CuEEG: EEG Verilerinin Hızlı İşlenmesi için GPU Tabanlı Bir Yaklaşım
CuEEG: A GPU-Based Approach for Fast Processing of EEG Data

Musa PEKER[1], Baha ŞEN[2]

[1] Bilgisayar Mühendisliği
Karabük Üniversitesi
pekermusa@gmail.com

[2] Bilgisayar Mühendisliği
Yıldırım Beyazıt Üniversitesi
bsen@ybu.edu.tr

Özet

Bu çalışmada EEG verilerinin sınıflandırılmasına yönelik yeni bir yaklaşım sunulmaktadır. Bu yaklaşımın dikkate aldığı kriter hız parametresidir. Şimdiye dek gerek epilepsi teşhisi gerekse EEG verilerinin sınıflandırılmasına yönelik yapılmış hemen hemen bütün çalışmalar sınıflandırma doğruluğunu iyileştirmeye yönelik olarak gerçekleştirilmiştir. Bu çalışmada özellikle gerçek zamanlı uygulamalarda bir vazgeçilmez olan hız parametresi dikkate alınmakta ve EEG verilerinin hızlı bir şekilde sınıflandırılması için yeni bir yaklaşım sunulmaktadır. Bunun için özellikle son yıllarda popüleritesi gittikçe artan grafik işlemciler üzerinde programlama yapmamızı sağlayan CUDA programlamadan yararlanılmıştır. Grafik işlemcilerden sınıflandırma aşamasında yararlanılmıştır. Sınıflandırma algoritması olarak EEG verilerinin sınıflandırılmasında yüksek doğruluk oranı veren geri yayılımlı sinir ağı tercih edilmiştir. Bu çalışmada epileptik nöbet tespiti için kullanılan bir veri tabanı üzerinde deneyler gerçekleştirilmiştir. Sonuçlar incelendiğinde CUDA programlamanın sonuçlar üzerinde olumlu etkilerinin olduğu görülmüştür. Bu şekilde sadece EEG verilerine yönelik değil diğer biyomedikal işaretlerin sınıflandırılmasında da grafik işlemciler kullanılarak gerçek zamanlı uygulamalara yönelik hızlı bir sistemin altyapısı oluşturulabilmektedir.

Abstract

The study presents a new approach in the classification of EEG data. The basic criterion in this approach is the parameter of speed. Almost all up to date studies both for the diagnosis of epilepsy and for the classification of EEG data aim to improve classification accuracy. This study focuses on the speed parameter which is an indispensable part in real time applications and presents a new approach to classify EEG data in a faster manner. CUDA program has been utilized that makes it possible to make programs on Graphic Processors which is highly popular especially in the last years. Graphic processors have been used in the classification phase. As classification algorithm, back propagation neural network with a higher accuracy rate has been preferred in the classification of EEG data. The study utilized experiments on a data base used for diagnosis of epileptic attacks. Data show that CUDA programming has positive effects on the results. It is believed that this approach can be used for the building the infrastructure for a speedy system for real time applications by using graphic processors for the classification of not only EEG data but also of other biometric signals.

1. Giriş

GPU' lar (Graphics Processing Unit – Grafik İşleme Birimi), esas olarak grafiksel işlemlerde CPU (Central Processing Unit – Merkezi İşlemci Birimi) üzerindeki yükü azaltmak ve daha iyi performans sağlamak için özelleşmiş yapılardır. Günümüzde, GPU'lar da CPU'lar gibi çok çekirdekli yapılara sahip hale gelmiştir. Paralel çok çekirdekli, genel matematik işlemlerini grafik işleme birimi üzerinde yapmaya olanak sağlayan GPU' lar, grafik kartlarının günümüzde özellikle yüksek performans gerektiren sinyal işleme algoritmalarının gerçeklemesinde önemli bir yere gelmesini sağlamıştır [1].

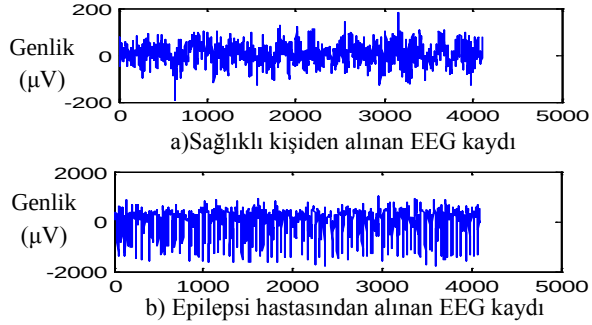
Sinyal işleme yazılımlarında kullanılan algoritmaların karmaşıklığı arttıkça ve verilerin boyutları büyüdükçe büyük verileri kısa sürelerde işlemeye yönelik bir arayış ortaya çıkmaya başlamıştır. GPU' ların kabiliyetlerini arttırmaya yönelik olarak 2007 yılında serbest bir platform olan CUDA' nın NVIDIA tarafından piyasaya sürülmesinin ardından, bu alana ilgi artmış, grafik kartları üzerinde de genel amaçlı matematiksel işlemlerin yapılabileceği görülmüş ve ilerleyen süreçlerde bu alanda çalışmalar hızlanmıştır [1].

Bu çalışmada CUDA Programlama ile EEG verilerinin daha hızlı sınıflandırılması amaçlanmıştır. Sınıflandırma aşamasında, genellikle EEG verilerinin sınıflandırılmasında yüksek doğruluk değerleri veren geri yayılımlı sinir ağı tercih edilmiştir. Yapay sinir ağlarının eğitimi, optimal performansa ulaşmak için gerekli ağırlık güncellemeleri ve yüksek iterasyon sayısından dolayı zaman alıcı olmaktadır [2]. Hesaplama maliyetini azaltmak veya yakınsama hızını yükseltmek için çaba gösterilen çalışmalar mevcuttur [3, 4]. CUDA üzerinde geri yayılımlı sinir ağı uygulaması gerçekleştirilirken BLAS kütüphanesinin bir CUDA uygulaması olan CUBLAS kullanılarak süreç basitleştirilmiştir. Bununla beraber CUBLAS gerekli bütün kütüphanelere sahip olmadığı için çekirdek (kernel) fonksiyon kullanımına ihtiyaç duyulmuştur.

Uygulama epilepsi ve sağlıklı insanların EEG işaretlerinden oluşan bir veri tabanı üzerinde gerçekleştirilmiştir. NVIDIA grafik kartı üzerinde tek gizli katman ile gerçekleştirilen uygulama, performans açısından CPU' ya göre daha iyi bir performans vermiştir.

2. Veri Seti

Bu çalışmada Bonn Üniversitesi epileptoloji kliniğinde hazırlanan EEG veri tabanı kullanılmıştır [5]. Bu veri tabanında A, B, C, D, E olmak üzere, 5 farklı EEG veri kümesi bulunmaktadır. Bu çalışmada A ve E veri kümesi kullanılmıştır. A kümesi, sağlıklı kişilerden, gözler açık biçimde, yüzey elektrotları ile kaydedilmiş EEG verisini içerir. E kümesi ise hastalıklı kişilerden, epilepsi krizi anında, girişimsel olarak kaydedilmiş EEG verisini içerir. Sinyaller 12 bit analog-sayısal dönüştürücü ile dönüştürüldükten sonra bilgisayar ortamına 173.61 Hz örnekleme frekansı ile aktarılmıştır. Epileptik özellikler kendini 30-40 Hz altındaki frekans bantlarında gösterdiğinden spektral aralığı 0.5-85 Hz olan sinyallere 0.53-40 Hz bant geçiren filtre uygulanmıştır. Örnek EEG görüntüleri Şekil 1' de görülmektedir.



Şekil 1: A ve E veri setlerinden alınan örnekler

3. Öznitelik Çıkarma

Bu aşamada EEG sinyallerinden anlamlı bilginin keşfi için 9 öznitelik çıkarma algoritması kullanılmıştır. 4096 noktadan oluşan her EEG sinyali bu öznitelik çıkarma aşamasından geçtikten sonra 9 nokta ile temsil edilmiştir. Bu öznitelikler istatiksel ve enerji tabanlı özniteliklerdir. Bu çalışmada tercih edilen öznitelik algoritmaları sırasıyla minimum değer, maksimum değer, standart sapma, aritmetik ortalama, varyans, ortanca, sıfır geçiş sayısı, ortalama teager enerjisi ve ortalama enerji değerleridir.

4. Sınıflandırma

Bu bölümde ilk aşamada sınıflandırma aşamasında kullanılan geri yayılım sinir ağı hakkında kısa bir bilgi verilmiştir. İkinci aşamada CUDA programlamadan bahsedilecektir.

4.1. İleri Beslemeli Geri Yayılımlı Sinir Ağı

Danışmanlı öğrenen, ileri beslemeli geri yayılım ağları, adını hatayı yayma biçiminden almıştır. Hata azaltma işlemini hatayı tüm ağa yayarak gerçekleştirdiği için bu ağa "Hatayı Geriye Yayma" da denmektedir. Bu algoritmada elde edilen çıktı ile olması gereken çıktı arasındaki fark yani hata değeri, tüm ağırlıklara yansıtılarak dereceli olarak minimum düzeye indirilmeye çalışılır [6]. Geri yayılım algoritmasında eğitime rastgele bir ağırlık kümesi ile başlanır, Q katmanlı ileri beslemeli bir ağ için geri yayılım algoritması [7];

q = 1, 2, 3, ... , Q katman numarası, X_i^q: q' inci katmandaki i biriminin girdisi, y_i^q: q' inci katmandaki i biriminin girdisi, w_{ij}^q: *(q-1)*'inci katmandaki i birimini, q' ncu katmandaki j birimine bağlayan ağırlık olmak üzere aşağıdaki adımlarla gerçekleştirilir:

1. Adım: w' ye reel değerli küçük rastgele sayıları başlangıç değeri olarak atanır.

2. Adım: Rastgele bir (giriş-hedef) çalışma modeli seçilir ve q katmanındaki her bir j birimi için ileri yönde çıktı değerleri hesaplanır. Böylece çıkış,

$$y_i^q = f\left(\sum_i y_i^{q-1} w_{ij}^q\right) \quad (1)$$

3. Adım: Çıkış birimleri için hata terimleri (δ) hesaplanır:

$$\delta_i^Q = \left(y_i^q - y_i^p\right) f'\left(X_i^Q\right) \quad (2)$$

4. Adım: q = Q, Q −1, ... , 2 katmanlarındaki tüm i birimleri için geriye yayılımla gizli katman birimleri için hata terimleri hesaplanır.

$$\delta_i^{q-1} = f'\left(X_i^{q-1}\right) \sum_i \delta_i^q w_{ij}^q \quad (3)$$

5. Adım: Ağırlık değerleri aşağıdaki denklemlere göre güncellenir.

$$w_{ij}^{yeni} = w_{ij}^{eski} + \Delta w_{ij}^q \quad (4)$$

Öğrenme katsayısı η kullanılarak, öğrenme aşamasında ağırlıkta yapılan değişim Δw_{ij}^q, aşağıdaki eşitliğe göre olmalıdır.

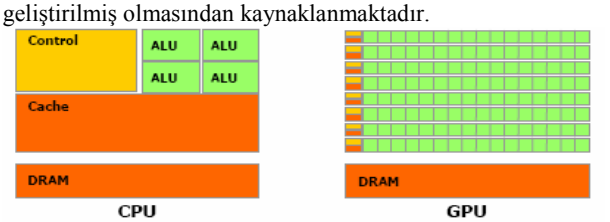
$$\Delta w_{ij}^q = \eta \delta_i^q y_i^{q-1} \quad (5)$$

6. Adım: 2. adıma dönüp, toplam hata kabul edilebilir bir düzeye gelene kadar her bir p modeli için işlemler tekrarlanır. Bu çalışmada Q=3 için işlemler yapılmıştır.

4.2. Cuda Programlama

CUDA, GPU üzerindeki yüzlerce çekirdeği kullanarak genel amaçlı matematik işlemleri yapmaya olanak sağlayan bir GPU mimarisi, yazılımı ve programlama platformudur [1].

CUDA' nın başarılı sonuçlar üretmesinin altında yatan neden, CUDA mimarisinin, grafik işleme gibi işlem yoğunluğu olan ve yüksek derecelerde paralellik gerektiren işlemler için geliştirilmiş olmasından kaynaklanmaktadır.

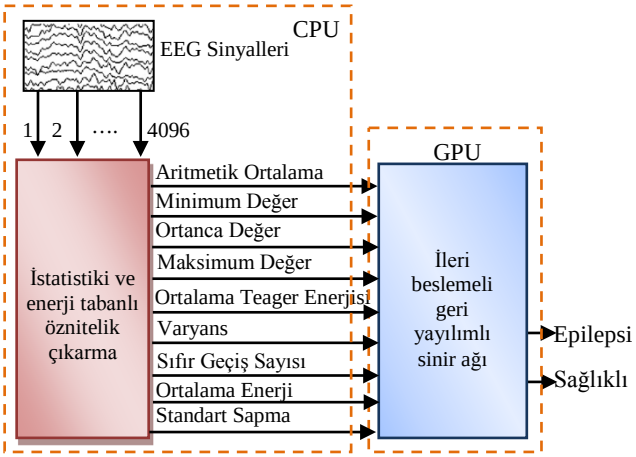


Sekil 2: GPU veri işleme için daha fazla transistor tahsis eder.

CUDA programlama ile ilgili geniş bir bilgi [8] numaralı kaynaktan alınabilir.

5. Uygulama ve Değerlendirme

Bu çalışmada EEG verilerinin sınıflandırılması için önerilen yapıya ait blok gösterim Şekil 3' de verilmektedir. Bu yapıda verilerin özniteliklerinin çıkarılabilmesi için hem A veri seti (sağlıklı), hem de E veri seti (epileptik aktivite durumu) için 100*4096 boyutunda iki ayrı vektör matris oluşturulmuştur. Bu vektörlerden, 100*4096 boyutundaki A ve E veri setinin her bir sütunu için 9 adet öznitelik parametresi elde edilerek geri yayılımlı sinir ağı ile sınıflandırılmıştır.



Şekil 3: Epileptik atak tespiti için önerilen yöntem

5.1. CUDA Tabanlı Sinir Ağının Gerçekleştirimi

Sinir ağı uygulaması CUDA tarafından uygulanan bir seri matris çarpımı ve aktivasyon fonksiyonu uygulaması olarak gerçekleştirilebilir. CUDA blok başına paylaşılan hafızayı (shared memory) kullanarak matris çarpımını etkin şekilde işleyebilmektedir. Uygulamalarda paylaşımlı bellek verimi önemli oranda artıracaktır. Örnek verilirse; GPU genel işlemlerinde global hafızaya erişim için yaklaşık olarak 400-600 döngü gerekliyse, CUDA' nın hafıza ortamında paylaşılan hafızaya erişimi için sadece 4 döngü yeterli olmaktadır [9]. Bu nedenle CUDA' nın paylaşımlı hafızası işlemlerin etkin yürütülmesi için yardımcı olmaktadır. Ayrıca matrisin eleman sayısına eşit sayıda dizin (thread) sağlayarak ve daha sonra da her bir dizinde (thread) işlemleri bağımsız olarak hesaplayarak aktivasyon fonksiyonlar da paralel olarak uygulanabilir [9].

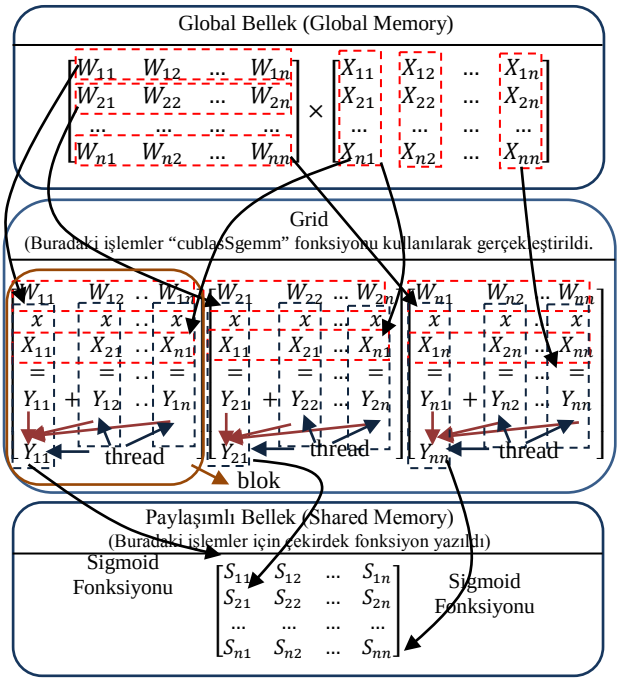
Şekil 4' de CUDA üzerinde gerçekleştirilen sinir ağının ileriye doğru hesaplama adımlarından bir kısmının matris işlemleri sunulmaktadır. Şekil incelendiğinde, sinir ağlarının paralel işlemeye müsait bir yapısının olduğu görülecektir.

Geri yayılımlı sinir ağı algoritmasının paralel uygulaması, verilerin GPU' da matris ve vektör işlemlerinin uygulanmasından ibarettir [10]. Bu çalışmada iki tür paralel işlem mevcuttur: vektör-matris işlemleri ve aritmetik işlemler. Vektör-matris işlemlerine örnek olarak giriş değerleri ile giriş ve gizli katman değerleri arasındaki ağırlık değerlerinin çarpımı verilebilir. Aritmetik işlemlere de örnek olarak sigmoid fonksiyonu ve ağırlık güncellemeleri verilebilir. Bu çalışmada ilk aşamada yani matris ve vektör işlemleri için CUBLAS kütüphanesi tercih edilmiştir. CUBLAS optimum paralelliğe uygun yapılandırılmış bir seri algoritma sunduğundan, bloklar ve dizinlerin tanımlanması problem teşkil etmemektedir [10]. İkinci aşamada yani aritmetik işlemler için ise çekirdek (kernel) tanımlanarak paralelliğin gerçekleştirilmesi sağlanmıştır. Burada kernel, GPU üzerinde çalışan fonksiyona verilen isimdir. Bu iki aşama ile ilgili bazı örnekler aşağıda sunulmaktadır.

Matris çarpım işlemlerinde CUBLAS fonksiyonlarından "cublasSgemm" fonksiyonu kullanılmıştır. Örnek bir kullanım aşağıda sunulmaktadır.

'cublasSgemm (char transa, char transb, int m, int n,int k, float alpha, const float *A, int lda,const float *B, int ldb, float beta,float *C, int ldc)'

A, B ve C matris değerleridir. Alpha ve beta ise skaler değerlerdir.



Şekil 4: CUDA kullanarak geri yayılımlı sinir ağının matris çarpım işlemleri (ileri hesaplama adımlarından bir kesit)[9]

Bu fonksiyona göre A matrisi *mxk*, B matrisi *kxn* ve C matrisi de *mxn* boyutundadır. cublasSgemm fonksiyonu C = (alpha x A x B) + (beta x C) matris işlemini hesaplar. Alpha=1 ve beta=0 olarak alındığında işlem C=AxB' ye indirgenir. Lda, ldb ve ldc ise sabit değerlerdir. Fonksiyondaki trans ifadesi matris transpoze işlemleri için kullanılmaktadır. Genellikle 'n' yapısı tercih edilir. Kullanımı aşağıda belirtilmiştir.

' Eğer trans*a*='N' veya 'n' , A=A.,
Eğer trans*a*='T','t', 'C' or 'c' ise $A=A^T$'dir.
trans*b* ifadesi ise aynı amaçla B matrisi için kullanılır.'

CUBLAS kullanırken dikkat edilmesi gereken MATLAB' da olduğu gibi matrislerde verinin "column major (veriler sütun öncelikli olarak bellekte tutulmaktadır)" şeklinde işlendiği ve tutulduğudur. Belirtilen cublasSgemm fonksiyonunun dışında bu çalışmada GPU bellek ayrımı için cublasAlloc, ayrılan bellek alanını tekrar bırakmak için cublasFree, CPU' dan GPU' ya vektör aktarımı için CublasSetVector, GPU'dan CPU' ya vektör aktarımı için CublasGetVector, CPU' dan GPU' ya matris aktarımı için CublasSetMatrix, GPU' dan CPU' ya matris aktarımı için CublasGetMatrix fonksiyonlarından yararlanılmıştır.

Aritmetik işlemlerin paralelliği için kernel tanımlanmıştır. Aktivasyon fonksiyonu, hata fonksiyonu ve ağırlık güncellemeleri için çekirdek (kernel) fonksiyonlar geliştirilmiştir. Sigmoid aktivasyon fonksiyonu ile ilgili hesaplamalarda işlemlerin GPU üzerinde gerçekleştirilmesini sağlayan örnek bir kernel Şekil 5'de sunulmuştur.

```
__global__ void sigmoid( float *cikis1, int maksimum) {
__shared__ float u_shared[32]; //Paylaşımlı bellek ayırma
 int sutun = blockIdx.x * blockDim.x + threadIdx.x; //GPU dizininin
(thread) işleyeceği adres bilgisi
if (sutun<maximum){
u_shared[sutun] = d_an[sutun]; //Paylaşık hafızaya yazma
u_shared[sutun]  = (1/(1+__expf(-1*u_shared[sutun]))); //sigmoid
fonksiyon değerinin hesaplanması
__syncthreads();  } // bloktaki tüm threadlerin bu noktaya erişmesini
bekletir (Senkronizasyon amaçlı kullanılır).
d_an[sutun]=u_shared[sutun]; //Global hafızaya yazma }
```

Şekil 5: CUDA donanımı üzerinde çalışacak sigmoid fonksiyonu için yazılan kernel fonksiyonu

Bu çekirdek fonksiyonunun çağrımı aşağıdaki gibi gerçekleştirilmiştir.

sigmoid<<< Grid Sayısı, Blok Sayısı >>>(… , …);

GPU üzerinde gerçekleştirilen sinir ağı için çeşitli optimizasyonlar yapılmıştır. Gerçekleştirilen optimizasyon aşamalarından önemli olanlar aşağıda listelenmiştir.

- Paralel yapılabilen kod parçalarının CUDA üzerinde, seri olması gereken işlemlerin CPU üzerinde çalıştırılmasına dikkat edilmiştir.
- CUDA ile işlenecek verilerin bir kerede GPU üzerine alınması ve GPU üzerinde tüm işlemler yapıldıktan sonra CPU'ya alınmasına dikkat edilmiştir. (Şekil 5' de verilen örnekte bu kurala uygun bir örnek görülmektedir.)
- Paylaşımlı bellek kullanımı, hesaplamalar esnasında bellek erişimini çokça kullanan fonksiyonlarda önemli derecede performans artışı sağlamıştır.

5.2. Uygulama Sonuçları

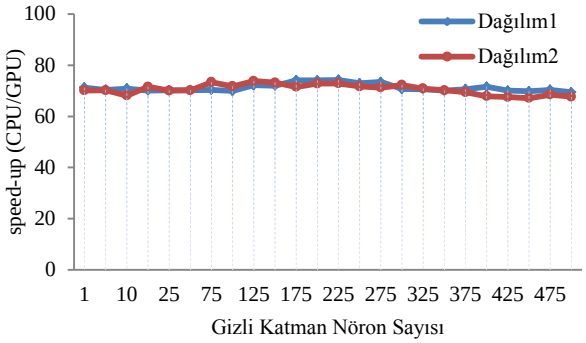
GPU performansı için deneyler NVIDIA GeForce GT 525M grafik kartı üzerinde yapılmıştır. Bu grafik kartında 96 Cuda çekirdeği mevcuttur. İşlemci hızı 1200 Mhz' dir. CPU performansını tespit etmek için aynı bilgisayarın işlemcisi tercih edilmiştir. Bu işlemci Intel (R) Core™ i7-2670QM CPU @ 2.20 GHz özelliklerine sahiptir. Bilgisayarın belleği 8 GB kapasiteye sahiptir. Karşılaştırmaların adil olması için yazılımlar aynı programlama dilinde kodlanmıştır. Ayrıca kodlamalarda tek duyarlıklı (single precision) gösterim tercih edilmiştir. Geliştirilen yazılımlar Visual C++ 2010 programlama dili ile hazırlanmıştır. Uygulamada kullanılan veri seti 2 şekilde kullanılmıştır. İlkinde veri setinin %50 si eğitim için, %50 si test için ayrılmıştır. İkinci şekilde veri setinin %70' i eğitim için, %30'u test için ayrılmıştır. Bu ayrım rastgele bir şekilde yapılmaktadır. Deneysel sonuçlarda kolaylık olması açısından birinci dağılım için Dağılım1, ikinci dağılım içinse Dağılım2 denilecektir.

5.2.1. Hız Performansı

Paralel programlamanın etkisini görmek için 2 aşamada deneyler yapılmıştır. İlk aşamada sonlandırma kriteri olarak eşit iterasyon değeri tercih edilmiştir. Bu aşamada gizli katman nöron sayısının artması durumunda GPU ve CPU' nun sonuca ulaşma zamanları karşılaştırılmıştır. Gizli katman nöron sayısı arttıkça ağırlık matrisleri de büyüyecek ve problem daha karmaşık bir hale gelecektir. İkinci aşamada ise ağ mimarisi sabit tutulup iterasyon değerleri değiştirilmiştir. Ağ mimarisi 9-20-1 olacak şekilde sabitlenmiştir. Burada 9 sayısı giriş katmanındaki elemanların sayısını, 20 gizli katman nöron sayısını ve 2 ise çıkış katmanındaki elemanların sayısını vermektedir.

Algoritmaların çalıştırılmaları sonucu elde edilen sonuçlar Şekil 6 ve Şekil 7' de görülmektedir. Sonuçlar milisaniye (ms) cinsinden hesaplanmıştır. Hesaplama süreleri Şekil 3' de görüldüğü gibi CPU ortamında elde edilen 9 öznitelik değerinin GPU' ya taşınması, bellek ayrımı, GPU-CPU arası veri aktarımları ve ileri beslemeli sinir ağının uygulanması sonucunda elde edilen sürelerin toplamından oluşmaktadır. Her deney 10 defa tekrar edilmiş ve bulunan değerlerin ortalaması kullanılmıştır.

Şekil 6-7 incelendiğinde CUDA Programlamanın sonuçlara etkisinin yüksek olduğu görülmektedir.

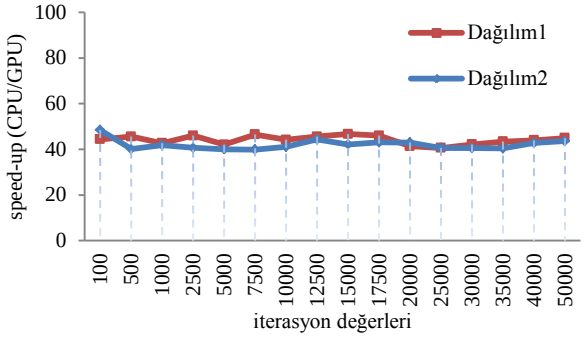


Şekil 6: Gizli katman nöron sayısına göre değişen speed-up değerleri

Şekil 6'da Dağılım1 için ortalama speed-up değerinin 71.21, Dağılım2 için ortalama speed-up değerinin 70.56 çıktığı görülmektedir. Bu durumda gizli katman nöron sayısının etkisi göz önüne alındığında CUDA Programlama ile ortalama 70x civarında bir başarının sağlandığından bahsedilebilir. Grafikteki bazı değerler aşağıda tablo halinde sunulmuştur. Çizelge 1' de görüldüğü gibi paralel programlamanın etkisiyle yaklaşık 48 saniye süren bir işlemin 0.69 saniyeye düşürüldüğü görülmektedir.

Çizelge 1: Gizli katman nöron sayılarına göre sınıflandırma süreleri(msn)

Gizli Katman Nöron Sayısı	Dağılım1		Dağılım2	
	GPU	CPU	GPU	CPU
1	225.1	16034.5	250.1	17529.1
100	335.4	23450.4	342.4	24559.4
300	445.7	31500.7	450.5	32555.3
500	690.1	47900.3	722.4	48895.3



Şekil 7: İterasyon sayısına göre değişen speed-up değerleri

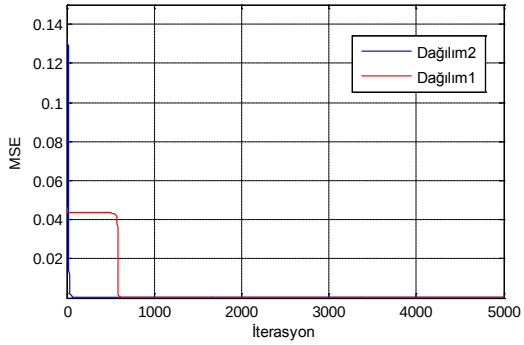
Şekil 7' de Dağılım1 için ortalama speed-up değeri 42.01, Dağılım2 için ortalama speed-up değeri 44.61 çıkmıştır. Bu durumda iterayon değerinin etkisi göz önüne alındığında CUDA Programlama ile ortalama 43x civarında bir başarının sağlandığından bahsedilebilir. Grafikteki bazı değerler aşağıda tablo halinde sunulmuştur. Çizelge 2' de görüldüğü gibi paralel programlamanın etkisiyle yaklaşık 395 saniye süren bir işlemin yaklaşık 9 saniyeye indiği görülmektedir.

Çizelge 2: İterasyon değerlerine göre sınıflandırma süreleri(msn)

İterasyon Değeri	Dağılım1		Dağılım2	
	GPU	CPU	GPU	CPU
1000	312.4	13338.3	322.6	13500.6
10000	2400.1	106020.4	2600.1	106786.5
20000	4911.8	203008.1	5250.5	225655.4
50000	8807.5	395459.4	11450.08	498788.5

5.2.2. Başarı Performansı

Çalışmada en iyi sonucu veren uygun parametre değerlerinin tespiti deneme-yanılma metodu ile bulunmuştur. Buna göre öğrenme katsayısı 0.6, momentum katsayısı 0.5, gizli katman nöron sayısı 5 olarak belirlenmiştir. İterasyon değeri 5000 olarak sabitlenmiştir. Bu değerler kullanılarak eğitilen ağın İterasyon-MSE (Ortalama Karesel Hata) grafiği Şekil 8' de sunulmaktadır. Grafik incelendiğinde veri setinin daha büyük bir oranını eğitim için kullanan Dağılım2' nin daha başarılı sonuçlar verdiği görülmektedir.



Şekil 8: İterasyon-MSE grafiği

Başarı performansı için ayrıca istatistiki analiz değerleri hesaplanmıştır. Bulunan değerler 10 farklı deneme sonucu elde edilen değerlerin ortalamasıdır. Performans değerlendirme kriterlerine göre elde edilen sonuçlar Çizelge 3' de sunulmaktadır.

Çizelge 3: Performans değerlendirme kriterlerine göre elde edilen sonuçlar

Performans Kriteri	**Dağılım1**	**Dağılım2**
Doğruluk oranı (DO)	% 96.8	% 98.5
Duyarlılık	% 96.9	% 98.02
Özgünlük	% 96.1	% 98.9
MSE	$2.17x10^{-6}$	$9.49x10^{-7}$

Bu çalışmada elde edilen doğruluk oranları ile aynı veri seti üzerinde daha önce yapılmış çalışmalar karşılaştırılmıştır. Doğruluk değerleri dikkate alınarak yapılan karşılaştırmalı analiz Çizelge 4' de sunulmaktadır.

Çizelge 4: Literatürdeki çalışmalar ile doğruluk değerlerinin kıyaslanması

Ref. No	**Metot**	**DO**
[11]	Dalgacık Dönüşümü-Bağıl Dalgacık Enerjisi-Sinir Ağları (YSA)	%95.0
[12]	Lineer olmayan ön işleme filtreleri-YSA	%97.2
[13]	İstatiksel öznitelikler-Diskriminant Analizi	%96.9
[14]	Zaman ve frekans tabanlı öznitelikler-Geri beslemeli sinir ağları	%99.6
[15]	Entropi tabanlı öznitelikler-Uyarlamalı sinirsel bulanık denetim sistemi	%92.2
[16]	Hızlı Fourier dönüşümü-Karar ağaçları	%98.7

Sonuçlar incelendiğinde önerilen metodun EEG verilerini hızlı bir şekilde ve yüksek doğruluk değerlerinde sınıflandırdığı görülmektedir. Önerilen metodun epilepsi hastalığına yönelik olarak da nihai karar açısından hekimlere faydalı olacağı öngörülmektedir.

6. Sonuçlar

Bu çalışma ile CUDA teknolojisi sayesinde EEG sinyallerinin daha kısa zamanda sınıflandırıldığı görülmüştür. Bu çalışmada kullanılan grafik kartı ortalama bir performansa sahiptir. Piyasada 400-1000 arası çekirdek sayısına sahip Fermi, Tesla gibi grafik kartları da mevcuttur. Yine karşılaştırma amaçlı kullanılan CPU ise şu anda piyasadaki en iyi işlemciler arasında gözükmektedir. Ortalama performansa sahip bir ekran kartı ile çok iyi performansa sahip bir işlemcinin karşılaştırıldığını düşündüğümüzde CUDA programlamanın önemi daha ön plana çıkacaktır. Ayrıca çalışmada önerilen metodun sınıflandırılma doğruluğu ve diğer istatistiki sonuçları da elde edilmiştir. Başarı performansı yönüyle de yüksek doğruluk değerleri elde edilmiştir. Bu yönüyle de önerilen metodun hem hız hem de performans yönüyle etkili bir metot olduğu söylenebilir.

İleriki çalışmalarımızda ön işleme aşamasını da GPU üzerinde gerçekleştirerek, tamamen GPU üzerinde çalışan bir sistem hedeflemekteyiz. Bu tarz çalışmalarla sadece Biyomedikal Sinyal İşleme için değil, gerçek zamanlı çalıştırılması hedeflenen birçok çalışmanın hızlı bir şekilde çalıştırılıp, hızlı ve başarılı sonuçların elde edilmesi sağlanabilir.

7. Kaynaklar

[1] Sağel, B., Ceylan, S., Akpulat, M.C., Türken, H. ve Erdoğdu, S., "CUDA Altyapısı Kullanılarak Sinyal İsleme Yazılımı Optimizasyonu", *5. UYMS*, 159-162.

[2] Şen, B. ve Peker, M., "Novel Approaches for Automated Epileptic Diagnosis using FCBF Feature Selection and Classification Algorithms", *Turkish Journal of Electrical Engineering & Computer Sciences*, 10.3906/elk-1203-9, 2012.

[3] Cristea, A. ve Okamoto, T., "A Parallelization Method for Neural Networks with Weak Connection Design", *International Symposium on High Performance Computing*, 1997, 397-410.

[4] Seiffert, U., "Artificial Neural Networks on Massively Parallel Computer Hardware", *ESANN' 12*, 2002, 319-330.

[5] http://epileptologiebonn.de/cms/front_content.php?idcat=193&lang=3&changelang=3 (Erişim Tarihi: 25.03.2011)

[6] Aydoğmuş, Z. ve Çöteli, R., "Yapay Sinir Ağları Yardımıyla İzolatör Yüzeyinde Potansiyel Tahmini", *F.Ü. Fen ve Müh. Bil. Dergisi*, 17/2, 239-246, 2005.

[7] Keleşoğlu, Ö. ve Akarsu, E.E., "Betonarme Bir Binada Yıllık Isı Kaybı ve Enerji İhtiyacının Yapay Sinir Ağları ile Belirlenmesi", *e-Journal of New World Sciences Academy Natural and Applied Sciences*, 3(2), 381-390, 2008.

[8] CUDA Programming Guide 4.2, NVIDIA CUDA, 2012

[9] Lin, J. ve Lin. J, "Accelerating BP Neural Network-Based Image Compression by CPU and GPU Cooperation", *International Conference on Multimedia Technology*, 2010, 40-44.

[10] Canto, X.S., Ramirez, F.M. ve Cetina, V.U., "Parallel Training of a Back-Propagation Neural Network Using CUDA", *ICMLA*, 2010, 307-312.

[11] Guo, L., Rivero, D., Seoane, J.A. ve Pazos, A., "Classification of EEG Signals using Relative Wavelet Energy and Artificial Neural Networks", *GCE'09*, 2009, 12-14.

[12] Nigam, V. ve Graupe, D., "A Neural-Network-based Detection of Epilepsy", *Neurological Research*, 26(1), 55–60, 2004.

[13] Fathima, T., Khan, Y.U., Bedeeuzzaman, M. ve Farooq, O., "Discriminant Analysis for Epileptic Seizure Detection", *ICDeCom' 11*, 2011, 1-5.

[14] Srinivasan, V., Eswaran, C. ve Sriraam, N., "Artificial Neural Network based Epileptic Detection Using Time Domain and Frequency Domain Features", *Journal of Medical Systems*, 29(6), 647–660, 2005.

[15] Kannathal, N., Choo, M. L., Acharyab, U. R., ve Sadasivana, P. K., "Entropies for Detection of Epilepsy in EEG", *Computer Methods and Programs in Biomedicine*, 80, 187-194, 2005.

[16] Polat, K. ve Güneş, S., "Classification of Epileptiform EEG using a Hybrid System Based on Decision Tree Classifier and Fast Fourier transform", *Applied Mathematics and Computation*, 187(2), 1017-1026, 2007.