**KARADENİZ TEKNİK ÜNİVERSİTESİ**
**BİLGİSAYAR MÜHENDİSLİĞİ**

# PARALEL BİLGİSAYARLAR

2.ÖDEV RAPORU

DERSİN SORUMLUSU
**DOÇ.DR.İBRAHİM SAVRAN**
HAZIRLAYANLAR
**295083 MURAT CAN VARER**
**295079 TÜLİN YARDIMCI**

# CUDA

CUDA, NVIDIA'nın GPU (grafik işlem birimi) gücünü kullanarak hesaplama performansında büyük ölçüde artışlara olanak veren paralel hesaplama mimarisidir.CUDA,C++ ve Fortran gibi dilleri desteklediği için öğrenilmesi ve kullanılması basit bir tool kit'tir.

Yazılım geliştiriciler, bilim adamları ve araştırmacılar bugüne kadar satılan milyonlarca CUDA etkinleştirilmiş GPU ile görüntü ve video işlem, hesaplamaya dayalı biyoloji ve kimya, akışkan dinamiği, bilgisayarlı tomografi, sismik analiz, ışın izleme ve çok daha fazlası dahil olmak üzere geniş bir aralıkta kullanım alanları bulmaktadır.Hesaplama, CPU üzerindeki "merkezi işlemden" CPU ve GPU üzerindeki "birlikte işleme" doğru bir evrim geçirmektedir. NVIDIA, hesaplamadaki bu yeni paradigmaya olanak vermek için uygulamacılar için önemli bir taban olan ve GeForce, ION Quadro ve Tesla GPU'lar üzerinde temin edilen CUDA paralel hesaplama mimarisini geliştirdi.

# FLOPS

Floating point operations, mikroişlemcilerin hızını ölçmek için kullanılır.FLOPS, kesirli sayıları içeren tüm işlemleri içerir. Tamsayı işlemlerinden daha fazla süreceği için mikroişlemci testlerinde kullanılır.

Bir megaFLOPS (MFLOPS) saniyede bir milyon floating-point işlemi ve bir gigaFLOPS (GFLOPS) saniyede bir milyar kayan nokta operasyonuna eşittir. TeraFLOPS (TFLOPS) saniyede bir trilyon kayan nokta operasyonuna eşittir.

Formül aşağıdaki gibidir :

GFlops = (CPU speed in GHz)x(number of CPU cores)x(CPU instruction per cycle)x(number of CPUs per node)

# HIZLANMA ve VERİMLİLİK nedir?

Hızlanma, bir programın seri olarak (bir işlemci ile) paralel olarak (birçok işlemci ile) yürütülmesi gereken süreye bölünmesi ile geçen süre olarak tanımlanır.

$$Hızlanma = \frac{Toplam\ Geçen\ Sure}{Seri\ Geçen\ Sure + \frac{Paralel\ Geçen\ Sure}{islemci\ sayisi}}$$

Verimlilik ,hızlanmadan elde edilen sonucun 100 ile çarpılıp kullanılan işlemci sayısına bölümüdür.

$$Verimlilik = \frac{Hızlanma * \%100}{islemci\ sayisi}$$

# Projenin İçeriği

Sparse(Seyrek) matris çarpımı, kısmi diferansiyel denklemleri çözerken veya bilimsel mühendislik uygulamalarında görülür .Bu projede Sparse matris denilen içeriğinin çoğunluğunun 0 değerinden oluştuğu matrisi daha hızlı paralel bir şekilde nasıl bir vektör ya da matris ile çarpabiliriz onu yapıyoruz.
Bunun akabinde Sparse denilen matrisi özel olarak 3 parçaya ayırılıp çarpma işleminin yapılması göstermek ve geçen süreleri hesaplanması(blok ve thread sayısına göre değişiklikler ile birlikte) gösterilme istendi.

Bu Sparse matris 3 parçasından oluşan değerler şunlardır;

$$\mathtt{A} = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix}$$

$$\mathtt{ptr} = \begin{bmatrix} 0 & 2 & 4 & 7 & 9 \end{bmatrix}$$
$$\mathtt{indices} = \begin{bmatrix} 0 & 1 & 1 & 2 & 0 & 2 & 3 & 1 & 3 \end{bmatrix}$$
$$\mathtt{data} = \begin{bmatrix} 1 & 7 & 2 & 8 & 5 & 3 & 9 & 6 & 4 \end{bmatrix}$$

Bizden istenilen **ptr**(10001 boyutlu) **data**(99911 boyutlu) **indices**(99911 boyutlu) değerleri kullanılarak, bir **A[10000]** büyüklüğünde bütün değerleri 1 den oluşan vektörü optimum şekilde çarpma ve geçen süreleri hesaplamaktır.

# Yapılan İşlemler

Çarpma işlemindeki algoritma şu şekildedir:

**ptr** ile sırası ile o satırları alarak hangi **val** değerlerimizi kullanacağımızı belirlemektir ve daha sonra bu **val** değerlerimizi çarpacağımız vektör veya matristeki hangi indisine denk düştüğünü **A[col_in[i]]** (i değeri hangi row_pt'de iş yapıyorsak onları almak için) ile belirledikten sonra bunları çarpıp bu **ptr**'ler arasında çıkan sonuçlar ile toplamaktır. En son **ptr** geldiğimizde ise döngüden çıkarak işlemi sonlandırmaktır.

Fakat burada bir optimizasyon yapmamız gerekirse şu şekilde olmalıdır :
Bizim çarptığımız vektörün bütün indis değerleri "1" karşılık düştüğü için **A[indices[i]]'**yi hesaplayıp çarpmamıza gerek kalmayacak çünkü hangi indisle çarparsak çarpalım hep 1 gelecek ve sürekli aynı şeyi tekrarlayacak, bu değeri kodumuzdan kaldırdığımızda da yine bize aynı sonucu verecektir.

# Algoritmalar:

*CPU kodu :*

```
for (int i = 0; i <10000; i++){

        if (i < 10000)//son satıra girmemek için
        {
                float sonuc = 0;

                int satirBasi = row_P[i];
                int satirSonu = row_P[i + 1];
                for (int k = satirBasi; k < satirSonu; k++){
                        ToplamCarpim += data[k – 1];
                }
                y[i] = ToplamCarpim;

        }
    }
```

*GPU kodu:*

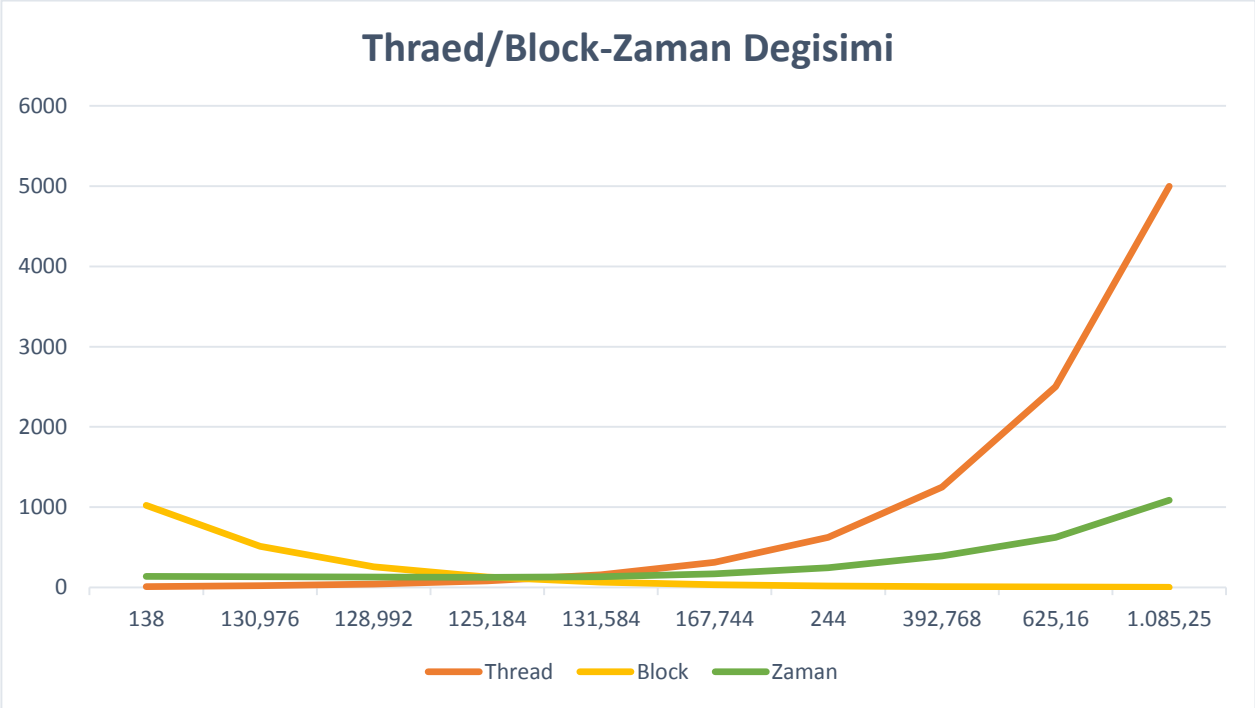
```
int satir = threadIdx.x + blockIdx.x*blockDim.x;
    if (satir< 10000)//10000'den fazla thread'in çalışmaması için
        {
                float ToplamCarpim = 0;

                int satirBasi= row_P[row];
                int satirSonu = row_P[row + 1];
                for (int i = satirBasi; i < satirSonu; i++)
                        ToplamCarpim += data[i – 1];
                y[satir] = carpan;
        }
```

Örneğin ilk 50 çıktısı( Her satır sonu diğer satır sonunun diğer satır başından yani ilk satırın sonuna bakacak olursak **4,54 5,621 5,22 , … ,2,852 6,664** gibi eklendiğini varsayınız.)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 2,3 | 4 | 9,239 | 15,638 | 1,678 | 5,002 | 7,621 | 2,907 | 2,11 | 4,54 |
| 5,621 | 5,22 | 2,47 | 7,669 | 4,832 | 7,68 | 7,091 | 2,529 | 5,48 | 2,852 |
| 6,644 | 1,512 | 7,397 | 5,361 | 5,728 | 1,26 | 3,75 | 6,746 | 5,116 | 4,222 |
| 5,633 | 7,812 | 3,154 | 4,866 | 4,63 | 1,922 | 5,224 | 0,861 | 5,311 | 4,205 |
| 3,289 | 5,129 | 2,664 | 5,383 | 4,286 | 7 | 3,918 | 3,315 | 4,059 | 3,121 |

# Seri ve paralel çalışan fonksiyonların zamanları ve İş Paylaşımı

## 1. Paralel Çalışma




| Block | Thread | Microsaniye |
|---|---|---|
| 10 | 1024 | 138 |
| 20 | 512 | 130,976 |
| 40 | 256 | 128,992 |
| 80 | 128 | 125,184 |
| 157 | 64 | 131,584 |
| 312 | 32 | 167,744 |
| 625 | 16 | 244 |
| 1250 | 8 | 392,768 |
| 2500 | 4 | 625,16 |
| 5000 | 2 | 1.085,25 |

Grafiği açıklayacak olursa: Thread ile Block sayisinin carpimi <10000 olacak şekilde belirlendikten sonra hesaplama sürelerine ilişkin grafiktir.Değeleri milisaniye olarak grafikte göstermek istediğimiz net bir şekilde farkı göremiyorduk bunun üzerine zamanı **microsaniye'ye** cevirdik ve bu şekil ortaya çıktı.
**Yeşil** ile belirtiğimiz en verimli sonuçtur.
Bu grafikte block sayini arttırıp thread sayisini azalttığımızda programın koşma zamanı artmaktadır.Bunun nedeni çok fazla block olduğu için bir birleri ile haberleşmelerinden kaynaklanmaktadır.
Bizim 10000 boyutunda bir matrisimiz olduğu için o kadar thread'e ihtiyacımız var aksi taktirde çarpma işlemini eksik yapardık.Bunun için thread sayısını yarı yarıya azaltırken blok sayısını da 2 katını alarak arttırdığımızda geçen süre turuncu renkte de görüldüğü gibi artmaktadır. Bunun sebebi ise bloklar arası ortak bir bellek üzerinden haberleşme olmadığı içindir. Bunu aşmak için **shread_memory** kavramı kullanarak yapabilir .Bizden istenilen şimdilik bu kadardı.

# 2. Seri Çalışma

Cpu'da yazdığımız kodun zamanını runtime süresini bu şekilde aldık

```
clock_t tStartCPU = clock();
cpu(vectorRow_p, vectorColumn_p, vectorData_p, vectorCarpan_p);
clock_t tStopCPU = clock();
printf("CPU'da gecen sure: %.5fs\n", (double)(tStopCPU - tStartCPU));
```

Şeklinde çalıştırdığımız da **1 sn** olarak sonuçlanıyor.

## KAYNAKLAR


http://www.nvidia.com.tr/object/cuda-parallel-computing-tr.html

http://www.webopedia.com/TERM/F/FLOPS.html

http://www.novatte.com/our-blog/197-how-to-calculate-peak-theoretical-performance-of-a-cpu-based-hpc-system

https://home.wlu.edu/~whaleyt/classes/parallel/topics/amdahl.html

http://people.cs.georgetown.edu/~jfineman/papers/csb.pdf

http://www.geeksforgeeks.org/sparse-matrix-representation/