EJONS International Journal on Mathematic, Engineering and Natural Sciences | ISSN 2602 - 4136

Article Arrival Date: 23.06.2021 | Article Type: Research Article | Article Published Date: 20.09.2021

Doi Number: http://dx.doi.org/10.38063/ejons.446

CUDA İLE YÜKSEK BAŞARIMLI BİR YÜZ TANIMA UYGULAMASI

A HIGH PERFORMANCE FACE RECOGNITION APPLICATION WITH CUDA

Fikriye ATAMAN

Van Yüzüncü Yıl Üniversitesi, Enformatik Bölümü, Van / Türkiye.

ORCID: 0000-0002-0257-7730

ÖZ

Son yıllarda hızla gelişen GPU teknolojisi özellikle araştırmacıların ve paralel yazılım geliştiricilerinin dikkatini çekmiştir. Grafik kartlarındaki işlemcilerin (GPU) genel amaçlı hesaplamalar için kullanılabilmesine imkân sağlayan NVIDIA CUDA programlama ara yüzlerinin geliştirilmesi paralel programlama ile ilgilenen herkesi heyecanlandırmıştır. Genel amaçlı GPU olarak adlandırılan bu işlemciler, doğasında yüksek derecede veri paralelliği barındıran tıbbi görüntü işleme gibi farklı görüntü işleme uygulamalarında ve 3D modelleme gibi birçok alanda kullanılmaya başlanmıştır. Bu çalışmada ilk olarak, NVIDIA firmasının CUDA destekli grafik kartlarının görüntü işleme alanında yüksek performanslı uygulamalar geliştirmeyi nasıl kolaylaştırdığı incelenmiştir. Buna ek olarak C++ programlama dili ile CUDA tabanlı bir paralel yüz tanıma uygulaması da hayata geçirilmiştir. Söz konusu uygulamanın seri (CPU) ve paralel (CUDA) sürümleri performans açısından karşılaştırılmış ve çıkan bulgular ayrıntılı olarak sunulmuştur. Yüz tanıma için Temel Bileşen Algoritmasından faydalanılmıştır.

Anahtar kelimeler: CUDA, Paralel Programlama, Yüz Tanıma, TBA, Özyüzler.

Abstract

Developing rapidly in recent years, GPU technology attracted especially the attention of researchers and paralel software developers. Development of the interfaces programming such as NVIDIA CUDA that made it possible to use GPUs in graphic cards, for general-purpose computations, get excited everyone studying on parallel programming. This new programming platform ,which is also named GPU for general purposes , has started to be used in many different image processing applications such as medical image processing,- in which it requires a high degree of data parallelism- and 3D modeling. In this study , first it is examined how , the CUDA based graphic cards of the NVIDIA firm, can make it easier to develop high performanced applications in the field of image processing then CUDA based parallel application of face recognition- in which C++ programming language is used, has also been implemented. Serial (CPU) and parellel (CUDA) types of the application is compared according to their performance and the results are presented with detailed statistics. The Principal Component Analysis algorithm is used for face recognition.

Keywords: CUDA, Parallel Programming, Face Recognition, PCA, Eigen Faces.

1. Giriş

Biyometrik analiz araştırmaları ivme kazanan veri üretimine karşın gittikçe önem kazanmaktadır. Tanımlama amaçlı en çok kullanılan biyometriler: Ses, parmak izi, retina, iris, el geometrisi, imza ve yüz analizleridir [1-4].

Biyometrik tanımlama yöntemleri birçok organ üzerinde yapılmaktadır. Yüz tanıma sistemleri insan tarafından kabul edilmesi ve kolay uygulanması itibari ile geçerliliğini kabul ettirmiştir [1, 5, 6]. Yüz tanıma sistemlerinin güvenilir bir biyometrik tanımlama olarak kabul görmesi sonucu birçok alanda kullanımı yaygınlaşmıştır. Ancak yüz tanıma sistemlerinde kullanılan biyometrik resimlerin boyutlarının daha büyük olması nedeniyle genel bir performans sıkıntısı bulunmaktadır [1, 7].

Tablo 1'de İngiltere Pasaport Servisi tarafından yürütülen ve Nisan-Aralık 2004 dönemleri arasında yapılan ve 10.000 kişiyi kapsayan Biyometrik Kayıt Deneyi sonuçları görülmektedir [2, 5]. Tablo 2'de ise biyometrik teknolojiler karşılaştırmalı olarak yer verilmektedir.

Tablo 1. Biyometrik teknolojilerin karşılaştırılması [5].

Biyometrik Özellikler	Doğruluk	Güvenlik	Sosyal Kabul Edilirlik
Yüz	Y	Y	ÇY
Parmak İzi Eşleme	ÇY	Y	D
El Geometrisi	Y	D	D
Retina Tarama	ÇY	ÇY	ÇD
Ses Tanıma	D	D	Y
İmza Karşılaştırma	D	D	ÇY
İris Tanıma	ÇY	ÇY	ÇD
Y:Yüksek, ÇY: Çok Yüksek, D:Düşük, ÇD: Çok Düşük			

Tablo 2. Biyometrik teknolojilerin karşılaştırılması [1].

Biyometrik Özellik	Evrensellik	Benzersizlik	İstikrar	Elde Edilirlik	Performans	Kabul Edilirlik	Tuzağa Düşürme
Yüz	Y	D	O	Y	D	Y	D
Parmak İzi	O	Y	Y	O	Y	O	Y
El Geometrisi	O	O	O	Y	O	O	O
Tuş Vuruşları	D	D	D	O	D	O	O
El Damarları	O	O	O	O	O	O	Y
İris Tanıma	Y	Y	Y	O	Y	D	Y
Retina Tarama	Y	Y	O	D	Y	D	Y
İmza	D	D	D	Y	D	D	D

Biyometrik Özellik	Evrensellik	Benzersizlik	İstikrar	Elde Edilirlik	Performans	Kabul Edilirlik	Tuzağa Düşürme
Ses Tanıma	O	D	D	O	D	Y	D
Y:Yüksek, O:Orta, D:Düşük							

Yüz tanıma sistemlerindeki performans düşüklüğüne yol açan yüksek boyutlu resim verisidir [1, 2, 8]. Yaptığımız bu çalışmada, resim verisi boyutlarının indirgenmesine olanak sağlayan özyüzler yaklaşımı tercih edilerek, Özyüzler algoritması CUDA teknolojisi ile parelleştirilmesi sonucunda performans artışı sağlanmaya çalışılmıştır.

1.1. Özyüzler Yaklaşımı

Öz yüzler yaklaşımı, Temel bileşen analizinin (TBA) yüz tanımaya uyarlanmış halidir ve TBA, Karhunen-Louve teoreminin genişletilmiş halidir. TBA teoremi veri analizinde yüksek boyutlu verilerde veri boyutunu düşürerek, iyi sonuçlar vermektedir. Öz yüzler yaklaşımı ilk kez Sirovich ve Kirby tarafından yüz tanıma alanında, etkin yüz gösterimi amacıyla kullanılmıştır [9, 10]. Sirovich ve Kirby, yaptıkları çalışmada yüz resimleri gruplarından başlayarak bu resimlerin temel bileşenlerini hesaplamışlardır. Daha sonra da yüz resmini yeniden oluşturmak için öz vektörün sadece küçük parçalarının ağırlıklı birleşimi kullanılmıştır [9-11]. Bu metotlarını 115 yüz resmi üzerinde test etmişler ve ortalama %3 hatayla 40 öz vektörün bir yüzü yeniden oluşturmak için yeterli olduğunu göstermişlerdir. Daha sonra yüzün simetrisini temel alarak metotlarını son hale getirmişlerdir. Çalışmalarının son hali olan algoritmayı 87 kişiye ait resim içeren bir veri seti üzerinde test etmişlerdir [9-11].

Turk and Pentland (1991), daha sonra bu algoritmayı daha da geliştirerek ilk tam otomatik sistem olarak sunmuşlardır [10-12].

Turk ve Pentland tarafından geliştirilen analizin iki boyutlu veriler üzerinde uygulanması özyüzler yaklaşımı olarak adlandırılır. Biyometrik tanımlama uygulamalarının çoğunda özyüzler yöntemi sıklıkla kullanılmaktadır. Özyüzler yöntemi ile görüntü boyutları küçültülerek aynı sonuçlara ulaşılmaktadır[10-12]. Veri boyutunun düşürülmesi iş yükünün azaltılmasını ve dolayısıyla performans artışını sağlamaktadır. Temel bileşenler analizi bu amaçla sıkça başvurulan bir yöntem olmuştur. [12].

2. Materyal ve Metot

2.1. Kullanılan donanım ve özellikleri

Bu çalışmada Intel i5 işlemcili bir dizüstü bilgisayar kullanılmıştır. İşletim sistemi olarak Windows 8.1. Single Language versiyonu yüklüdür. Grafik işlemcisi ise GeForce GT 740M'dir. İşleme kapasitesi 3.5'dir. GeForce GT 740M grafik kartına ait ayrıntılar Tablo 3'de verilmektedir.

Tablo 3. Kullanılan donanımın özellikleri

İşlemci	Intel(R) Core(TM) i5-3337U CPU @ 1.80GHz
Mimari	x64
Frekans	1.796 MHz
Çekirdek Sayısı	4
Sayfa Boyutu	4.096
Toplam Fiziksel Bellek	3.974,00 MB
Kullanılabilir Fiziksel Bellek	1.790,00 MB
İşletim Sistemi Versiyonu	Windows 8.1 Single Language
Sürüm Numarası	6.2.9200

2.2. Özyüzler yaklaşımı ile yüz tanıma

Özyüzler yöntemi ile iki boyutlu görüntülerin tanımlanması yüksek doğrulukla mümkündür [2, 10, 12]. Veri tabanındaki görüntülerin kovaryans matrisi hesaplanır ve kovaryans matrisinin özdeğerleri elde edilir. En büyük özdeğerlere sahip özvektörler bulunur ve bunlar özyüzler olarak adlandırılır. Kovaryans matrisi simetrik bir matristir. Dolayısıyla hesaplanan özvektörler ile görüntü uzayındaki vektörler birbirine ortogonaldır. Bu sayede her bir görüntü, özyüzlerin doğrusal bileşimi ile yeniden oluşturulabilir. Oluşturulan her bir özvektörün, her bir görüntü üzerine izdüşümünün alınması ile her bir görüntüye ait ağırlık vektörleri hesaplanır. Görüntünün tanımlanmasında bu ağırlık vektörleri kullanılır [2, 10, 11].

Temel Bileşenler Analizi (TBA) görüntüler arasındaki temel farklılıkları ortaya çıkarmak için boyut azaltan bir tekniktir [12]. İlişkili olan verilerin bilgi kaybı olmadan daha düşük bir boyut ile istenilen sonuca ulaşılmasını sağlar [12]. Öz yüzler yöntemi aşağıda adım adım anlatılmaktadır.

1. Öncelikle veri tabanında kayıtlı olan *nxm* boyutlu görüntüler satır vektörüne dönüştürülür. Her bir resim için resimdeki her bir satır alınır uç uca eklenerek resim bir vektöre dönüştürülür. *nxm=N* olarak alınırsa, *M* tane *N* boyutlu yüz vektörü olur. Burada *M* resim sayısını *N* ise resimdeki piksel sayısını ifade eder.

$$Görüntü = I1 = \begin{pmatrix} I1_{11} & \cdots & I1_{1m} \\ \vdots & \ddots & \vdots \\ I1_{n1} & \cdots & I1_{nm} \end{pmatrix} \quad (1)$$

$$I1 = \begin{bmatrix} I1_{11} \\ I1_{12} \\ \vdots \\ I1_{1m} \\ I1_{21} \\ I1_{22} \\ \vdots \\ I1_{2m} \\ \vdots \\ I1_{nm} \end{bmatrix} \quad (2)$$

2. Elde edilen tüm görüntülerden ortalama görüntü elde edilir [12]. Her bir ortalama pikselin hesaplanması sonucu ortalama görüntü elde edilmektedir.

Γ: görüntü vektörü,

Ψ: ortalama görüntü vektörü,

M: toplam görüntü sayısı, olmak üzere,

$$\psi = \frac{1}{M}\sum_{n=1}^{M}\Gamma_n, \quad \psi = \frac{1}{M}\begin{pmatrix} \vec{I}1_1 + \vec{I}2_1 + \vec{I}3_1 + \cdots + \vec{I}M_1 \\ \vec{I}1_2 + \vec{I}2_2 + \vec{I}3_2 + \cdots + \vec{I}M_2 \\ \vec{I}1_3 + \vec{I}2_3 + \vec{I}3_3 + \cdots + \vec{I}M_3 \\ \vdots \\ \vec{I}1_n + \vec{I}2_n + \vec{I}3_n + \cdots + \vec{I}M_n \end{pmatrix} \tag{3}$$

3. Her bir görüntü vektöründen ortalama görüntü vektörü çıkarılarak elde edilen vektörlerden bir görüntü matrisi oluşturulur. Bu matrisin ortalaması sıfırdır. Böylelikle her görüntü ortalamaya göre merkezlenmiş olur [12].

$$\Phi_i = \Gamma_i - \psi \tag{4}$$

Bu vektörlerden oluşturulan veri matrisi A aşağıdaki eşitlikler ile ifade edilir[2].

$$A = [\Phi_1, \Phi_2, \Phi_3, \Phi_4, ..., \Phi_M] \tag{5}$$

$$\vec{I}1_m = \begin{pmatrix} \vec{I}1_1 - m_1 \\ \vec{I}1_2 - m_2 \\ \vec{I}1_3 - m_3 \\ \vdots \\ \vec{I}1_{N^2} - m_{N^2} \end{pmatrix}, \vec{I}2_m = \begin{pmatrix} \vec{I}2_1 - m_1 \\ \vec{I}2_2 - m_2 \\ \vec{I}2_3 - m_3 \\ \vdots \\ \vec{I}2_{N^2} - m_{N^2} \end{pmatrix}, \cdots, \vec{I}M_m = \begin{pmatrix} \vec{I}M_1 - m_1 \\ \vec{I}M_2 - m_2 \\ \vec{I}M_3 - m_3 \\ \vdots \\ \vec{I}M_{N^2} - m_{N^2} \end{pmatrix} \tag{6}$$

olmak üzere;

$$A = [\vec{I}1_m, \vec{I}2_m, \vec{I}3_m, \cdots, \vec{I}M_m][2] \tag{7}$$

4. Bulunan A veri matrisi devriği ile çarpılarak kovaryans matrisi elde edilir. Bunun için eşitlik 8 kullanılır [12]

$$C = \frac{1}{M} \sum_{n=1}^{M} \Phi_n . \Phi_n^T = A.A^T \tag{8}$$

Dikkat edilmesi gereken bir husus şudur: Kovaryans matrisinin boyutu, görüntü vektörünün uzunluğuna bağlı oluşmaktadır. N^2'lik görüntü vektörlerinden oluşan kovaryans matrisinin boyutu N^2xN^2 olacaktır. Resim görüntü vektörünün boyutu (N^2) toplam görüntü sayısından büyük olursa bu kadar büyük bir matrisin özdeğer ve özvektörlerinin hesaplanması oldukça zor olacaktır. N^2 x N^2 boyutlu kovaryans matrisinden elde edilen N^2 sayıdaki özvektörün yalnızca *M-1* tanesi geçerli olacaktır. Çünkü diğer özvektörler, sıfır değerine sahip özdeğerlere karşılık gelir. Bu yüzden anlamlı olan özvektör sayısı veri tabanındaki görüntü sayısına bağlıdır. Dolayısıyla N^2xN^2ebatlı bir matrisle çalışmak yerine *MxM* ebatında bir matrisle çalışmak daha uygun olmaktadır. Kovaryans matrisinin hesaplanmasında temel olarak eşitlik 8 yerine, eşitlik 9 kullanılabileceği bildirilir [12].

$$C = \frac{1}{M} \sum_{n=1}^{M} \Phi_n^T . \Phi_n = A^T . A \tag{9}$$

5. Kovaryans matrisinin özvektörleri ve bu özvektörlere karşılık gelen özdeğerleri hesaplanmaktadır. Kovaryans matrisinin özvektörlerini v_i ve özdeğerlerini de μ_i ile gösterilirse eşitlik 10'a ulaşılmaktadır. Bu eşitliğin her iki tarafı A matrisi ile çarpılırsa eşitlik 11'e ulaşılır ve bu eşitlikten $A.A^T$ ifadesi kovaryans matrisini ifade etmektedir. Dolayısıyla kovaryans matrisin öz vektörleri $A.v_i$ olarak çıkarmaktadır. Özvektörler artık özyüzler olarak adlandırılmaktadır. Özyüzlerden oluşan matris de görüntü uzayı olarak ifade edilir [12].

$$A^T . A v_i = \mu_i . v_i \tag{10}$$

$$A.A^T . A v_i = \mu_i . A.v_i \tag{11}$$

6. Tanımlama için her bir görüntünün görüntü uzayı üzerinde izdüşümü bulunarak birer ağırlık vektörü hesaplanır. Bu ağırlık vektörleri bir matriste toplanarak tanımlama için esas olan ağırlık matrisi elde edilir [12].

$$w_k = V_k^T . A \tag{12}$$

$$\Omega = [w_1^T, w_2^T, w_3^T, \cdots, w_M^T] \tag{13}$$

2.3. Özyüzler ile test edilen resmin tanınması

1. Resim matrisi görüntü vektörüne çevrilir[12].
2. Görüntü vektöründen ortalama görüntünün farkı alınır[12].

3. Bu fark herbir özyüz üzerine düşürülerek her özyüz için ağırlık vektörleri hesaplanır[12]. Test ağırlık vektörlerinin hesaplanması eşitlik 14 ile gösterilmektedir.

$$w_k = V_k^T . \Phi_{Test} = V_k^T . (\Gamma_{Test} - \psi) \quad (14)$$

4. Test görüntüsüne ait ağırlık vektörünün hesaplanır[12].

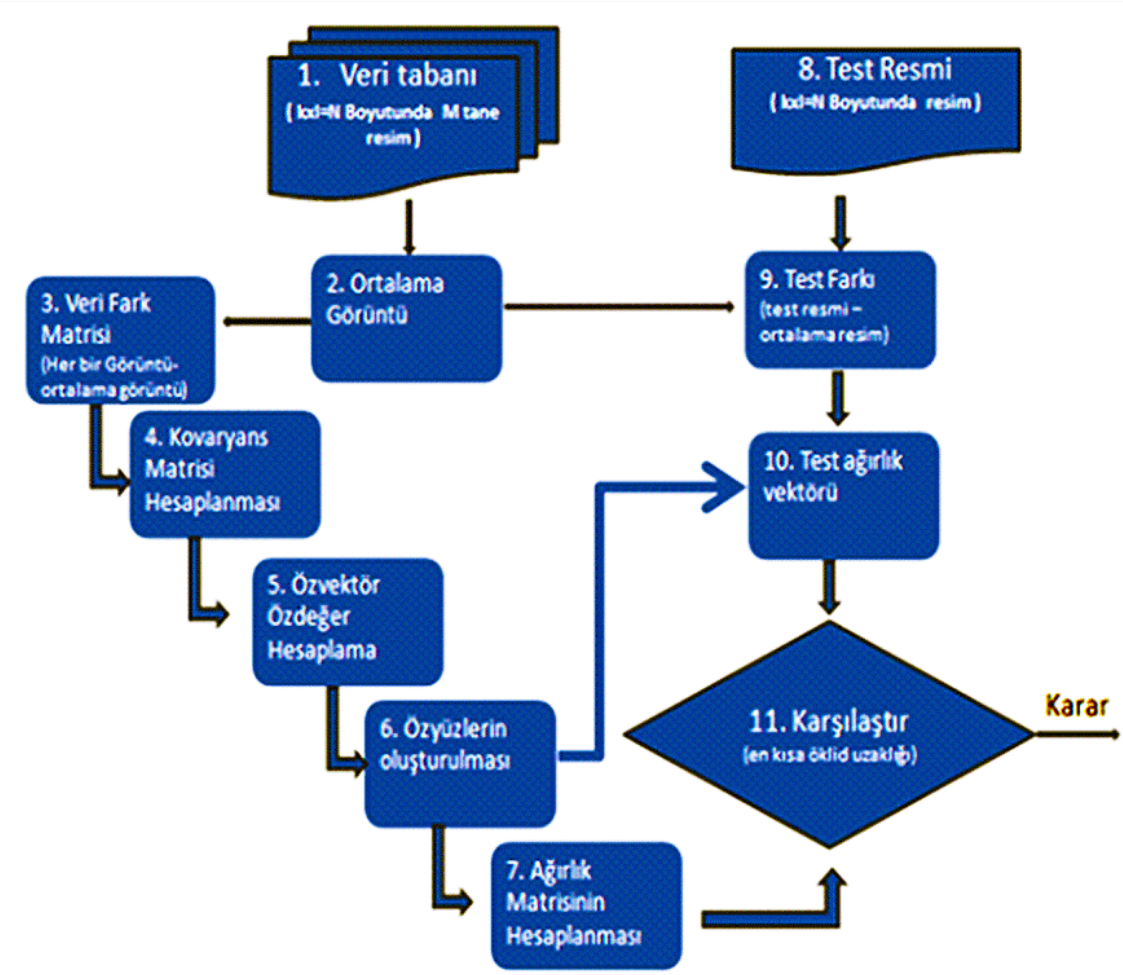
$$\Omega_{Test} = [w_1, w_2, w_3, \cdots, w_M] \quad (15)$$

5. Yakınlık mesafesi öklid eşitliği ile hesaplanarak, tanımlama yapılır. Tanımlamada görüntü uzayındaki her bir görüntünün ağırlık matrisine olan mesafesi esas alınır[12].

$$\varepsilon_k^2 = P(\Omega - \Omega_k).P^2 \quad (16)$$

Ağırlık matrisi ile görüntüler arasında hesaplanan Öklid mesafeleri içerisinde en küçük olan mesafeye denk gelen görüntü en uyumlu görüntü olarak tanımlanır. Dolayısıyla tanımaya karşılık gelen görüntüdür[12].

Şekil 1'de TBA algoritmasının işlem akış şeması gösterilmektedir. TBA algoritmasında matris çarpımı, matris devrik işlemi ve ortalama alma işlemleri bulunmaktadır. Nvidia CUDA mimarisindeki grid yapısı ile matris yapısı benzer yapıda olduğundan matris işlemlerin uygulanması için uygun bir ortam sunulmaktadır. Bu çalışmada da matris çarpım, ortalama, devrik alma vb. işlemleri CUDA teknolojisi ile paralel programlanmıştır.

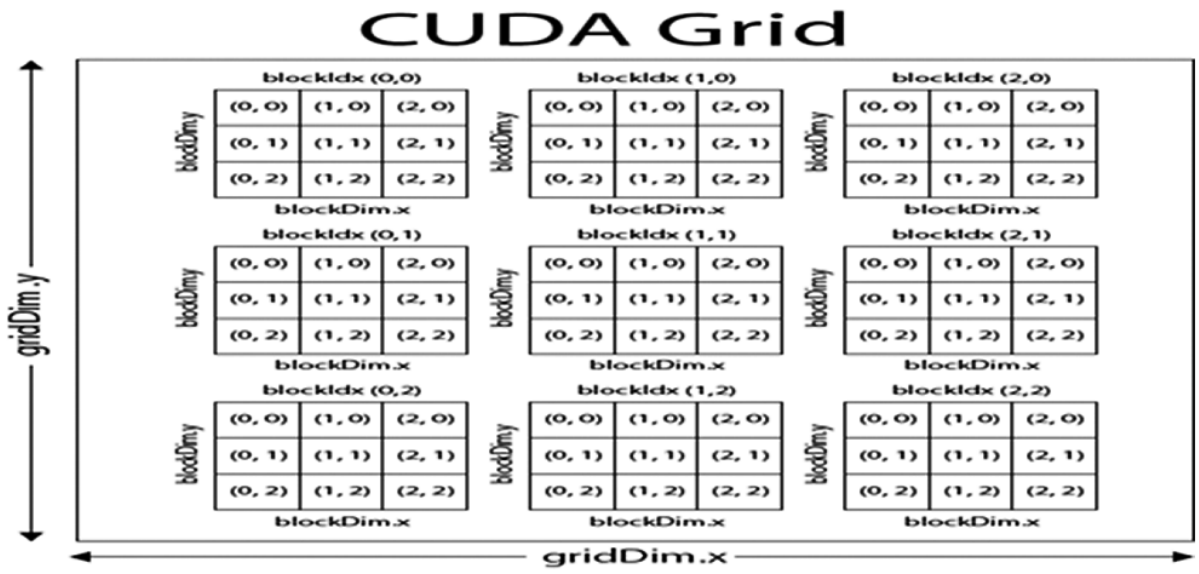


Şekil 1. Özyüz algoritması akış diagramı

2.4. Paralelleştirme işlemleri

Bu çalışmada görüntü işleme alanında bir yüz tanıma-tahminleme uygulaması üzerinde çalışma yapılmıştır. TBA algoritması CUDA teknolojisi ile C++ programlama dili kullanılarak paralel olarak programlanmıştır. Tüm programlama sürecinde tek boyutlu diziler üzerinde çalışılmıştır. Resimler uygun olarak tek boyutlu dizilere dönüştürülerek işlemler gerçekleştirilmiştir. TBA algoritmasındaki adımlar önce seri olarak programlanmış daha sonrada aynı fonksiyon veya program parçacıkları CUDA-C dili ile paralel programlanarak, seri-paralel uygulamaların performans karşılaştırılmaları yapılmıştır. Tek boyutlu diziler üzerinde çalışıyor olmak genel performansı olumsuz yönde etkilemektedir. Ancak dikkat çekmeye çalışılan nokta paralel ve seri programlamadaki performans ölçümlerindeki farktır. Bunun yapılması için program içerisinde yapılan her süre ölçümü kayıt altına alınmıştır.

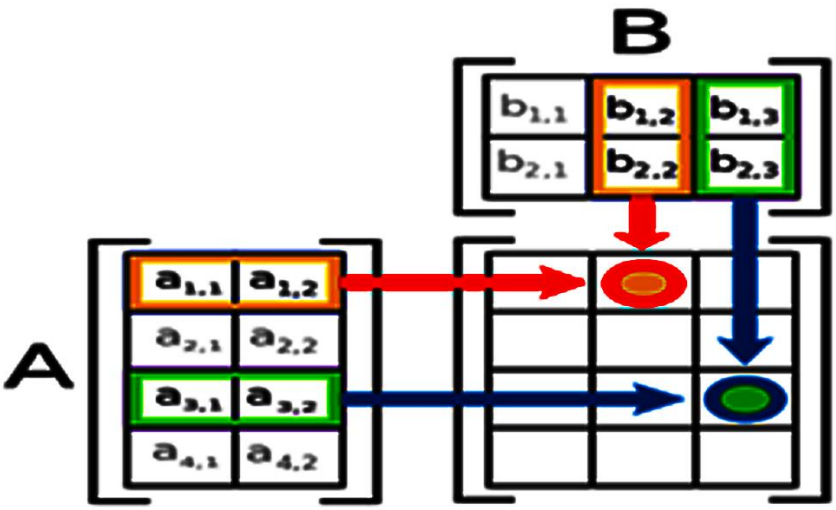
Paralelleştirme işlemlerinde CUDA grid yapısı ve çalıştırma mantığı Şekil 2 ile özetlenmiştir.



Şekil 2. CUDA grid yapısı [13]

2.5. Matris çarpım çekirdeği

Matris çarpım işlemlerinde Kernel_Carpim çekirdeği kullanılmıştır ve çarpılacak matrislerin satır ve sütunlarını alt matrisler olacak şekilde alınarak bunların çarpımlarını yapılmıştır. Her bir iplik, sonuç olarak çıkacak C matrisinin bir elemanını hesaplamak üzere programlanmıştır. Şekil 3. ile matris çarpımı gösterilmektedir.



Şekil 3. Matris çarpımının gösterimi [14]

"dim3 ipliksayisi(BLOCK_SIZE, BLOCK_SIZE);" ile BLOCK_SIZE boyutunda bir blok boyutu tanımlanmıştır. Bir blokta kaç tane iplik olacağı belirtilmiştir.

"dim3 bloksayisi(B.en / ipliksayisi.x, A.boy / ipliksayisi.y);" ile kaçtane blok üzerinde işin paylaştırılacağı belirtilmiştir. Bu çarpılacak matrislerin boyutuna göre değişmektedir.

"Kernel_Carpim<<<bloksayisi,ipliksayisi>>>(d_A, d_B, d_C);" ile de çekirdek çağırılmaktadır.

2.6. Uygulama detayları

Bu çalışma kapsamında 80 x 96 piksel boyutunda resimler üzerinde çalışılmış ve her bir resim 80x96=7680 tek boyutlu bir diziye aktarılmıştır. Bu satır vektörlerinin yani 32 resmin (tüm resimlerin) bir diziye aktarımı ile 7680x32 boyutunda bir matris oluşturulmuştur. Bu matrisin transpozu ile çarpımı sonucu 7680x7680'lik bir görüntü matrisi oluşturularak bu çarpım işlemindeki her bir eleman Şekil 4'deki kernel ile paralel olarak hesaplanmıştır.

A matrisinin boyutu 7680x32 ise A.en=32,A.boy=7680'dir.

B matrisinin boyutu 32x7680 ise B.en=7680,B.boy=32'dir.

C sonuç matrisinin boyutu 7680x7680'dir.

Bu matrisin16'lık bloklar üzerinde paylaştırılması durumunda, ipliksayısı=(16,16) olur. Böylece her bir blokta 16*16=256 iplik bulunmaktadır. Blok sayısı ise (7680/16,7680/16)=(480,480) olmaktadır. Bu şekilde grid içerisinde 480x480 tane blok ve her bir blokta 16x16 iplik bulunmaktadır. Toplamda 480x480=230400 blok, ve her bir blokta 16*16=256 tane iplik mevcut olmaktadır. Toplamda iplik sayısı hesaplanacak olursa: 230400*256=58982400 tane iplik hesaplamaya katılmış olmaktadır. C çarpım sonuç matrisi 7680x7680 boyutunda iken, toplam olarak 7680*7680=58982400 tane elemanı bulunmaktadır. Her bir elemana bir iplik verilerek her bir iplik, sonuç matrisinin bir elemanını hesaplayacak şekilde paralel programlanmıştır. Şekil 4' de program parçacıkları gösterilmektedir.

```
__global__ void Kernel_Carpim(Matris A, Matris B, Matris C)
{
    // Bloktaki ilgili satır ve sutunu gösterir.
    int blockRow = blockIdx.y;  int blockCol = blockIdx.x;
    // her bir iplik bloğu C sonuç matrisinin bir alt matrisini
    Matris Csub = GetSubMatrix(C, blockRow, blockCol);
    // Herbir iplik Cnin bir alt matrisinin bir elemanını hesap
    float Cvalue = 0.0;
    // Her bir C alt matrisi için ilgili ipliğin satır ve sutun
    int row = threadIdx.y;  int col = threadIdx.x;
    // Csub alt matrisin değerlerini hesaplamak için A ve B nin
    // Tüm değerler için döngü devam eder...
    for (int m = 0; m < (A.en / BLOCK_SIZE); ++m)
    {

            Matris Asub = GetSubMatrix(A, blockRow, m);// A nın
            Matris Bsub = GetSubMatrix(B, m, blockCol);// B nin
            __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];//A ve
            __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];//için
            As[row][col] = GetElement(Asub, row, col);// Asub a
            Bs[row][col] = GetElement(Bsub, row, col);// herbir
            __syncthreads();// alt matrislerin yüklendiğinden e
            for (int e = 0; e < BLOCK_SIZE; ++e)//Asub ve  Bsub
            Cvalue += As[row][e] * Bs[e][col];
            __syncthreads();// A ve B den bir sonraki iterasyon
    }   //Csub GPU belleğine yazdırılır.
    SetElement(Csub, row, col, Cvalue);// Her iplik bir elemanı
}
```

512

Şekil 4. Matris çarpım çekirdeği

Yüz tanıma için kullanılan yöntem olan TBA algoritmasında, özyüzlerin hesaplanması için kullanılan özdeğer - özvektörlerin hesaplanmasında MATLAB Engine API'den faydalanılmıştır. Özvektör ve özdeğerler için C++ Programlama dili ortamında ilgili matris tek boyutlu bir dizi ile gönderilerek MATLAB Engine API program kodlarıyla MATLAB ortamında hesaplanmakta ve sonuçlar geri çağırılmaktadır. Yüz tanıma uygulamasında grafik arayüzü oluşturmak için OpenCV kütüphanesi kullanılmıştır.

2.7. Devrik alma çekirdeği

Devrik alma işlemi Transpoze isimli bir çekirdek programlanmıştır. Grid içerisinde veri matrisinin (7680*32) ebatlarına uygun olacak şekilde 240x1 boyutunda bloklar tanımlanmıştır. Her bir blok boyutu 32x32 thread olacak şekilde ayarlanmıştır. 240 blok x 1024 thread=245760 thread olmaktadır. Devrik alma işlemlerinde kullanılan veri matrisinin de 7680*32=245760 elemanı mevcuttur. Böylelikle her bir eleman için bir iplik atanarak devrik alam işlemi gerçekleştirilmektedir. Transpoze çekirdeği kodları Şekil 5. ile gösterilmektedir.

Çekirdeğin çağrılma şekli şöyledir:

"dimBlock(BLOCK_SIZE_TRANS, BLOCK_SIZE_TRANS);

dim3 dimGrid(B.en / dimBlock.x, 1);

Transpoze_Paylasimli<<<dimGrid,dimBlock>>>(d_A.elemanlar,d_B.elemanlar,d_A.boy,d_A.en);"

"dimBlock(BLOCK_SIZE_TRANS, BLOCK_SIZE_TRANS);" ile bloklarda kaç iplik olacağı belirtilmektedir. BLOCK_SIZE_TRANS değişkeni için 32 değeri tanımlanmıştır.

"dim3 dimGrid(B.en / dimBlock.x, 1);" ile grid için blok yapısı tanımlanmaktadır. Sonuç matrisinin en değerine göre değişecektir. Sistem için (7680/32= 240,1) şeklinde olmaktadır.

Transpoze_Paylasimli<<<dimGrid,dimBlock>>>(d_A.elemanlar,d_B.elemanlar,d_A.boy,d_A.en);" ile çekirdek çağrılmaktadır.

```
__global__ void Transpoze_Paylasimli(float *odata, float *idata,int width, int height)
{
    __shared__ float tile[TILE_DIM][TILE_DIM];
    int xIndex = blockIdx.x*TILE_DIM + threadIdx.x;
    int yIndex = blockIdx.y*TILE_DIM + threadIdx.y;
    int index = xIndex + width*yIndex;
    for (int i=0; i<TILE_DIM; i+=BLOCK_SIZE)
    {   tile[threadIdx.y+i][threadIdx.x] =idata[index+i*width]; }
    __syncthreads();
    for (int i=0; i<TILE_DIM; i+=BLOCK_SIZE)
    {   odata[index+i*width] =  tile[threadIdx.y+i][threadIdx.x];}
}
```


Şekil 5. Matris devrik alma çekirdeği

2.8. Ortalama çekirdeği

Ortalama görüntü almak için ortalama çekirdeği kullanılmaktadır. Çekirdeğe parametre olarak giren veri matrisi bir tek boyutlu dizi ile temsil edilmektedir. *n* satır sayısını ve *m* sütun sayısını göstermektedir. Veri matrisindeki her bir satır vektörü için ortalama alarak ortalama görüntü vektörünü hesaplamaktadır. Çekirdek çağırılırken veri matrisindeki her satırı bir blok olacak ve her blokta sütün sayısı kadar eleman olacak şekilde ayarlanmıştır ve program kodları Şekil6. İle gösterilmektedir.

"Kernel_Ortalama<<<A.boy,A.en>>>(d_A.elemanlar,d_B.elemanlar,d_A.boy,d_A.en);" ile çekirdek çağrılmaktadır.

```
__global__ void Kernel_Ortalama(float* in, float* out, int n, int m)
{
    int jdx;//n=boy=satir sayısı, m=en=sutunsayisi
    int idx = blockIdx.x*blockDim.x+threadIdx.x;
    float top = 0;
    if ( idx < n)
    {
        for (jdx=0; jdx < m; ++jdx)
        top += in[idx*m+jdx];
    }
    out[idx] = top/m;
}
```

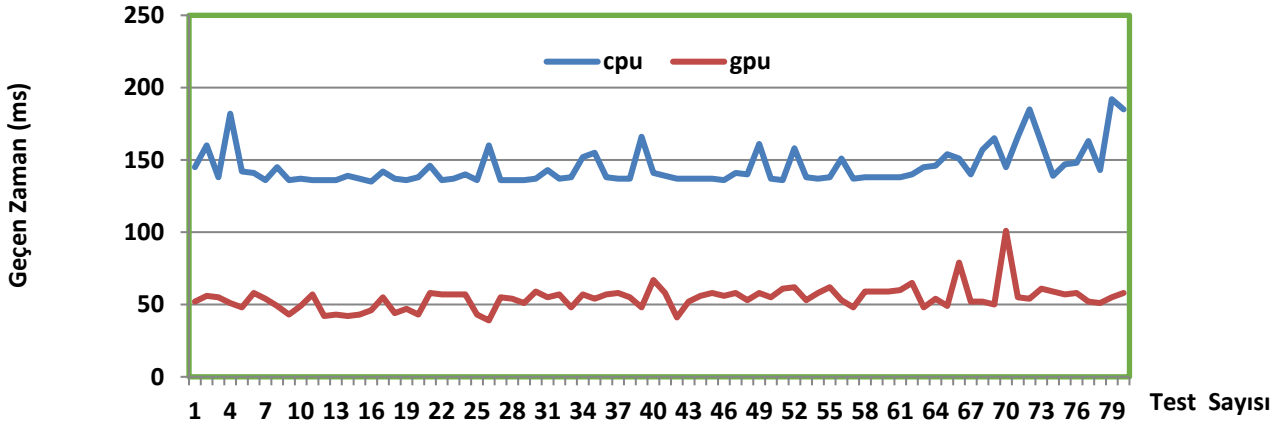
Şekil 6. Ortalama çekirdeği

3. Araştırma Bulguları ve Tartışma

Bu çalışmada 8 kişiden alınan 32 adet resim, eğitim verisi olarak eğitilmiştir ve her bir kişi için 10 kez olmak üzere paralel ve seri olarak yüz tanımlama sistemi uygulanmıştır. Toplam 160 kez test işlemi uygulanarak, her uygulama aşamasında geçen süreler kaydedilmiştir ve performans değerleri belirtilmiştir.

3.1. Paralel–seri çarpım işlemleri karşılaştırması

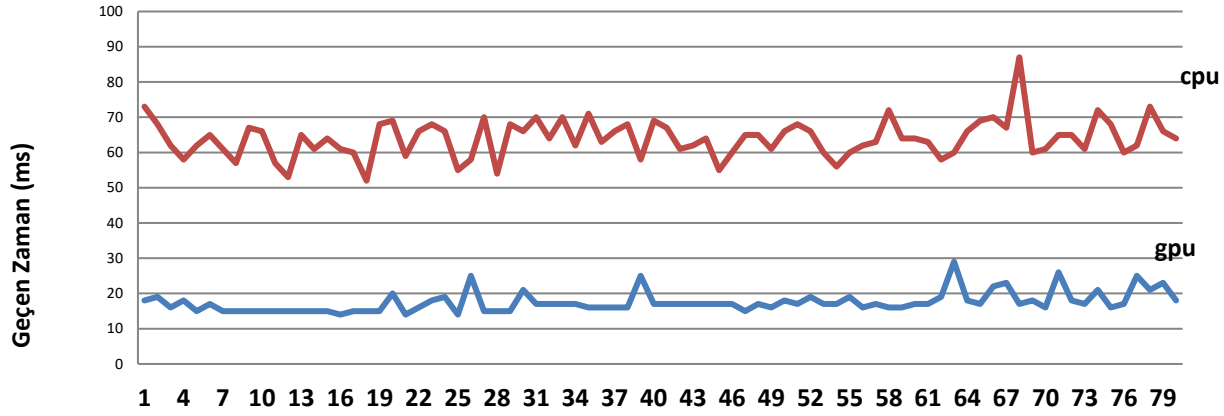
7680x32 ve 32x7680 boyutlarındaki iki matrisin çarpımının 16 blok ile paylaştırılarak paralel işlenmesi ve normal seri işlenmesi sonucu performans karşılaştırması ile Şekil 7 grafiği oluşmaktadır. Normal işlemci üzerinde gerçekleştirilen uygulama sonuçları ile grafik işlemci üzerinde gerçekleştirilen uygulama sonuçları karşılaştırıldığında, grafik işlemcideki işlemlerin normal işlemcideki işlemlere göre ortalama olarak 3-4 kat daha az zaman aldığı görülmektedir. Dolayısıyla grafik işlemci üzerinde paralel olarak programlanan çarpım işleminin hızlandırılmış olduğunu söylemek mümkündür. Grafik işlemci üzerindeki uygulamalar boyunca ortalama 50 ms'lik süre ölçümleri alınırken, normal işlemci üzerindeki uygulamaların çalışma süreçlerinde ortalama 150 ms'lik süre ölçümleri gerçekleşmiştir.



Şekil 7. Çarpım performansları

3.2. Paralel–seri matris devrik alma işlemleri

TBA algoritmasındaki devrik alma işlemleri CUDA C ile paralel programlanarak seri çalışan program uygulaması ile performans ölçümleri karşılaştırılmıştır. Uygulama boyunca devrik alma işlemi hem GPU ortamında hem de CPU ortamında gerçekleştirilmiştir. GPU ortamında paralel olarak programlanan transpoze fonksiyonunda, uygulama esnasında geçen süreler ile CPU ortamında seri olarak programlanan transpoze fonksiyonunda, uygulama esnasında geçen süreler kayıt edilmiştir. CPU ortamında uygulanan transpoze fonksiyonunun uygulama aşamasında ortalama 60 ms'lik zaman alırken, GPU ortamında paralel olarak programlanan fonksiyonlar uygulama aşamasında ortalama 20 ms'lik zaman almaktadır. Bu durumda GPU, CPU'nun 3 katı oranında bir performans artışı sağlamaktadır. 7680x32 boyutunda matris devrik alma işleminin seri ve paralel olarak yapılması aşağıdaki grafik Şekil 8. ile gösterilmiştir.

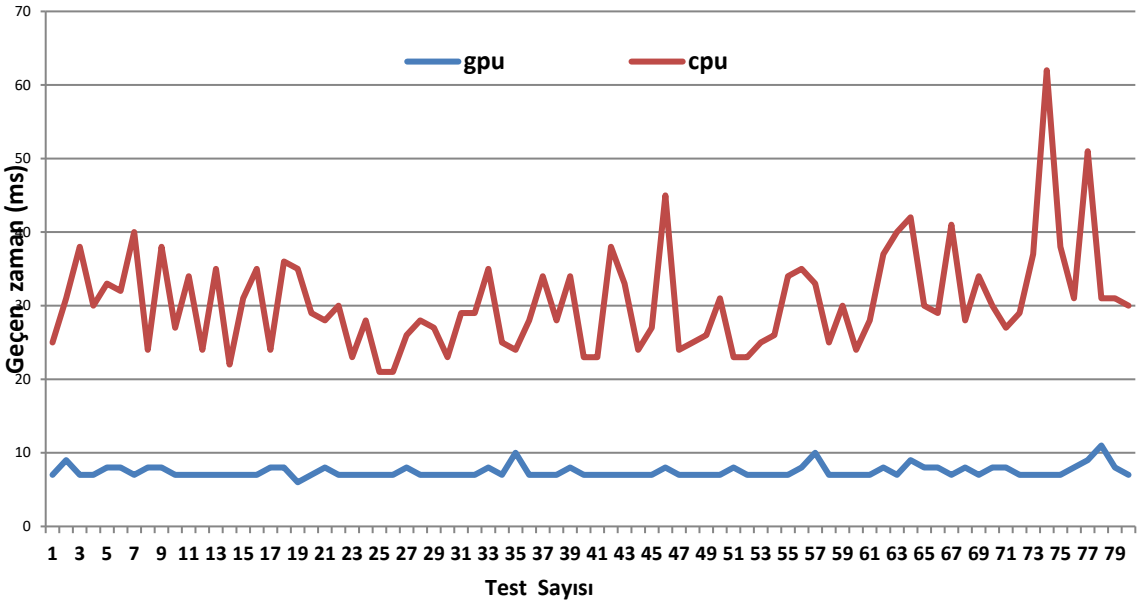


Şekil 8. Matris devrik alma performansları

3.3. Paralel–seri ortalama işlemleri

TBA algoritmasının adımlarından olan ortalama görüntü alma işlemi GPU ve CPU ortamında programlanmıştır. GPU ortamında paralel olarak programlanan ortalama fonksiyonu için ortalama çekirdeği yazılmıştır. CPU ortamında seri olarak programlanan ortalama çekirdeği için uygulama boyunca geçen süreler kaydedilmiştir. Test aşamasında her bir test resmi için uygulama hem seri hem de paralel olarak uygulanmıştır. Aynı resim için seri ortalama fonksiyonu ve paralel ortalama fonksiyonu uygulama aşamasında geçirilen süreler kaydedilmiştir. Böylece ortalama performans

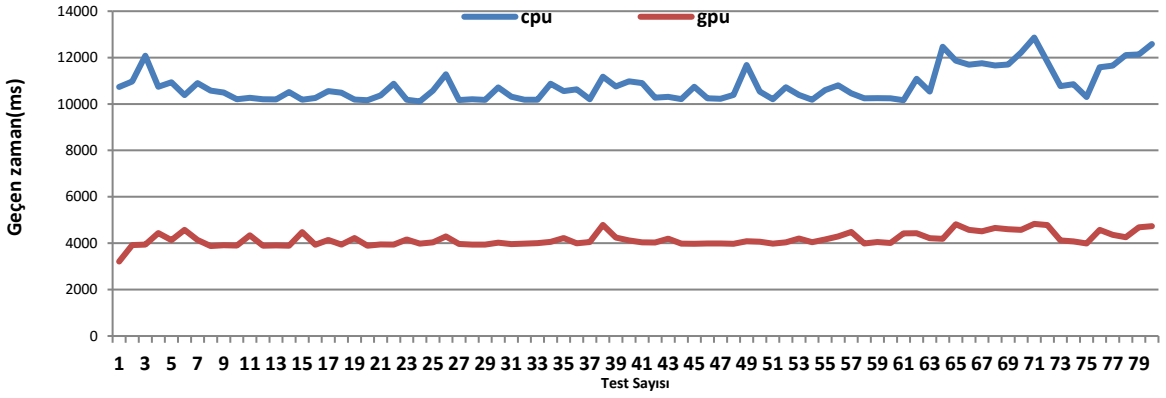
değerleri ortaya çıkmaktadır. Paralel ortalama uygulamasında, ortalama olarak 7-8 ms'lik süre geçerken, seri ortalama uygulamasında ortalama 30-35 ms'lik süreler geçmektedir. Dolayısıyla ortalama alma işleminde 4 katı oranında performans artışı sağlanmıştır. 7680x32 boyutunda matris ortalama işleminin seri ve paralel olarak işlenmesine ait performans grafiği Şekil 9. ile gösterilmektedir.



Şekil 9. Ortalama alma fonksiyonu performans grafiği

3.4. Tahminleme performansı

Test aşamasında 8 kişi için 32 adet resim kullanılmıştır her bir kişi için 10 kez olmak üzere paralel ve seri olarak yüz tanımlama sistemi uygulanmıştır. Toplam 160 kez test işlemi uygulanmıştır. Her test edilen resim için seri ve paralel uygulama adımları için geçen süreler kayıt edilmiştir. Sistem boyunca toplam geçen zaman baz alınmaktadır. Paralel ve seri test işlemleri için geçen süre ölçümleri kıyaslandığında, seri test işlemi için ortalama 110000 ms'lik süre geçmekte iken paralel test işlemi için ortalama 4000 ms'lik süre geçmektedir. Böylelikle GPU ile yapılan paralelleştirme ile tahminleme sisteminde ortalama 3 kat oranında performans artışı sağlanmıştır. Paralel-seri uygulamalara ait performans değerlerini gösteren grafik Şekil 10 ile gösterilmektedir.



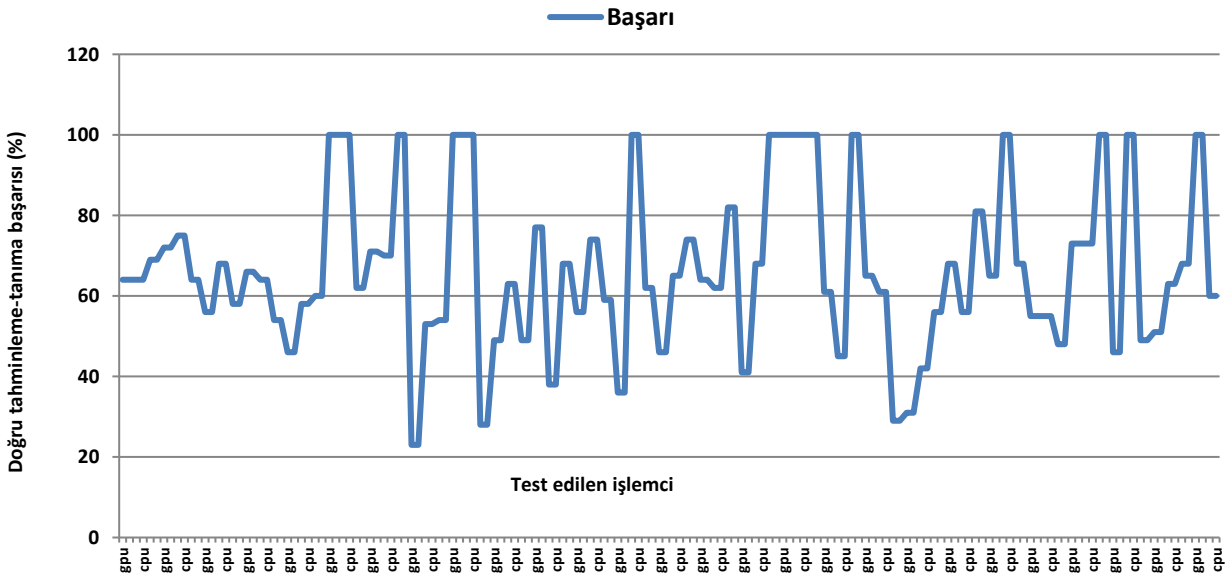
Şekil 10. Sistem paralel-seri tanıma performans grafiği

3.5. Tahminleme başarım durumu

Tahminleme Sürecinin Başarım Durumunu tespit etmek amacıyla 160 test için tüm uygulamalar gözlemlenerek başarım durumu olarak doğru tahminleme oranları tespit edilmiştir. Doğru tahminlemede yardımcı fonksiyon olarak eşik değeri hesaplanmıştır. Sisteme tanımlanmamış kişiler sistemde tanımlanmaya çalışıldığında aradaki öklid uzaklık değerleri genellikle 0.65 ve üzerinde çıkmıştır. Bu yüzden eşik değeri olarak 0.7 alınmıştır. Yani öklid uzaklık hesabında, 0.7'nin altındaki mesafe değerleri doğru resim–doğru tahmin olarak kabul edilmiştir. Bunun için de tahmini sonuç olarak bulunan resim ile gerçek test resminin öklid uzaklığı ayrıca hesaplanarak doğru tahminleme oranları hesaplanmıştır.

Resmin eğitim setindeki herhangi bir resim ile aynı resim olması durumu '1', eğitim setinde olmayan bir resim olması durumu'0' alınması durumunda, başarım durumu '1'-'0' arasında değişen bir değer olacaktır. Benzerlikten farklı olmayı çıkarırsak yani aradaki uzaklık mesafesi çıkarılırsa ne kadar benzediği bulunur. Dolayısıyla:

Benzerlik=resmin kendisi-farklı olması (Öklid uzaklığı) eşitliği ile tanıma ölçütü belirlenmiştir. Resimlerdeki poz, ışık gibi dış etkenler göz önüne alındığında test sonuçlarının %75-%95 oranında doğru tahminleme sağladığı gözlemlenmiştir. Tahminleme başarım değerleri grafiği Şekil 11 ile gösterilmektedir.



Şekil 11. Tahminleme başarı grafiği

4. Sonuç ve Öneriler

Bu çalışmada Nvidia CUDA paralelleştirme platformu ve görüntü işleme üzerine çalışılmıştır. Görüntü işleme olarak yüz tanıma algoritmaları anlatılmış ve bunlardan istatistiksel bir çıkarım yöntemi olan TBA-özyüzler algoritmasının CUDA teknolojisi ile paralel işlenmesi sağlanmaktadır. Paralel işleme için algoritma yeniden programlanmış olup, hem CPU üzerinde hem de GPU üzerinde çalışan metodları geliştirilmiştir. Literatür çalışmalarında %80 doğru tanıma, %20 hatalı tanıma sonuçları görülmüştür[2, 6-8, 10, 11]. Resim verisi olarak genellikle ORL veya FERET resim veri tabanlarından faydalanıldığı görülmüştür, bu çalışmada kendi veri tabanımızın oluşturulmuş olması çalışmaya özgünlük ve farklı bir boyut kazandırmıştır. Ayrıca tek boyutlu

diziler üzerinde çalışılması çalışmayı zenginleştirmiştir. Literatürdeki benzer çalışmalarda, yakın çekirdek içeren işlemcilerde başarım oranları bu çalışmadaki başarım oranlarına çok yakın çıkmasının yanında kepler gibi mimarilerdeki işlemcilerde başarım oranları %400-%500 fark etmektedir. Dolayısıyla donanımsal yeterlilikler performans için önemli bir etkendir. Sonuç olarak performans analizleri yapılarak; tanımlamada önemli performans artışı sağlanmıştır. Yaklaşık olarak 3 katı oranında bir iyileştirme–hızlandırma sağlanmış, ancak donanımsal yetersizliklerin, iyileştirmenin önünde bir engel olmaktadır. Donanımın elverdiği kadarıyla iyileştirme yapmak mümkün olmuştur. Yüz tanıma başarısı %75-95 arasında değişmiştir. Yakalanan test resminin çekildiği ortamın aydınlığı, verilen pozun yönü, durumu vb. etkenin tanımayı etkilediği görülmüştür.

Literatürde yapılan çalışmalarda genellikle hazır kütüphaneler ve fonksiyonlar aracılığıyla yapılan uygulamalar ile karşılaşılmıştır. OpenCV, OpenGL, OpenCL gibi CUDA desteği sunan kütüphaneler içeren hazır bilgisayarlı görü uygulamalarından faydalanıldığı görülmüştür. Bu çalışmada tek boyutlu diziler üzerinde CUDA C diliyle çekirdekler yazılmıştır. Bu yönüyle paralel programlamaya katkı sunulması amaçlanmıştır. Ayrıca Türkiye'de yapılmış çalışmalara bakıldığında CUDA konusunu içeren tez çalışmalarının sayıca azlığı göz önüne alındığında bu çalışmamızın literatüre katkı sunacağı ve CUDA ile paralel programlama alanında çalışma yapacak olan araştırmacılara önemli katkılar sunacağı düşünülmektedir.

[*] Sorumlu yazarın, "CUDA Tabanlı Görüntü İşleme ve Bir Paralel Yüz Tanıma Uygulaması" isimli yüksek lisans tezinden üretilmiş yayındır.

Yazarların Katkısı

Yazarlar makaleye eşit oranda katkı sağlamıştır.

Çıkar Çatışması Beyanı

Yazarlar arasında herhangi bir çıkar çatışması bulunmamaktadır.

KAYNAKLAR

[1] Jain, A., Bolle, R., Pankanti, S. (2006). Biometrics: Personal Identification in Networked Society (Vol. 479). Springer Science & Business Media, US, 1-411.

[2] Erdoğan, A.Y. (2010). Yüz Tanımada Özyüz ve Fisher Yüz Algoritmalarının İncelenmesi. Yüksek Lisans Tezi, Anakara Üniversitesi, Fen Bilimleri Enstitüsü, Ankara, 1-136.

[3] Jain A.K., Kumar A. (2012). Biometric Recognition: An Overview. In: Mordini E., Tzovaras D. (eds) Second Generation Biometrics: The Ethical, Legal and Social Context, The International Library of Ethics, Law and Technology, Springer, Dordrecht,11:49-79.

[4] Sabhanayagam, T., Venkatesan, V.P., and Senthamaraikannan, K. (2018). A Comprehensive Survey on Various Biometric Systems. International Journal of Applied Engineering Research, 13 (5), pp. 2276-2297.

[5] Origin, A. (2005). UKPS Biometric Enrolment Trial. Biometric Technology Today, 13(7):6-7.

[6] Bhele, S.G., and Mankar, V. (2012). A Review Paper on Face Recognition Techniques. International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), 1(8): 339-346.

[7] Hassaballah, M., and Aly, S. (2015). Face Recognition: Challenges, Achievements and Future Directions. IET Computer Vision, 9(4): 614-626.

[8] Singh, S., and Prasad, S. (2018). Techniques and Challenges of Face Recognition: A Critical Review. Procedia Computer Science, 143: 536-543.

[9] Kirby, M., and Sirovich, L. (1990). Application of the Karhunen-Loeve Procedure for the Characterization of Human Faces. IEEE Transactions on Pattern Analysis and Machine Intelligence, 12 (1): 103-108.

[10] Imran, M., Miah, M., Rahman, H., Bhowmik, A., and Karmaker, D. (2015). Face Recognition Using Eigenfaces. International Journal of Computer Applications, 118 (5).

[11] Abbas, E. I. (2015). Effect of Eigenfaces Level On The Face Recognition Rate Using Principal Component Analysis. Enginnering and Technology Journal, *33*(3).

[12] Turk, M., and Pentland, A. (1991). Eigenfaces for Recognition. Journal of Cognitive Neuroscience, 3 (1): 71-86.

[13] Anonim (2015). Matrix Multiplication. https://en.wikipedia.org/wiki/Matrix_multiplication. Erişim Tarihi:(15.07.2015).