GPU Programlama ile Yüksek Boyutlu Yoğun Matrislerin Kronecker Çarpımlarının Hesaplanması

***Ahmet Duran[1], Mehmet Tunçel[1,2], Hayati Ünsal Özer[1,3],**

*[1] İstanbul Teknik Üniversitesi Fen-Edebiyat Fakültesi Matematik Mühendisliği, İSTANBUL
E-posta: aduran@itu.edu.tr
[2] İstanbul Teknik Üniversitesi Bilişim Enstitüsü Hesaplamalı Bilim ve Mühendislik, İSTANBUL
[3] Yıldız Teknik Üniversitesi Kimya-Metalurji Fakültesi Matematik Mühendisliği, İSTANBUL

(Alınış / Received: 22.06.2019, Kabul / Accepted: 24.10.2019, Online Yayınlanma / Published Online: 01.04.2020)

Anahtar Kelimeler
Kronecker Çarpımı,
GPU Programlama,
Paralel Programlama,
Sayısal Lineer Cebir,
Yoğun Matris İşlemleri

Öz: Sayısal lineer cebir içinde yer alan ve birçok bilimsel hesaplama yöntemi içinde kullanılan önemli matris işlemlerinden biri Kronecker (tensör) çarpımıdır. Bu tip çarpımda işleme giren yoğun matris boyutu arttıkça hafıza ve zaman maliyeti çözülmesi gereken önemli bir problem olarak karşımıza çıkmaktadır. Bu çalışmamızda GPU üzerinde paralel programlama uygulaması yaparak seri programlamada karşılaşılan zaman maliyetini azaltmaya çalışıyoruz. Bunun için GPU paralelleştirme algoritması tasarlıyoruz. Rasgele sayı üreten fonksiyon ile değerleri ondalık sayılar olan gerekli büyük yoğun (dense) matrisler üretiyoruz ve CUDA iş parçacıklarıyla uygulama yapıyoruz. CPU ile GPU paralel programlama uygulamasını karşılaştırıyoruz. GPU programlama teknolojisinin bu uygulama için avantaj ve sınırlamalarını tartışıyoruz.

Computation of Kronecker Product for Large Dense Matrices Using GPU Programming

Keywords
Kronecker Product,
GPU Programming,
Parallel Programming,
Numerical Linear Algebra,
Dense Matrix Operations

Abstract: Kronecker (tensor) product is one of the important matrix operations in numerical linear algebra and used in many scientific computational methods. As the size of dense input matrix increases, memory and computation cost become a challenging issue in this type of operation. In this work, we use GPU parallel programming in order to diminish the long wall clock time consumed by serial programming. We design a new algorithm for GPU parallel programming. We generate the necessary large dense matrices using pseudo-random number generator and implement the algorithm via CUDA threads. Moreover, we compare the performance of CPU and GPU parallel programming implementations. We discuss the advantages and limitations of GPU programming technology in this particular application.

1.Giriş

Yeni yüksek başarımlı hesaplama teknolojilerinin sayısal lineer cebir içinde yer alan algoritmalar için kullanımını, avantajlarını ve sınırlamalarını analiz etmek önemlidir.

Sayısal lineer cebir içinde yer alan önemli matris işlemlerden biri de Kronecker çarpımıdır. Bu çarpım ismini 19. yy'da yaşamış, sayı teorisi, cebir ve mantık alanında önemli çalışmaları olan Leopold Kronecker adındaki Alman matematikçiden almıştır. Kronecker çarpımının tarihsel süreci için Henderson ve arkadaşlarının makalesi incelenebilir [1].

Matematik Mühendisliği Bölümü, İstanbul Teknik Üniversitesi, İstanbul

*İlgili yazar, email: aduran@itu.edu.tr

Son yıllarda bilimsel hesaplama yöntemleri içinde Kronecker çarpımının kullanımı giderek artmaktadır. Görüntü işlemede oluşturulan modellerde [2], görüntü eşleştirme için oluşturulan modüllerde [3] ve ayrıca görüntü şifreleme yönteminde güvenliği arttırmada, şifreleme ve şifre çözme anahtarı oluşturma [4,5] gibi alanlarda kullanılmaktadır.

Ağ teorisindeki çalışmalarda [6] ve modellemelerde [7] kullanıldığı görülmektedir. Tıp alanında toplanan verileri sınıflandırabilmek için tensör bazlı yöntemle oluşturulan denklemler kullanılır. Kronecker çarpımının bu denklemlerin oluşturulmasında kullanılması öneriliyor [8].

Yine son yıllarda sinyal işleme [9], çok parametreli özdeğer problemleri için çözüm elde etme [10], Fonksiyonel teori [11], olasılık modeli oluşturulması [12] gibi birçok alanda yapılmış olan çalışmalarda Kronecker çarpımının kullanıldığı görülmektedir.

Ayrıca, hızlı Fourier dönüşümü, quantum hesaplama, operatör teori, sistem teori, stokastik modeller ve fotogrametri [13] gibi alanlarda Kronecker çarpımının kullanıldığı çalışmalar mevcuttur. Kronecker çarpımları için uygulamalar arasında lineer matris denklem sistemlerinin çözümü yer almaktadır. Örneğin, A, B, C verilen matrisler, X bilinmeyen matris değişkeni olmak üzere, Lyapunov denklemi, $AX + XA^T = C$ ve Sylvester denklemi, $AX + XB = C$ vardır (detaylar için bkz. [23]).

Matris Kronecker (tensor) çarpımını tanımlamak için $A \in \mathbb{R}^{mxn}$ ve $B \in \mathbb{R}^{pxq}$ matrislerini ele alalım. Böyle iki matris için çarpım aşağıdaki gibi tanımlanabilir;

$$\boldsymbol{A \otimes B} = \begin{bmatrix} \boldsymbol{a_{11}B} & \cdots & \boldsymbol{a_{1n}B} \\ \vdots & \ddots & \vdots \\ \boldsymbol{a_{m1}B} & \cdots & \boldsymbol{a_{mn}B} \end{bmatrix} \tag{1}$$

Dolayısıyla, $A \otimes B = \mathbb{R}^{mpxnq}$ olur. Kronecker çarpımın önemli özelliklerinden dikkat çeken bir tanesi; çarpım sonucu elde edilen matrisin, çarpıma giren matrislerin sahip olduğu yapısal özellikleri gösterebilmesidir. Örneğin;

Eğer A ve B matrisleri $\left\{ \begin{matrix} pozitif\ tanımlı \\ negatif\ olmayan \\ simetrik \\ ortogonal\ (dik) \\ stokastik \\ şeritli \end{matrix} \right\}$ ise $A \otimes B$ ile oluşan matris de $\left\{ \begin{matrix} pozitif\ tanımlı \\ negatif\ olmayan \\ simetrik \\ ortogonal\ (dik) \\ stokastik \\ şeritli \end{matrix} \right\}$ olur.

Bu özelliklerin ispat ve detayı için [14]'e bakılabilir.

Bilgisayar teknolojilerinin hızla gelişim göstermesi Kronecker çarpımında yüksek boyutlu matrisleri elde etme imkânı sunuyor. GPU (Grafik İşlem Birimi) üzerinde paralel programlama uygulamaları, işlem bağımlılığı yüksek olmayan metotlar için işlerin aynı anda çalıştırılmasıyla elde edilen zaman maliyet kazanımı açısından meşhur bir hızlandırıcı olarak birçok alanda kullanılmaktadır (bkz. [18], [19] ve içindeki referanslar).

Kronecker çarpımında işleme giren matris boyutu arttıkça hafıza ve zaman maliyeti karşımıza çıkıyor. Bu çalışmamızda GPU üzerinde paralel programlama uygulaması yaparak seri programlamada karşımıza çıkan zaman maliyetini azaltmaya çalıştık. Böylelikle, yüksek boyutlu yoğun matrislerin Kronecker çarpımlarının hesaplanmasında paralel programlama kullanmanın verimi, koşulları, avantaj ve dezavantajları hakkında bir netice elde etmeye çalıştık.

Makale aşağıdaki şekilde organize edilmiştir. 2. bölümde tasarladığımız GPU paralelleştirme algoritmamız ve CUDA iş parçacıklarıyla uygulamamız anlatılmaktadır. 3. bölümde seri ve paralel testlerin yapıldığı makinaların özellikleri yer almaktadır. 4. bölümde seri algoritma hesaplama maliyeti teorik ve yaklaşık değerlerle karşılaştırılmıştır. 5. bölümde paralel algoritmanın performansı incelenmiştir. Çalışmada elde edilen sonuçlar 6. bölümde özetlenmektedir.

2. Materyal ve Metot

Kronecker çarpım uygulaması için iki kare matrisi ele aldık. Bu iki kare matrisi yoğun (dense) matrisler olarak seçtik ve rasgele sayı üreten fonksiyon ile matrislerin değerleri ondalık sayılar olarak atandı.

Yoğun matrislerle yapılan işlemlerde önemli miktarda hafızaya (RAM) ihtiyaç duyulmaktadır. Bu çalışma için yüksek başarımlı hesaplama yapabilen bilgisayar sistemi kullanıldı (UHeM). Kullandığımız bilgisayar sisteminin sahip olduğu hafıza (RAM) miktarının 128 GB ile sınırlı olmasından ötürü Kronecker çarpımında en fazla 129600×129600 boyutlu matris üretebildik.

Kronecker çarpımı yapılırken karşılaşılan sıkıntılardan bir tanesi hesaplanan matristeki ciddi ölçüdeki boyut artışıdır. Örneğin; A ve B $n \times n$ boyutlu matrisleriyle hesaplanmak istenen Kronecker çarpımında elde edilecek matrisin boyutu $n^2 \times n^2$ olacaktır. Kronecker çarpımı ile yüksek boyutlarda elde edilebilecek matris boyutu ve hesaplamada gerekebilecek hafıza miktarı için birkaç sayısal örneğe aşağıdaki Tablo-1'de bakılabilir. Gereken yaklaşık hafıza (RAM) miktarının hesaplanmasında çarpım yapılan matrislerdeki değerlerin çift değerli (double: 8 byte) değişken tipinde olduğu varsayılmıştır.

Tablo 1. Yüksek boyutlu Kronnecker çarpımı.

Girilen Matrislerin Boyutu (n)	Hesaplanan Kronecker Matris Boyutu (n^2)	Gereken Yaklaşık Hafıza (RAM-GB)
100	10000	0.745
120	14400	1.545
140	19600	2.862
160	25600	4.883
180	32400	7.821
200	40000	11.921
220	48400	17.453
240	57600	24.719
260	67600	34.047
280	78400	45.795
300	90000	60.350
320	102400	78.125
340	115600	99.565
360	129600	125.141

İlk önce Kronecker çarpımı için seri algoritma oluşturuldu ve belirlenen boyutlar için çalıştırılarak ölçümler yapıldı. Sonrasında seri algoritma GPU için paralelleştirildi ve aynı boyutlar üzerinden tekrar ölçüm yapıldı.

2.1. GPU'da paralelleştirme

Kronecker çarpımı GPU fonksiyonumuzda sonuç matrisinin bloklarının belli bir sıra halinde hesaplanmasıyla gerçekleşmektedir. $n \times n$'lik iki matrisin işlem görmesinde hesaplanan sonuç matris blokları bir satır boyunca tamamlandığında n satır ve n^2 sütunluk sonuç CPU'daki değişkene kopyalanmaktadır. Bu işlem sırasıyla tüm matris boyunca yapılmaktadır.

GPU kartlarının bellekleri ihtiyaç duyulan belleğe kıyasla oldukça sınırlıdır. Dolayısıyla GPU da çalıştırılacak algoritmada sonuç matrisini parçalı şekilde hesaplayıp sonuçları CPU (Merkezi İşlem Birimi) belleğine her ara hesaplamada göndererek bellek ihtiyacını optimize etmeyi planladık.

Algoritmamızda $n \times n$ boyutlu iki matrisin çarpımının sonucu olan $n^2 \times n^2$ boyutlu matrisi GPU belleğinde tutmak yerine sonuç matrisindeki $n \times n$ boyutlu bloklarda işlem yapan GPU fonksiyonumuzu iç içe iki *for* döngüsü içinde çağırılacak şekilde tasarladık. Ve ikinci *for* döngüsü her tamamlandığında $n \times n^2$ boyutlu toplam blok matrisini CPU belleğine taşıdık. Dolayısıyla $n \times n$ boyutlu iki matrisin Kronecker çarpımı için $n \times n$ kere GPU fonksiyonunu çağırmış ve n kere $n \times n^2$ boyutlu matris veri transferi yapmaktayız.

CUDA fonksiyonlarında iş parçacıkları warp olarak isimlendirilen aynı işlemi yapan 32'lik gruplar olarak işleme sokulur. GPU fonksiyonumuzda CUDA warp sayısı 32'nin katının iş parçacığı sayısında kullanılması performans için önemli. Dolayısıyla 32'nin katı olan $32\,x\,32$ boyutlu CUDA iş parçacığı bloklarını

kullandık. Bu blokların sayısını $n \times n$ boyutlu sonuç matris bloğunu örtecek şekilde iki boyutlu olarak ayarlayarak CUDA grid boyutumuzu belirledik. Böylece her iş parçacığının kendisi sonuç matrisinin ilgili satır ve sütununda çarpım işlemini aynı anda yapabilmektedir.

CUDA fonksiyonumuzun (kernel) genel yapısı Algoritma 1'de; bu fonksiyonu çağıran GPU algoritmamızın genel tasarım yapısı da Algoritma 2'de gösterilmiştir. Matris değişkenleri tek boyutlu dizi ve satır bazlı (row-wise) sıralama formuyla üretilmiştir. CPU'da kullanılan değişken ve fonksiyonların sonu "_cpu" ile GPU'da kullanılan fonksiyon ve değişkenlerin sonu "_gpu" ile bitmektedir. CUDA'nın iç tanımı gereği ".x" ilgili blok veya grid yapılarının x eksenindeki değerini; '.y' de y eksenindeki değerini vermektedir.

```
__global__ void blokKroneckerCarpimi_gpu(C_gpu,i ,j) {

   long int satirC = blockIdx.y*blockDim.y+threadIdx.y;
   long int sutunC = blockIdx.x*blockDim.x+threadIdx.x;

   double A_degeri = A_gpu[i*N + j];
   long int adim = j*N;
   if (satirC < N && sutunC < N) {
      C_gpu[adim + satirC * N*N + sutunC] = A_degeri * B_gpu[satirC * N + sutunC];
   }
}
```

Algoritma 1. CUDA fonksiyonumuzun (kernel) genel yapısı.

```
matrisUretDoldur_cpu(A_cpu, N, N)
matrisUretDoldur_cpu(B_cpu, N, N)
matrisUretDoldur_cpu(C_cpu, N*N, N*N)

matrisUret_cpu(A_gpu, N, N)
matrisUret_cpu(B_gpu, N, N)
matrisUret_cpu(C_gpu, N, N*N)

matrisKopyala_cpu(A_cpu,A_gpu)
matrisKopyala_cpu(B_cpu,B_gpu)

isParcacigiBlogu.x = 32;
isParcacigiBlogu.y = 32;

blokGridi.x = tavanDegeri(double(N)/double(isParcacigiBlogu.x));
blokGridi.y = tavanDegeri(double(N)/double(isParcacigiBlogu.y));

for i=1:N
    for j=1:N
        blokKroneckerCarpimi_gpu<<<blokGridi, isParcacigiBlogu>>>(C_gpu, i, j)
    blokMatrisKopyala(C_gpu, C_cpu[i*N*N*N])
```

Algoritma 2. GPU algoritmamızın genel yapısı.

GPU paralelleştirilmesinde performansı olumsuz etkileyen bir nokta CPU-GPU arası veri transferidir. GPU kartlarının yaygın kullanımında veri transferi PCI Express x16 üzerinden yapılmaktadır. Bu teknoloji ile 5.754 GB/s veri akışı elde etmekteyiz. Yalnız GPU'ların saniyede 1 trilyondan fazla işlem kapasitesine [18] ulaştığı günümüzde bu veri hızı dahi yavaş kalmaktadır.

3. Test Ortamı

Seri ve paralel testlerin yapıldığı makinaların özellikleri Tablo-2'te gösterilmiştir. Uygulamamızın yüksek bellek ihtiyacı için UHeM sisteminin hesap kaynakları kullanılmıştır.

Tablo 2. GPU test makinalarının özellikleri

CPU işlemci tipi	2 x Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz
CPU çekirdek sayısı	16
Bellek	128 GB
GPGPU tipi	NVIDIA Tesla K20m
GPU Bellek	5 GB
GPU çekirdek sayısı	2496

4. Seri Algoritma Hesaplama Maliyeti

Kronecker çarpımına giren $n \times n$ boyutlu iki matrisin çarpımı sonucu $n^2 \times n^2$ boyutlu matris oluşmakta ve sonuç matrisindeki her eleman tek tek hesaplanmaktadır. Seri algoritmamızı oluştururken belleği (RAM'i) daha verimli ve etkin kullanabilmek adına tüm matrisleri tek boyutlu dizi ve satır bazlı (row-wise) sıralama formuyla bellekte kullandık/depoladık. Algoritma 3 aşağıdaki gibi iç içe 4 for döngüsü ve en içteki döngüde çarpma işlemi içeriyor olduğundan algoritmanın işlem sayısı n^4 $flops$ (floating point operations) eder (algoritma işlem sayıları hakkında bkz [22]). Böylece seri algoritma hesaplama maliyeti (algoritma karmaşıklığı) teorik olarak $O(n^4)$ olarak gösterilir (gösterim hakkında bkz [21]). Buna aynı zamanda hesaplama maliyeti için asimptotik üst sınır da denilmektedir (bkz [21]).

$$
\begin{aligned}
&for(i = 0;\ i < n^2;\ i = i + n)\\
&\ \ for(k = 0;\ k < n^2;\ k = k + n)\\
&\ \ \ \ for(j = 0;\ j < n;\ j++)\\
&\ \ \ \ \ \ for(l = 0; l < n; l++)\\
&\ \ \ \ \ \ \ \ Kron\_Mat[i * n^2 + (j + k) * n + l\ ] = Mat1[i + j\ ] * Mat2[k + l\ ]
\end{aligned}
$$

Algoritma 3. Seri algoritmamızın genel yapısı

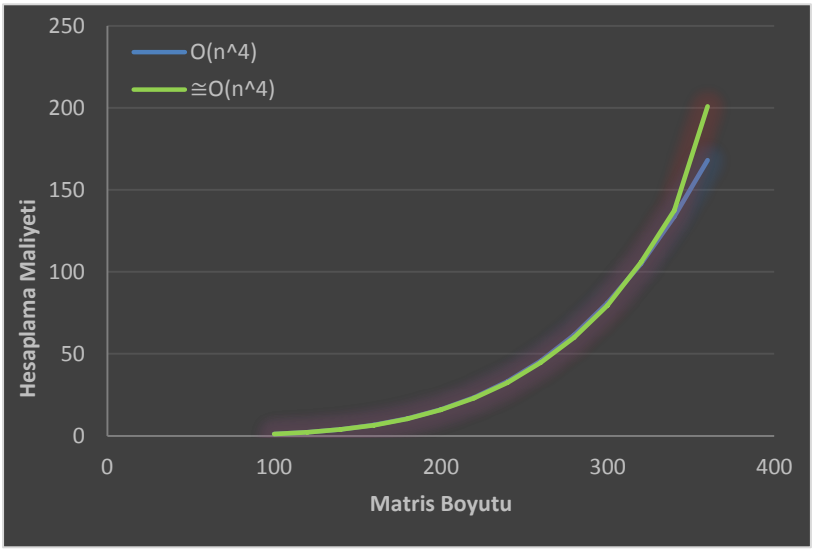
Algoritmanın uygulamadaki hesaplama maliyeti için aşağıdaki Tablo-3'e bakılabilir. Tabloda matris boyutundaki artışın hesaplama süresine olan etkisi ve bunun algoritma karmaşıklığı ile olan ilişkisi incelendi. Bunun için öncelikle Kronecker çarpımı yapılacak matrislerin çeşitli boyutlarına karşılık seri algoritmanın hesaplama süreleri ölçüldü. Süreler hesap süresi sütununda gösterilmektedir. Daha sonra, matris boyutlarına karşı gelen süre ile ilk matris boyutuna karşı gelen süre arasındaki büyüme oranı hesaplanarak yaklaşık bir algoritma karmaşıklığı elde edildi. Bu sonuçlarda $\cong O(n^4)$ sütunun altında belirtildi. Örneğin matris boyutunun 100'den 200 çıktığı durumda seri algoritmada hesaplama süresi yaklaşık olarak 15,66 olarak artmaktadır. Bu durum boyutun 2 kat arttığı durumda hızın teorik olarak $O(n^4) = O(2^4) = 16$ kat artması ile uygunluk göstermektedir

Tablo-3 ve grafik-1 incelendiğinde çarpıma giren matris boyutunun 300'e kadar olduğu durumda yaklaşık algoritma karmaşıklığı $\cong O(n^4)$ için teorik karmaşıklığın $O(n^4)$ altında ve ona yakın sonuçlar elde edilirken, 300'den sonraki boyutlarda $O(n^4)$ 'den daha büyük ve giderek artan değerler elde edildiği görülmektedir. Bu durum bize seri algoritmanın yüksek boyutlarda daha düşük performans göstereceği izlemini vermektedir. Buna sebep olarak yüksek boyutlarda ihtiyaç duyulan ciddi miktardaki hafıza maliyeti gösterilebilir.

Hesap süresine karşılık teorik $O(n^4)$ ve yaklaşık $\cong O(n^4)$ için doğrusal regresyon yapıldığında $T(n) = kO(n^4) + c$ denklemindeki k kat sayı değeri $O(n^4)$ için $\cong$ 0,573 ve $\cong O(n^4)$ için $\cong$ 0,522 olarak elde edildi. R^2 için $\cong$ 0,9917 değeri ve $ayarlı\ (adjusted)\ R^2$ için $\cong$ 0,9910 değeri hesaplandı. (R^2: Regresyon denkleminin başarısını ölçmede kullanılan belirleme katsayısı. $Ayarlı\ (adjusted)\ R^2$: R^2 'nin örnek büyüklüğü dikkate alınarak hesaplanmış hali.)

Tablo 3. Seri algoritma hesaplama maliyeti

Matris Boyutu (n)	CPU (Seri) Hesap Süresi (sn)	$O(n^4)$	$\cong O(n^4)$
100	0,518	1	1
120	1,056	2,073	2,039
140	1,957	3,842	3,777
160	3,327	6,554	6,423
180	5,320	10,498	10,270
200	8,105	16	15,647
220	11,845	23,426	22,867
240	16,760	33,178	32,356
260	23,070	45,698	44,537
280	31,012	61,466	59,869
300	41,142	81	79,424
320	54,786	104,858	105,765
340	71,176	133,634	137,405
360	104,043	167,962	200,855



Grafik 1. Seri algoritma hesaplama maliyeti

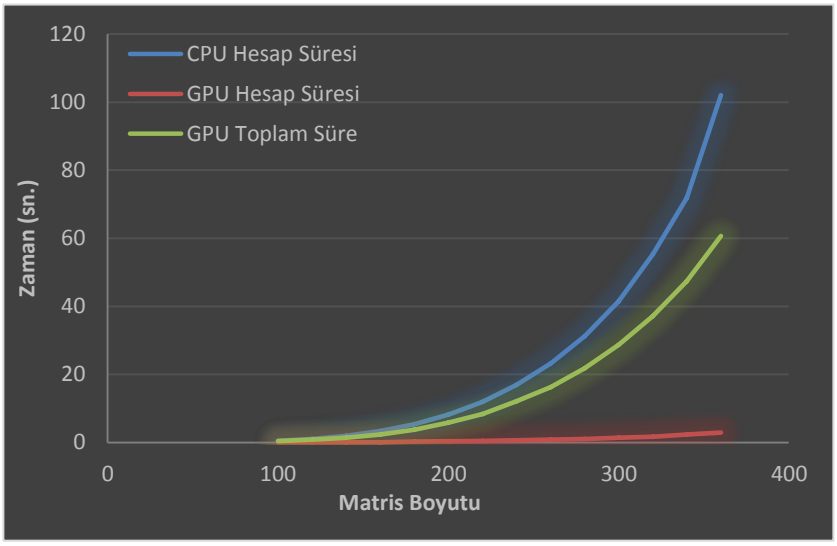
5. Paralel Algoritma Performansı

Paralel algoritmanın ise seri algoritmaya göre her zaman daha iyi sonuçlar verdiği görülmektedir. Paralel algoritmanın seri algoritmaya göre performansı için aşağıdaki Tablo-4 incelenebilir. Performans analizi için sadece GPU'deki hesap süresini dikkate aldık. Diğer bir deyişle veri kopyalama süresi analize dâhil edilmedi. Aşağıdaki tablo; Kronecker çarpımı yapılacak matrislerin çeşitli boyutlarına karşılık seri ve paralel algoritmaların hesaplama sürelerini ve tablonun son sütununda ise her boyut için bu sürelerin oranını içermektedir.

Tablo-4 ve Grafik-2 incelendiğinde çarpıma giren matris boyutu arttıkça seri algoritma hesap süresinin paralel algoritma hesap süresine oranı artmaktadır. Yani boyut arttıkça paralel algoritmanın seri algoritmaya göre daha fazla hızlandığını görebilmekteyiz. Böylece sistemin daha verimli çalışmasıyla paralel algoritmanın yüksek boyutlara çıkıldıkça performansının daha da arttığını söyleyebiliriz.

Tablo 4. Paralel algoritma performansı

Matris Boyutu (n)	CPU (Seri) Hesap Süresi (sn)	GPU (Par.) Hesap Süresi (sn)	GPU Veri Kopy. Süresi (sn)	GPU Toplam Süre (sn)	$\frac{CPU\ seri\ H.S.}{GPU\ Par.\ H.S.}$
100	0,522	0,069	0,356	0,425	7,608
120	1,065	0,102	0,712	0,814	10,418
140	1,975	0,147	1,302	1,449	13,432
160	3,356	0,201	2,202	2,403	16,671
180	5,366	0,283	3,505	3,788	18,962
200	8,176	0,410	5,388	5,797	19,961
220	11,946	0,509	7,851	8,360	23,454
240	16,910	0,660	11,500	12,161	25,606
260	23,272	0,900	15,410	16,311	25,853
280	31,276	1,084	20,721	21,805	28,842
300	41,528	1,411	27,315	28,726	29,431
320	55,296	1,727	35,422	37,149	32,024
340	71,837	2,345	45,056	47,401	30,629
360	102,061	2,903	57,750	60,653	35,160



Grafik 2. Seri ve paralel algoritma performanslarının karşılaştırılması

6. Tartışma ve Sonuç

Kronecker çarpımı ile yüksek boyutlardaki yoğun matrisleri elde etme çalışmamızda, seri algoritmanın GPU algoritmamız ile paralelleştirilmesiyle hesaplamanın daha hızlı yapılabileceğini gözlemledik.

GPU kullanımının CPU'ya kıyasla avantajını hesaplama yükü olarak görmekteyiz. Seri algoritmamıza göre yaklaşık 35 kata varan daha iyi bir performans elde edebildik. Yalnız toplam süre olarak baktığımızda GPU'larda bir kısıtlama olarak karşımıza çıkan veri transferinin ekstra yükünden dolayı seri algoritmadan daha iyi olmakla beraber daha önce tamamen CPU'da yaptığımız paralel algoritma uygulamasına [15] kıyasla daha yavaş sonuç elde etmekteyiz. Yeni çıkan yüksek bellekli GPU kartları ve hızlanan yeni veri transfer yolları ile bu kısıtlama gittikçe azalmaktadır.

Teşekkür

Bu çalışmada kullanılan hesaplama kaynakları Ulusal Yüksek Başarımlı Hesaplama Merkezi (UHeM), 4006072019 numaralı desteğiyle sağlanmıştır.

Kaynakça

[1] Henderson, H.V., Pukelsheim, F., Searl, S.R. 1983. On the history of the Kronecker product. Linear Multilinear Algebra. Taylor&Francis, 14(1983), 113-120.

[2] Liu, J., Psarakis, E.Z., Feng, Y., Stamos, I. 2018. A Kronecker Product Model for Repeated Pattern Detection on 2D Urban Images. Transactions on Pattern Analysis and Machine Intelligence, IEEE.

[3] Shen, Y., Xiao, T., Li, H., Yi, S., Wang, X. 2018. End-to-End Deep Kronecker-Product Matching for Person Re-Identification. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 6886-6895.

[4] Kumar, R., Bhaduri, B. 2017. Optical image encryption using Kronecker product and hybrid phase masks. Optics & Laser Technology. Elsevier 95(2017), 51-55.

[5] Maan, P., Singh, H., Kumar, A .C. 2019. Optical Asymmetric Cryptosystem Based on Kronecker Product, Hybrid Phase Mask and Optical Vortex Phase Masks in the Phase Truncated Hybrid Transform Domain. 3D Reseach. Springer(2019), 10:18.

[6] Syed Ali, M., Yogambiagi, J. 2016. Synchonization of complex dynamical networks with hybrid coupling delays on time scales by handling multitude Kronecker product terms. Applied Mathematics and Computation. Elsevier, 291(2016), 244-258.

[7] Moreno, S., Neville, J., Kirshner, S. 2018. Tied Kronecker Product Graph Models to Capture Variance in Network Populations. Transaction on Knowledge Discovery from Data (TKDD). ACM, Vol. 12, Is. 3, No. 35.

[8] Boussé, M., Goovaerts, G., Vervliet, N., Debals, O., Van Huffel, S., De Lathauwer, L., 2017. Irregular hearthbeat classification using Kronecker Product Equations. 39th Annual Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Seogwipo, South Korea, 438-441.

[9] Roth, I., Flinth, A., Kueng, R., Eisert, J., Wunder, G. 2018. Hierarchical restricted isometry property for Kronecker product measurement. 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton) IEEE, Moticello, IL, USA.

[10] Hochstanbasch, M. E., Meerbergen, K., Mengi, E., Plestenjak, B., 2019. Subspace Methods for Three-Parameter Eigenvalue Problems. Numer. Linear Algebra Appl. 26(4):e2240

[11] Sharma, A., Suryanarayana, P. 2018. On real-space Density Functional Theory for non-orthogonal crystal systems: Kronecker product formulation of the kinetic energy operator. Chemical Physics Letters, Elsevier, 700(2018), 156-162.

[12] Montúfar, G., Morton, J. 2017. Dimensions of Marginals of Kronecker Product Models. Journal on Applied Algebra and Geometry, SIAM, 1(2017), 126-151.

[13] Van Loan, C. 2000. The ubiquitous Kronecker product. Journal of Computational and Applied Mathematics, Elsevier, 123(2000), 85-100.

[14] Van Loan, C., Pitsianis, C. F. 1993. Approximation with Kronecker Product. In: Moonen M.S., Golub G.H., De Moor B.L.R., (eds), Linear Algebra for Large Scale and real-Time Applications. NATO ASI Series, Springer, 232(1993), 293-314.

[15] Duran, A., Özer, H. Ü., Tunçel, M. 2019. Yüksek Boyutlu Kronecker Çarpımı için İş Parçacığı Tabanlı Paralel Programlama Uygulaması. 2. Uluslararası İstatistik, Matematik ve Analitik Yöntemler Kongresi, 28 Mart İstanbul, 256-262.

[16] UHeM, Ulusal Yüksek Başarımlı Hesaplama Merkezi, http://www.uhem.itu.edu.tr/

[17] Zarges, N. M. 2019. Evaluation of On-Node GPU Interconnects for Training Deep Neural Networks. Technical University of Munich, Department of Informatics, B.S. Thesis, 97s, München.

[18] NVIDIA TESLA K-Series, 2013, Datasheet, https://computing.llnl.gov/tutorials/linux_clusters/gpu/Tesla-K10K40-datasheet.pdf (Erişim Tarihi: 01.03.2019)

[19] Duran, A., Piskin, S., Tuncel, M., 2016. Evaluating the maturity of OpenFOAM simulations on GPGPU for bio-fluid applications. Proceedings of the Emerging Technology (EMiT) Conference, Supercomputing Center, 2-3 June, Barcelona, Spain, 11-14.

[20] Li, F., Ye, Y., Tian, Z., Zhang. X. 2018. CPU versus GPU: which can perform matrix computation faster—performance comparison for basic linear algebra subprograms. Neural Computing and Applications, 1-13.

[21] Cormen, T. H., Leisorson, C. E., Rivest, R. L., Stein, C. 2009. Introduction to Algorithms, The MIT Press, London, England.

[22] Trefethen, L. N., Bau, D. 1997. Numerical Linear Algebra, SIAM, Philadelphia, USA.

[23] Horn, R. A., Johnson, C. R. 1991. Topics in Matrix Analysis, Cambridge University Press.