Araştırma Makalesi
Adıyaman Üniversitesi
Mühendislik Bilimleri Dergisi
5 (2016) 71-76

Asal sayıların Eratosten kalburu algoritması kullanılarak bulunmasında GPU ve CPU başarımlarının analizi

Mehmet TURAN[1,*] , M. Akif NACAR[2]

[1] Harran Üniversitesi, Mühendislik Fakültesi, Elektrik-Elektronik Mühendisliği Bölümü, Şanlıurfa/Türkiye
[2] Harran Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Şanlıurfa/Türkiye

ÖZET:

Elektronik hesaplama yöntemlerinin yaygınlaşması ile birlikte asal sayıların hesaplanması donanım testlerinde yaygın olarak kullanılmaya başlandı. Ağ güvenlik protokolleri, sesli iletişim ve kriptografi uygulamaları gibi birçok bilim dalında asal sayılar ve asal sayı üreteçleri yaygın olarak kullanılmaktadır. Bu çalışmada da Eratosten Kalburu Algoritması Grafik İşlem Birimi ile Merkezi İşlem Birimi için ayrı ayrı kullanılarak asal sayıların hesaplanmasındaki performans farklılıkları NVIDIA Tesla K40, NVIDIA Quadro 600 ve konvansiyonel CPU başarımları karşılaştırıldı. Hesaplama sonucunda GPU ile yapılan asal sayı hesaplamalarının CPU'ya oranla hesaplama süresinde kayda değer bir azalma olduğu gözlemlenmiştir.

Anahtar Kelimeler: Eratosten Kalburu; GPU; Paralel Hesaplama; Asal Sayılar

An analysis of performance of finding prime numbers on GPU and CPU by using sieving algorithm

ABSTRACT

Since the introduction of electronic methods of calculation, finding prime number programs has become a good method for hardware testing. Almost in all branches of science especially in network security protocols, voice communication and cryptography applications prime numbers and prime number generators are used intensely. This paper clarifies performance differences of finding prime numbers by using Sieve of Eratosthenes Algorithm on both of Graphics Processing Unit and Central Processing Unit. NVIDIA Tesla K40 and NVIDIA Quadro 600 were used to compare results with conventional CPU. The results indicate that finding prime number calculations using a GPU can significantly reduce run time in comparison with that using a conventional CPU.

Keywords: Sieve of Eratosthenes; GPU; Parallel Computing; Prime Numbers

1. Giriş

Yeni bir asal sayının bulunması kriptografi için çok büyük bir sürpriz olmasa da matematikçiler asal sayıların daha düşük maliyet ve daha kısa zamanda hesaplanması için yeni yöntemler aramaktadırlar. Yoğun aritmetik işlem gerektiren asal sayıların hesaplanması, sayılar büyüdükçe daha zor hale gelmektedir.

* Sorumlu Yazar: mturan@adiyaman.edu.tr

Asal sayıları tanımlamak basit olsa da asıl sorun yeni bir büyük asal sayının bulunması sürecinde yaşanmaktadır. Günümüze kadar asal sayıların dağılımı, bir sonraki büyük asal sayı, asal sayı hesaplama algoritmaları vb. konularda matematikçiler birçok teori geliştirdiler. En bilinen asal sayılara ilişkin çalışma Öklid'in yaklaşık olarak M.Ö. 300'lü yıllarındadır ve bu aynı zamanda aritmetiğin temel esaslarını teşkil etmektedir[1]. Yine Yunan matematikçilerin keşfettiği Eratosten Kalburu büyük sayıların hesaplanmasında maliyet ve zaman açışından pek yeterli değildi[2]. Daha sonra 17. yüzyılda Fermat ve Euler asal sayıların özellikleri ve daha anlaşılır olması için çalışmalar yaptılar. Günümüzde de Great Internet Mersenne Prime Search (GIMPS) projesi kapsamında en büyük asal sayıyı bulmak için çalışmalar yapılmaktadır.

Asal sayılara genelde özel bilgilerin korunabilmesi için şifreleme teknolojilerinde, bankalar kredi kartlarında, fizikçiler kuantum kaos teorisinde, internet şifrelerinde ve haberleşme sistemleri gibi birçok alanda ihtiyaç duyulmaktadır[3].

Teknolojinin gelişmesi ile birlikte birçok elektronik hesaplama tekniği ve aracı da yaygınlaşmaya başlamıştır. Paralelleştirme, birkaç aktivitenin aynı anda gerçekleşmesi olarak tanımlanırken, paralel hesaplama aynı görevin parçalara bölünerek çoklu işlemcilerde eş zamanlı olarak işletilmesidir[4]. İlk piyasaya sürülmesinden bu yana Grafik Hesaplama Birimi (GPU) paralel hesaplama alanında büyük bir ilerleme kaydetmiştir. NVIDIA şirketi GPU'nun güçlü hesaplama yeteneğini CUDA hesaplama platformu ve programlama modeli GPU üzerinde hesaplama işlemlerinin gerçekleştirilebilmesini sağlamaktadır.

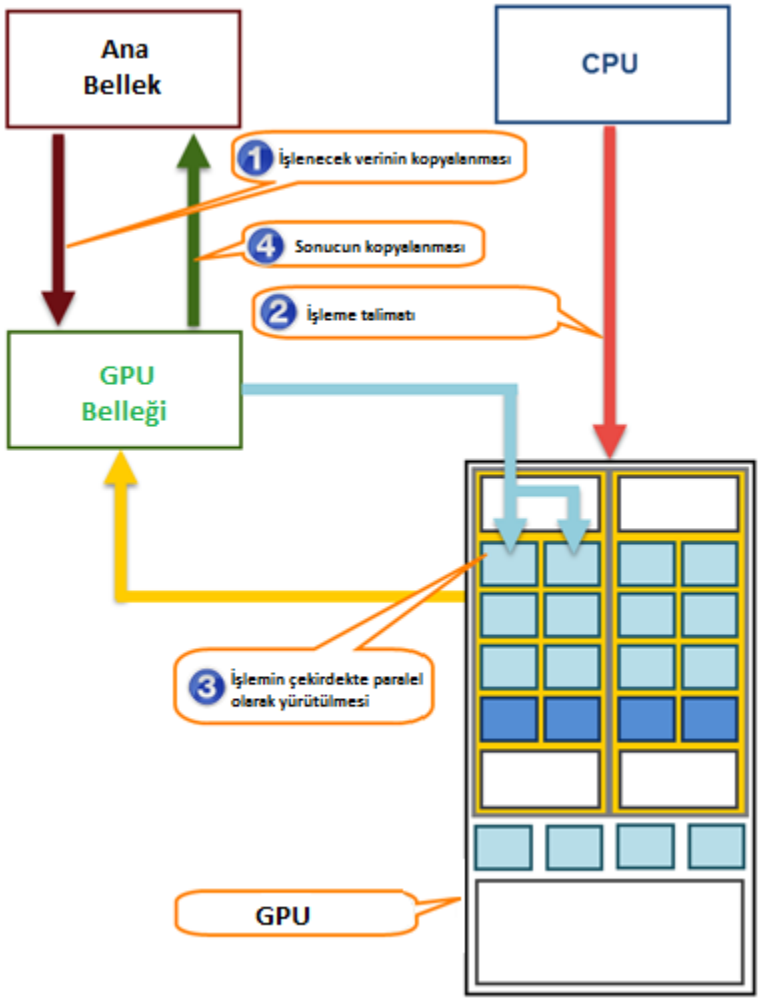
GPU tabanlı simülasyon sisteminde, hesaplama maliyetinde ve zamanında gözle görülür bir düşüş yaşanması daha fazla elemanın simülasyona dahil edilmesini ve böylece bu elemanların geometrik olarak daha iyi gösterilmesini ve daha hassas sonuçlar elde edilmesini sağlar[5].

GPU mimarisi sayesinde, GPU üzerindeki thread/block ve block/grid sayılarının doğru belirlenmiş olması ile hesaplama süreleri önemli ölçüde azaltılabilirken, bu değerlerin yanlış belirtilmesi hesaplama süresini olumsuz yönde etkilemektedir[6]. Yine GPU üzerinde yürütülen hesaplama işleminin maksimum performans ile gerçekleştirilebilmesi için CUDA uygulamasındaki algoritmanın uyumlu olması gerekmektedir. Doğru algoritma seçimi CUDA kodunun en önemli bileşenidir[7]. GPU belleğinin doğru kullanılması da bir diğer önemli etkendir. GPU'daki tüm belleğin yapılacak olan işlem için ayrılması host-device arasındaki veri transferi için gereken zaman kaybını önlemek için bir avantaj gibi görünse de daha fazla bellek gerektiren durumlar için tercih edilen bir yöntem değildir. Bu durum çoklu GPU paralelizmi kullanılarak aşılabilir[8].

Bu çalışma asal sayıların bulunmasında, bilgisayar performansını büyük ölçüde arttıran GPU, Eratosten Algoritmasının GPU kernel koduna dönüştürülmesiyle for-loop vd. iş parçacıkları thread'ler üzerine yürütülerek GPU ve CPU işlemcileri arasındaki hesaplama süreleri karşılaştırılacaktır.

2. CUDA ve GPU Programlama

CUDA'nın paralel düzenlenmiş işlemcileri kendi aralarında veri alışverişini yapabilirken, CPU da tek sıralı bir şekilde gerçekleşecek görevi GPU işlemcileri üzerinde yayarak yerine getirirler. Parçalanan ve GPU uyumlu haline getirilen görevin çoklu işlemciler üzerinde eş zamanlı olarak yürütülmesi ve istenen sonucun daha hızlı bir şekilde elde edilmesi amaçlanmaktadır. Şekil 1'de gösterildiği gibi, GPU iş akışı giriş verilerinin ana bellekten GPU belleğine kopyalanması ile başlar, işlem talimatının verilmesi ve her çekirdekte paralel olarak görevin yürütülmesi ile devam eder ve sonucun GPU belleğinden ana belleğe kopyalanması ile görev tamamlanmış olur.



Şekil 1. GPU iş akışı

CUDA ile paralelleştirmede hiyerarşik thread yönetimi ve çok boyutlu indeksleme olmak üzere iki önemli özellik vardır. Thread'ler block'lar tarafından ve block'lar ise grid'ler tarafından yönetilir, yani her thread kümesi bir block'a ve her block kümesi de bir grid'e aittir.

CUDA protokolü GPU da çalıştırılacak program kodu için kullanılır. NVIDIA tarafından sunulan bu protokol C++, C, Python, FORTRAN gibi programlama dilleri ve Opencast ve OpenCL gibi programlama yapılarını kullanarak uygulamalar geliştirilebilir. Device GPU kartını ve belleğini, host ise CPU kartını ve belleğini ifade eder. Host üzerinde yürütülecek olan kodlar CPU ve GPU belleğini yönetebilir ve device üzerindeki fonksiyonların yürütülmesi için kernel'i başlatabilir.

Tipik CUDA işlem sırası aşağıdaki gibidir:

1. Host ve device belleğinin belirtilmesi ve ayrılması.
2. Host verilerinin başlatılması.
3. Host'tan device'ye verilerin transfer edilmesi.
4. Sonuçların device'den host'a transfer edilmesi.
5. GPU'nun sonlandırılması.

CUDA'da cudaMemcpy GPU'dan CPU'ya ve CPU'dan GPU'ya verilerin kopyalanmasını 1 GB/s üzerinde gibi yüksek bir hızda gerçekleştirir. Fakat GPU kendi içerisindeki belleği kullanarak CPU belleklerinin kullanılmasına göre daha yüksek iletim hızı elde eder.

3. Yöntem

Şu ana kadar asal sayı üretmek için kanıtlanmış bir formül henüz keşfedilmemiştir. Bilgisayarlar hızlandıkça deneme-yanılma yoluyla, matematiksel hesaplamalar ve algoritmalar yardımıyla asal sayılar bulunmaya devam etmektedir.

Bu çalışmada da daha önce yapılan çalışmalardan farklı olarak Eratosten eleme algoritması GPU kullanılarak gerçekleştirilecektir. Şekil 2'de belirtilen Eratosten algoritmasında [00000000…00000] numara dizisinin her biri başlangıçta asal olarak kabul edildikten sonra ilk iki dizi elemanı (0 ve 1) asal olmadığından 1 olarak işaretlenir ve 2'den başlayarak her sayının kendi karesinden küçük ve 1'den büyük olan katları için dizideki sıra değeri 1 olarak işaretlenir. Örnek olarak 10'a kadar olan bir sayı dizisi (0,1,2,3,4,5,6,7,8,9,10) için [11001010111] şeklinde bir sonuç elde edilecektir. Elde edilen dizideki 1'den farklı olan her 0 değeri bir asal sayıyı temsil etmektedir.

```
Input: an integer n > 1

Let A be an array of Integer values, indexed by integers 0 to n,
initially all set to 0.
A[0]=1
A[1]=1
for i = 2, 3, 4, ..., not exceeding √n:
  if A[i] is 0:
    for j = i², i²+i, i²+2i, i²+3i, ..., not exceeding n:
      A[j] := 1
```

Şekil 2. Eratosten Kalburu algoritmasının pseudocode ile gösterimi

3.1. Başarımın Hesaplanması

Bu araştırmada, hem host hem de device asal sayıların bulunmasında ayrı ayrı kullanılırken uygulamanın çalışma sürecinde geçen zamanın karşılaştırması farkı göstermek açısından önem taşımaktadır. Süre farkı incelenirken;

1. Cihazdaki gelişmeyi görmek için, hem device hem de host için aynı maksimum sayıya kadar asal sayıların bulunması gerekmektedir.
2. Performans farkının gösterilmesi için CPU ve GPU'nun farklı nesillerindeki sonuçları karşılaştırılacaktır. Bunlar;
   a) Intel Core 2 Duo P8600
   b) Intel Xeon E5-1607
   c) Quadro 600
   d) Tesla K40

4. Uygulama Sonuçları

Uygulama ile asal sayıları bulmak için aynı algoritma CPU ve GPU üzerinde yürütülmüştür. Performans farkı Çizelge 1 ve Çizelge 2'de gösterilmektedir. Sonuçlar karşılaştırıldığında, daha büyük asal sayıları bulmak için geçen zamanın Tesla K40'da önemli ölçüde düştüğü, ancak diğerlerinde artış olduğu anlaşılmaktadır. Bu durumun oluşmasında ise GPU'nun bellek boyutu, okuma hızı ve çekirdek sayısının fazla olması büyük rol oynamaktadır.

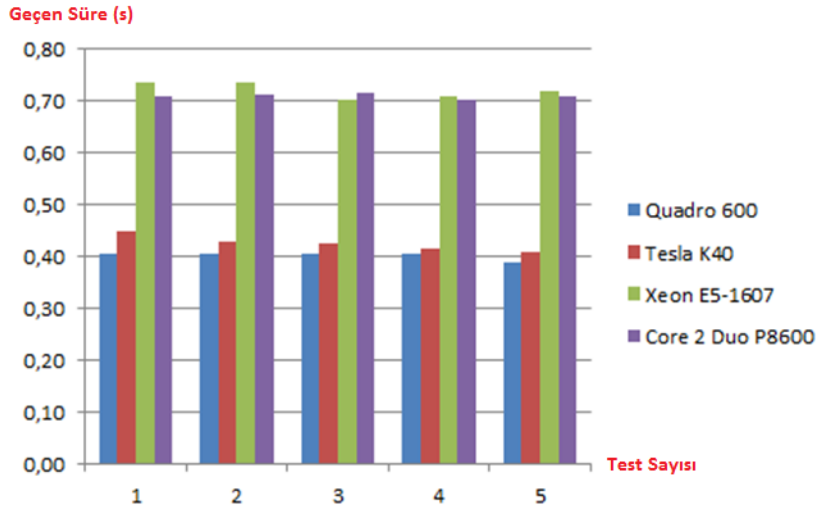
Çizelge 1. 10^7'den küçük asal sayıların bulunması için geçen süre

	Maksimum Sayı	**Geçen Ortalama Süre_(s)**
Tesla K40	10^7	0,43
Quadro 600	10^7	0,40
Xeon E5-1607	10^7	0,72
Core 2 Duo P8600	10^7	0,71

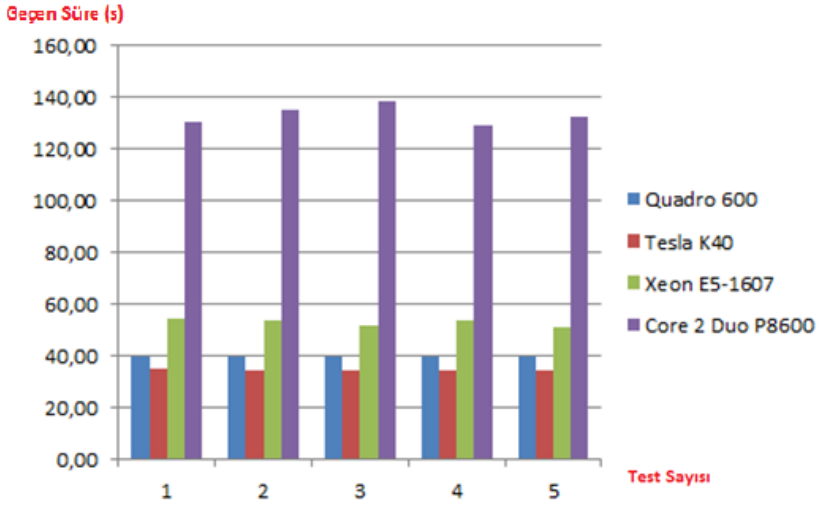
Çizelge 2. 10^9'dan küçük olan asal sayıların bulunması için geçen süre

	Maksimum Sayı	**Geçen Ortalama Süre_(s)**
Tesla K40	10^9	34,68
Quadro 600	10^9	39,78
Xeon E5-1607	10^9	53,09
Core 2 Duo P8600	10^9	133,27

Şekil 3 ve Şekil 4'deki her bir uygulamanın birden fazla çalıştırılması ile elde edilen test sonuçlarına göre performans gelişmelerine baktığımızda, hızlandırılmış hesaplama anlayışı daha belirgin hale gelmektedir. Hesaplanan maksimum asal sayı büyüdükçe Tesla'nın GPU'su aynı geliştirici çabası ile dikkate değer derecede yüksek bir performans sunmaktadır. Sonuçlardan da anlaşılmaktadır ki, GPU'lar hesaplama işlemlerinin hızlandırılması için mantıklı bir seçimdir.



Şekil 3. 10^7'ye kadar olan asal sayılar için performans karşılaştırması



Şekil 4. 10^9 'a kadar olan asal sayılar için performans karşılaştırması

5. Sonuç ve Öneri

Bu araştırmada asal sayıların bulunması için Eratosten Kalburu algoritması konvansiyonel CPU ve GPU üzerinde ayrı ayrı koşturularak bazı hesaplamalar yapıldı. GPU çekirdekleri bize hesaplama sonucunu daha hızlı bir şekilde verdi. Fakat bu uygulamada CUDA ile uyumlu büyük sayı kütüphanesi henüz sağlanmadığından algoritma küçük bir sayı aralığı için ($0\text{-}10^9$) uygulanabildi. Uygulama sonucunda elde edilen başarım dikkate alındığında paralelleştirilen algoritma paralel GPU'lar üzerinde yürütülerek daha yüksek başarım elde edilmesi öngörülebilir.

Kaynaklar

[1] Hendy, M. D., Euclid and the fundamental theorem of arithmetic. Historia Mathematica, 2001; 2:189-191.
[2] McGregor-Dorsey, Zachary S., Methods of primality testing. MIT Undergraduate Journal of Mathematics, 1999; 1:133-141.
[3] Curtis M., Tularam G. A., The importance of numbers and the need to study primes: The prime questions. Journal of Mathematics and Statistics, 2011; 7:262-269.
[4] Hyde D. C., Introduction to the principles of parallel computation. Lewisburg: Bucknell University; 1995.
[5] Cai Y., Wang G. L., Wang H., A high performance crash worthiness simulation system based on GPU. Advances in Engineering Software, 2015; 86:29-38.
[6] Abecassis F., Lavernhe S., Tournier C., Boucard P., Performance evaluation of CUDA programming for 5-axis machining multi-scale simulation. Computers in Industry, 2015; 71:1-9.
[7] Liu H., Tong C., GMP implementation on CUDA – A backward compatible design with performance tuning. Toronto: University of Toronto; 2012.
[8] Nagaoka T., Watanabe S., A GPU-based calculation using the three-dimensional FDTD method for electromagnetic field analysis. In: 32nd Annual International Conference of the IEEE. Buenos Aires; 2010.