlı ortorektifikasyonunun yapılabileceği değerlendirilmektedir.

Çalışma kapsamında yapılan ve ortaya konulan uygulamalarda yazılan CUDA kodu içerisinde herhangi bir optimizasyon çalışması yapılmamıştır. Literatür incelendiğinde GPGPU'nun uygulandığı başka uygulamalarda CUDA kodunun optimize edilmesiyle performansın arttığı görülmüştür. Çalışma kapsamındaki uygulama içerisinde de şayet bu optimizasyon gerçekleştirilmiş olsaydı, uygulamaların performansının artması kaçınılmaz olacaktır. İlerleyen çalışmalarda bu optimizasyonun da yapılarak, uygulamaların performans değerlerinin artırılabileceği değerlendirilmektedir.

Yapılan bu çalışmayla, daha önce çeşitli platformlardan elde edilen görüntülerin ortorektifikasyonu için kullanılmamış olan bir yöntem ile donanım altyapısı kullanılmış ve programlama algoritması uygulanmıştır. Çalışma içerisinde gerçekleştirilen uygulamalar ile GPGPU yöntemi kullanılarak görüntülerin çok hızlı ve donanım olarak çok daha ekonomik bir şekilde ortorektifikasyonunun yapılabileceği değerlendirilmektedir. Böylece, bu çalışmayla halen anlık karar verme ve hedef tespiti için çok önemli bir unsur olan zaman faktörünün en aza indirilmesi için bir yöntem ortaya konulmuş ve bu yöntemin geliştirilmesi için de bir başlangıç yapılmıştır.

K A Y N A K L A R

Bettemir O.H., (2006), **Sensitivity and Error Analysis of a Differential Rectification Method for CCD Frame Cameras and Pushbroom Scanners**, Master Thesis, METU, Ankara.

Biesemans, J and Everaerts, J., (2006), **Image Processing Workflow for the Pegasus HALE UAV Payload**, International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, Antwerp, Belgium.

Gruen, A. and Beyer, H., (2001), **Calibration and Orientation of Cameras in Computer Vision**, Springer Series in Information Sciences. Vol. 34, Springer-Verlag Berlin Heidelberg.

Jacobsen, K., (2002), **Calibration aspects in direct georeferencing of frame imagery**, In: Int. Archives PhRS (34), 1 I, pp. 82-89, Denver.

Karslioglu, M.O., Friedrich J., (2005), **A New Differential Geometric Method to Rectify Digital Images of the Earth's Surface Using Isothermal Coordinates**, IEEE Transactions on Geoscience and Remote Sensing, Vol. 43, No. 3, March.

Kiracı, A.C., (2008), **Direct Georeferencing and Orthorectification of Airborne Digital Images**, Master Thesis, METU, Ankara.

Kraus, K., (2007), **Fotogrametri Cilt 1 Fotoğraflardan ve Lazer Tarama Verilerinden Geometrik Bilgiler**, Istanbul Technical University, Nobel Yayın Dağıtım, 1.Basım.

Mercedes Marqu´es, Gregorio Quintana-Ort´ı, Enrique S. Quintana-Ort´ı, Robert van de Geijn, (2009), **Using graphics processors to accelerate the solution of out-of-core linear systems**, 8th IEEE International Symposium on Parallel and Distributed Computing, Lisbon.

Novak, K., (1992), **Rectification of Digital Imagery**, Photogrammetric Engineering and Remote Sensing, 339-344.

Nvidia, (2011a), **OpenCL Programming Guide for the CUDA Architecture**, Nvidia Corp., California, USA.

Nvidia, (2011b), **CUDA C Programming Guide**, Nvidia Corp. California, USA.

Nvidia, (2015), **CUDA Architecture, Introduction and Overview**, Nvidia Corp., California, USA.

Schwarz, K., P. (1993), **IntegratedAirborne Navigation System for Photogrammetry**, Photogrammetric Week'95, Wichmann, Germany.

Schwarz, K., P., Chapman, M., E., Cannon, E., Gong, P. (1993), **An Integrated INS/GPS Approach to the Georeferencing of Remotely Sensed Data**, Photogrammetric Engineering & Remote Sensing, 59(11), (pp.1667-1674).

Skaloud, J. (2002), **Direct Georeferencing in Aerial Photogrammetric Mapping**, Photogrammetric Engineering & Remote Sensing, 68(3), (pp.207-210).

Şahin, H. & Külür, M., S. (2011), **GPGPU Yöntemi İle Fotogrametrik Uygulamalar**, Türkiye Ulusal Fotogrametri ve Uzaktan Algılama Birliği VI. Teknik Sembozyumu TUFUAB, Antalya, Türkiye; Şubat 21-25.

Şahin, H. & Külür, M., S. (2012), **Orthorectification By Using GPGPU Method**, International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XXXIX-B4, Melbourne, Australia; August 25-September 01.

U. Thomas, F. Kurz, D. Rosenbaum, R. Mueller, P. Reinartz, (2008), **GPU-based Orthorectification of Digital Airborne Camera Images in Real Time**, The International Archives Of The Photogrammetry, Remote Sensing And Spatial Informatıon Sciences, ISPRS Congress Beijing, Volume XXXVII Part B1 Commission I.

White, S. and M Aslaksen, (2006), **Use of Direct Georeferencing to Support Emergency Response**,. NOAA's PERS Direct Georeferencing Column.

Yastıklı, N., (2003), **GPS/IMU Verilerini Kullanarak Hava Fotoğraflarının Doğrudan Yöneltilmesi ve Birleştirilmiş Blok Dengeleme Olanakları**, Doktora Tezi, Yıldız Teknik Üniversitesi, İstanbul.

Yılmaz, E., (2010), **Massive Crowd Simulation with Parallel Processing**, PhD Thesis, Information Systems Department, METU, Ankara.