Pearson İlinti Katsayısı Hesaplamasının Grafik İşlemci Birimleri Üzerinde Hızlandırılması

Speeding-Up Pearson Correlation Coefficient Calculation on Graphical Processing Units

K. Berker Loğoğlu[1,2], Tuğrul K. Ateş[1,3]

1. TÜBİTAK Uzay Teknolojileri Araştırma Enstitüsü

2. Bilişim Sistemleri Bölümü
Orta Doğu Teknik Üniversitesi

3. Elektrik ve Elektronik Mühendisliği Bölümü
Orta Doğu Teknik Üniversitesi

{berker.logoglu,tugrul.ates}@uzay.tubitak.gov.tr

Özetçe

Veri işleme, çoğulortam, örüntü tanıma ve yapay zeka uygulamalarında, örneklem ilinti katsayısı, işaret benzerliği bulmada sıkça kullanılır. Ayrık işaretler arasındaki ilinti katsayısının en sık rastlanan ölçütü Pearson İlinti Katsayısı'dır. Dev örüntü veritabanlarında benzerlik araması yapmak, sayısal vektörler arasında ilinti katsayısı hesaplanmasının hızlı bir yolunu gerektirir. Bu bildiride, tüketiciye yönelik merkezi işlemci birimleri (CPU) ve grafik işlemci birimleri (GPU) üzerinde Pearson İlinti Katsayısı'nı hesaplatacak paralel ve verimli bir yöntem önerilmektedir. Oldukça fazla sayıda mimari yapı ve geniş değişken aralığında, C++, OpenCL ve CUDA için farklı gerçekleştirimler karşılaştırılmaktadır. Deneysel sonuçlar, karşılaştırılmalı bir biçimde verilmekte ve hem yazılımsal hem de donanımsal bakış açılarıyla incelenmektedir.

Abstract

Sample correlation coefficient is used widely for finding signal similarity in data processing, multimedia, pattern recognition and artificial intelligence applications. Pearson Correlation Coefficient is the most common measure for the correlation coefficient between discrete signals. Similarity search in huge pattern databases require a fast way of calculating the correlation coefficient between numerical vectors. In this paper, a parallel and efficient way of calculating Pearson Correlation Coefficient on commodity central processing units (CPUs) and graphical processing units (GPUs) is proposed. Different implementations for C++, OpenCL and CUDA are compared over a vast number of architectures and through a wide parameter range. Experimental results are given in a comparative manner and investigated in both software and hardware perspectives.

1. Giriş

Tüketiciye yönelik grafik kartları, aslen elektronik oyunlar ve üç boyutlu grafik çizimleri için geliştirilmiş donanımlar olmalarına rağmen, paralel veri işleme yeteneklerinden ötürü son yıllarda grafik işlemeden bağımsız işlerde de kullanılmaya başlanmıştır. GPU tasarımlarının, paralel ve yoğun hesaplama gerektiren, düzenli hafıza erişimi olan uygulamalar için uygun olmasından ötürü, *grafik işleme birimi üzerinde genel amaçlı programlama* (GPGPU) giderek yaygınlaşan bir yordam haline gelmiştir.

GPU programlamanın kendisi ve uygulamaları üzerine birçok çalışma, son birkaç yıllık kısa zamanda literatürde yer edinmişlerdir. Owens ve Luebke [1] çalışmalarında, GPGPU'nun altında yatan teknik güdüleri, donanım ve yazılımdaki gelişmelerin bu konuya olan ilgiyi nasıl ortaya çıkardığını detaylı bir şekilde anlatırlar. Farklı bir çalışmalarında [2] ise yine GPGPU uygulamalarının donanım ve programlama modellerinin geçmişinden bahsedip, oyun içi fiziği ve hesaplamalı biyofizik gibi birçok farklı uygulamada elde ettikleri performans artışlarını sunarlar.

GPU programlama sayesinde, bilgisayarla görü gibi grafik çizim bakış açısına yakın alanlarda yapılan çalışmaların [3, 4, 5] yanı sıra, grafik işleme ile çok ilgili olmayan geniş bir alan yelpazesinde uygulamalar geliştirilmiştir. Che ve Boyer yaptıkları çalışmada [6], k-ortalama kümelemeyi de içeren birçok farklı uygulamayı CUDA ile gerçekleştirip performans analizi sunarlar. Ryoo, çalışmasında [7] CUDA ile H-264 video kodlama ve *sonlu eleman yöntemini* (FEM) de içeren 13 farklı uygulamanın hızlandırabildiğini gösterir. Nottingham ve Irwin [8] OpenCL'in ağ güvenliği için, ağ paketlerinin paralel olarak işlenip sınıflandırılması amacı ile kullanılmasını önerirler. Zhang [9] ise bilgisayarlı tomografide imge geriçatımı ve tanıma işlemlerini OpenCL ile GPU üzerinde gerçekleştirir ve önemli miktarda hesap hızı artışı elde eder.

OpenCL'in taşınabilir sistemlerde de çalışabilir olması, onu taşınabilir yazılım geliştirenler için önemli bir altyapı haline getirmektedir. Taşınabilir aygıtlar, limitli ve göreceli

olarak az miktarda merkezi işlemci gücüne sahiptir. Bu sebeple, her geçen gün daha çok aygıtta kullanılmaya başlanan taşınabilir GPU'ların da genel amaçlı hesaplamalarda kullanılması önemlidir. Leskela [10] ve Pulli [11], OpenCL'in birçok farklı imge işleme uygulamalarında kullanılması önerirler. Leskela, çalışmasında, imge uygulamalarını OpenCL ile CPU dan GPU'ya taşıyarak taşınabilir sistemlerde hem performans artışı hem de enerji verimliliği elde etmektedir.

Bu bildiride, Pearson İlinti Katsayısı'nın tüketiciye yönelik GPU'lar üzerinde paralel ve verimli bir şekilde hesaplanmasını olanaklı kılacak bir algoritma önerilmektedir. Bildirinin ikinci bölümünde GPU mimarileri üzerinde paralelleşmeyi sağlayacak araç ve bakış açıları anlatılmaktadır. Üçüncü bölümde Pearson İlinti Katsayısı eşitliği ve bu eşitliğin paralelliğe uygun gösterimleri verilmekte, ilinti hesabının GPU üzerinde gerçekleştiriminin farklı yolları ele alınmaktadır. Dördüncü bölüm çalışma sırasında yapılan deneyler ve elde edilen sonuçları içermektedir. Son bölümde ise çalışmanın kısa bir özeti ve elde edilen çıktılara yönelik fikir çalışması bulunmaktadır.

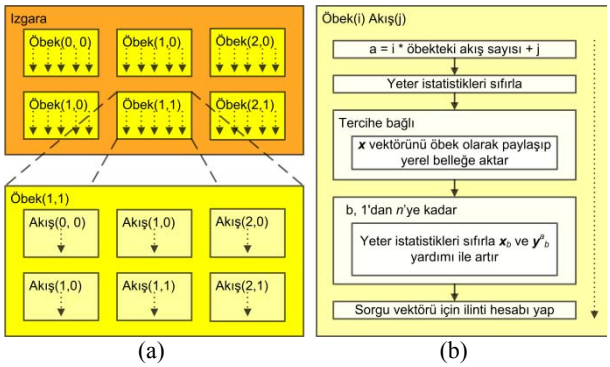
2. Grafik İşlemci Birimi

Grafik işlemci birimleri (GPU), geometri bilgisini imge bilgisine çevirmek için özelleşmiş, tüketiciye yönelik bilgisayarların görüntü kartlarında bulunan, işlemci mimarilerdir. Grafik işlemciler üzerinde çizilecek sahneler, geometrik gösterimlerinden önce çizilecek poligonlara, oradan poligon parçacıklarına ve sonunda ekranda gösterilecek piksellere dönüştürülür. Bu çizim sürecine uygun olarak grafik işlemciler *akış işlemcileri* olarak tasarlanmıştır.

Grafik işlemciler, akış işleyici özellikleri nedeniyle geleneksel von Neumann mimarilerin *tek-iş-tek-veri* (SISD), yeteneğinin yanında *tek-iş-çok-veri* (SIMD) ve *çok-iş-çok-veri* (MIMD) yeteneklerine de sahiptirler. GPU'ların çizim işlemlerini özelleştirmek üzere, 2000'lerden itibaren programlanabilir olarak tasarlanmaları ile birlikte, bu mimarilerin hesaplama yetenekleri, çizim dışı alanlarda, genel programlama işleri için kullanılmaya başlanmıştır. O dönemlerde, GPU ile çalışacak sistemlerin kapsamı ne olursa olsun, bu sistemlerin, grafik çizime yönelik tasarlanmış diller (GLSL, Cg gibi) ve dolayısıyla grafik çizim bakış açısı ile geliştirilmeleri gerekmekteydi.

Her bilgisayarda bulunabilecek bu paralel akış işlemciler üzerinde genel amaçlı programlamanın yaygınlaşması üzerine, NVIDIA, 2006'da *Hesap Birleşik Aygıt Mimarisi* (CUDA) adını verdiği donanım ve yazılım altyapısını programcılara sunmuştur. CUDA sayesinde, programcılar C++ benzeri bir dil ve GPU programlamadan donanımın imge çizmeye yönelik oluşunun soyutlanması sayesinde göreli olarak daha kolaylıkla hem CPU hem de GPU üzerinde çalışacak birleşik sistemler geliştirmeye başlamışlardır. Nickolls [12], CUDA hakkındaki çalışmasında bu altyapının başarısını sorgulamış ve neden kısa sürede yaygınlaştığının cevabını aramıştır.

CUDA'nın ortaya çıkmasından yaklaşık bir buçuk yıl sonra Apple yeni bir standart için ön ayak olmuştur. Aralık 2008'de, OpenCL 1.0 belirtimi [13], açık kaynak kodlu *uygulama geliştirme arayüzü* (API) tasarlamak üzerine yoğunlaşmış, aynı zamanda OpenGL'in de geliştirilmesini yöneten sınai bir kurul olan Khronos tarafından yayınlanmıştır. Bu girişim bütün büyük CPU ve GPU üreticilerinin yanında taşınabilir telefon üreticileri tarafından da desteklenmektedir [14].



(a) (b)

Şekil 1: (a) Hesap Birleşik Aygıt Mimarisi. (b) Paralel Pearson İlinti Katsayısı hesabı.

OpenCL, C tabanlı bir dil olarak tasarlanmıştır. Programcılara, çok çekirdekli merkezi işlemci birimleri, grafik işlemci birimleri, *hücre geniş bant motoru* (CBE) ve *sayısal sinyal işleyiciler* (DSP) gibi birçok farklı yapıdaki donanımda çalışabilecek uygulamalar geliştirebilecekleri, taşınabilir ve verimli bir programlama ortamı sunmayı hedeflemektedir. OpenCL, bu yolla programcıların, hızla gelişen mimari gelişmelerden etkilenmeyen, çok çekirdekli yapılara uygun uygulamalar geliştirmelerine yardımcı olmaktadır.

Hesap birleşik sistemler kabaca Şekil 1.a'da verilen işlem akış ağacına sahiptir [15]. Sistemdeki her hesaplama cihazı (CPU ya da GPU) bir *akış ızgarası* ile temsil edilir. Bu cihazlarda yapılacak hesaplama işleri *öbeklere* ayrılır ve her öbeğin bir çoğul-işlemci birimine atanması sağlanır. Çoğul-işlemcilerin görevi kendilerine verilen öbekteki işleri aynı anda akışlarda çalıştırarak sonuçlandırmaktır. Bu iki seviye bölme işlemi yazılımsal olarak bir, iki ya da üç boyutlu indislerle yapılabilmektedir.

Birleşik mimaride, tüm ızgara üzerindeki işlerin erişebileceği evrensel bir bellek olduğu varsayılır. GPU mimarilerde, akışların evrensel belleğe ardışık erişimlerini tek seferde yapma yetenekleri vardır [15]. Bu evrensel bellek cihaz üzerindeki *temel rastgele erişim* belleğine (RAM) denk düşer. Aynı zamanda çoğul-işlemciler üzerinde, her öbeğe özel, yazmaç hızına yakın hızlarda olan bir *paylaşımlı bellek* bulunur. Paylaşımlı bellek rastgele erişim sağlar ve sadece öbek içindeki akışlar tarafından paylaşılabilir. GPU mimarilerde akışlar, paylaşımlı bellek erişimlerini, ardışık olmasa bile belli kurallar içinde, aynı anda yapabilirler [15]. Son olarak, her akışın kendine ait yazmaçları bulunur. Bu yazmaçlar, temel sayısal veri tipleri, GPU mimarilerde bulunan vektörler ve CPU mimarilerde bulunan *akışkan tek-iş-çok-veri eklentilerine* (SSE) denk düşer.

3. Pearson İlinti Katsayısının Hesaplanması

Uzunlukları n olan iki ayrık işaret, $\boldsymbol{x}$ ve $\boldsymbol{y}$, arasındaki ilinti (1) ile ölçülebilir. Eşitlikte μ ile ifade edilen değerler işaretlerin örneklem ortalamalarını ifade etmektedir. Pearson ilintisi, işaretlerin ortak değişintisinin işaret standart sapmalarına bölünerek elde edilen [-1, 1] aralığındaki katsayı değerini bulur.

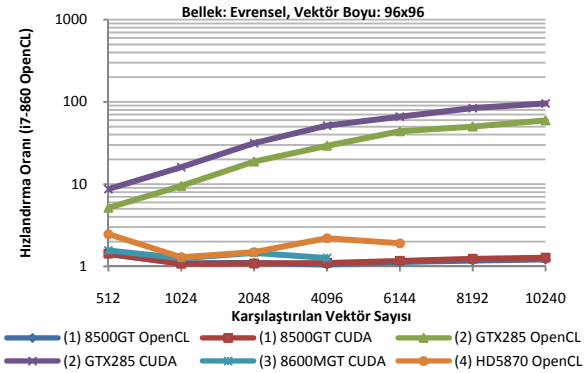
$$\begin{aligned} \rho_{x,y} &= cov(\boldsymbol{x},\boldsymbol{y})/\sigma_x\sigma_y \\ &= \frac{\sum_{i=0}^{n}(\boldsymbol{x}_i-\mu_x)(\boldsymbol{y}_i-\mu_y)}{\sqrt{\sum_{i=0}^{n}(\boldsymbol{x}_i-\mu_x)^2}\sqrt{\sum_{i=0}^{n}(\boldsymbol{y}_i-\mu_y)^2}} \end{aligned} \qquad (1)$$

Tablo 1: Deneylerde kullanılan bilgisayarların CPU, GPU ve yazılım bilgileri.

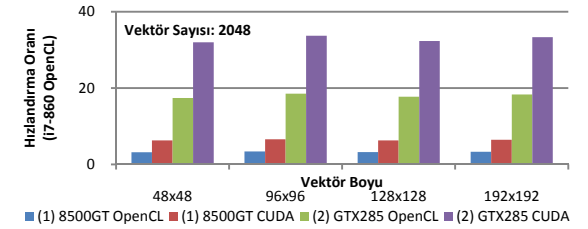
#	Merkezi İşlemci	Çk.	Hız	Bellek
1	Intel Pentium 4	1 (2HT)	3.20GHz	2GB
2	Intel Core 2 Quad Q9400	4	2.66GHz	4GB
3	Intel Mobile Core 2 Duo T7300	2	2.00GHz	2GB
4	Intel Core i7-860	4 (8HT)	2.80GHz	2.8GB

#	Grafik İşlemci	Çk.	Hız	Bellek	Bel. Hızı
1	NVIDIA GeForce 8500 GT	16	450MHz	512MB	800MHz
2	NVIDIA GeForce GTX 285	240	648MHz	1024MB	2484MHz
3	NVIDIA GeForce 8600M GT	32	475MHz	256MB	800MHz
4	ATI Radeon HD 5870	320	850MHz	1024MB	1200MHz

#	İşletim Sistemi	Ekran Kartı Sürücüsü
1	OpenSUSE Linux 11.1 32bit	190.29 CL, 190.53 CUDA
2	Windows XP Professional SP3 32bit	190.39
3	Windows Vista Home SP2 32bit	195.38
4	Windows 7 Professional 64bit	9.12



Şekil 3: Yalnızca evrensel bellekli GPU gerçekleştirimlerinin i-7 860 OpenCL'e göre hızlandırmaları oranları.

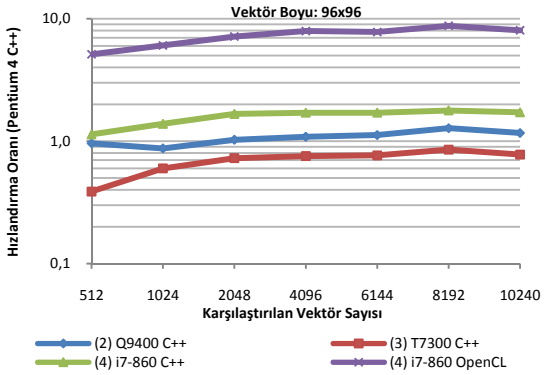


Şekil 5: GPU CUDA ve GPU OpenCl gerçekleştirimlerinin karşılaştırılması.

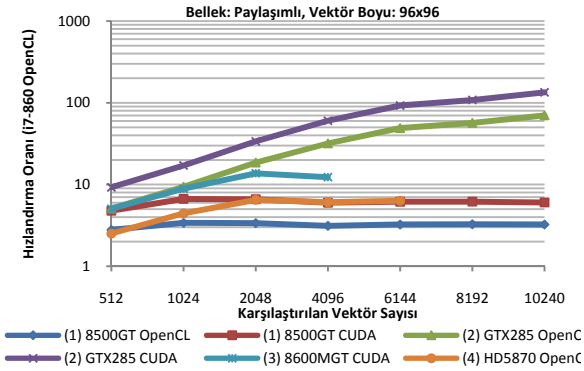
Pearson İlinti Katsayısı hesabının gerçekleştiriminde (2) ile verilen eşdeğer ifadeyi kullanmak daha faydalıdır. Bu ifadede ilinti katsayısı hesabı için gerekli olan yeter istatistiklerin hepsi ($\sum \boldsymbol{x}_i$, $\sum \boldsymbol{y}_i$, $\sum {\boldsymbol{x}_i}^2$, $\sum {\boldsymbol{y}_i}^2$ ve $\sum \boldsymbol{x}_i \boldsymbol{y}_i$) tek döngü içinde hesaplanabileceği için, hem daha hızlı hem de donanımla iletişimi daha tahmin edilir hesaplamalar elde edilir.

$$\rho_{x,y} = \frac{n\sum_{i=0}^{n} \boldsymbol{x}_i \boldsymbol{y}_i - \sum_{i=0}^{n} \boldsymbol{x}_i \sum_{i=0}^{n} \boldsymbol{y}_i}{\sqrt{n\sum_{i=0}^{n} {\boldsymbol{x}_i}^2 - \left(\sum_{i=0}^{n} \boldsymbol{x}_i\right)^2}\sqrt{n\sum_{i=0}^{n} {\boldsymbol{y}_i}^2 - \left(\sum_{i=0}^{n} \boldsymbol{y}_i\right)^2}} \quad (2)$$

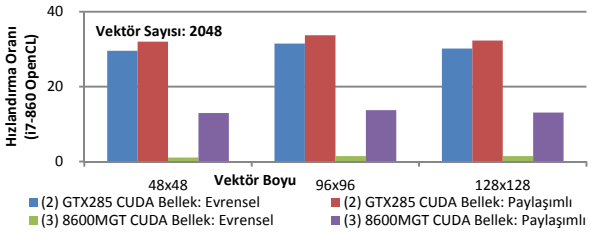
Devasa boyutlarda örüntü veritabanlarında benzerlik araması yapacak sistemlerin ölçeklendirilebilirliğini sağlamak için Pearson İlinti Katsayı'nın paralel olarak hesaplanması yararlı olacaktır. Bu paralelliğin işaretler içindeki örneklemler arasında uygulanması, her örneklemi bir diğerine bağlı kılan toplama işlemleri yüzünden sistemin verimsiz olmasına neden olacaktır. Paralelliğin, karşılaştırılacak örüntüler arasında yapılması, hem paralel çalışabilecek donanımlardan en üst



Şekil 2: CPU gerçekleştirimlerinin 96×96 vektör boyunda verilerle Pentium 4'e göre hızlandırmaları oranları.



Şekil 4: Paylaşımlı bellek kullanan GPU gerçekleştirimlerinin i-7 860 OpenCL'e göre hızlandırmaları oranları.



Şekil 6: Paylaşımlı bellek kullanan ve kullanmayan GPU gerçekleştirimlerinin karşılaştırılması.

düzey faydayı elde etmeye yarayacak, hem de daha sonraki aşamalarda kullanılacak sıralama yöntemlerine paralellik zemini kazandıracaktır.

Bildirimizde ele aldığımız senaryo, bir örüntü vektörünün çok sayıda başka vektörle arasındaki ilinti katsayılarının hesaplanmasıdır. Geleneksel yaklaşım, bir döngü içinde bütün karşılaştırılacak vektörleri elden geçirmek olacaktır. Sorunu, hesap birleşik mimarilere uyarlamak her ilinti katsayısı hesabını ayrı bir iş olarak tanımlamak ve kullanılacak cihazın ızgarasına iletmek olacaktır. Önceki bölümde ifade edildiği gibi, GPU mimarilerde ardışık ya da yakın bellek erişimleri paralel yapılabilir. Dolayısıyla, matris sütunları olarak ifade edilen sayısal vektörleri, belleklere *satır temelli* dökmek en iyi hız başarımı elde etmeyi sağlayacaktır.

Öbeklere verilen her bir iş, kaynak vektör ile işi çalıştıracak akışa özel sorgu vektörü üzerinde çalışacaktır. İlinti hesabındaki yeter istatistiklerden olan $\sum \boldsymbol{x}_i$ ve $\sum {\boldsymbol{x}_i}^2$ her iş için ortak hesaplanacak ara değerlerdendir. Bu ara değerlerin önceden hesaplanıp saklanmasının mümkün olmadığı durumlarda, her akışın, bellekten $\boldsymbol{x}_i$ değerini alması

gerekir. Evrensel belleğe erişimin maliyetli olacağı kabulünden hareketle, bu değerin her akış için ayrıca erişilmesi yerine öbek tarafından topluca ve tek kez paylaşımlı belleğe alınması öneriyoruz.

Pearson İlinti Katsayısı hesabının paralel hesaplanmasına yönelik önerdiğimiz yöntemin akış şeması Şekil 1.b'de görülebilir. Bu akış şeması ızgaraya atılacak işlerin yalnızca birini temsil etmektedir. Bütün akışlar tamamlandığında, bütün ilinti katsayıları da bulunmuş olacaktır.

4. Deneyler

Pearson İlinti Katsayısı için önerilen paralelleştirme yöntemi Tablo 1'de verilen dört farklı bilgisayardaki CPU ve GPU üzerinde C++, OpenCL ve CUDA gerçekleştirimleriyle ile denenmiştir. Deney düzeneği, farklı boyutlarda alınan bir sayısal vektörün değişen sayıda başka vektör ile kıyaslanması ve sonuçların referans C++ gerçekleştirim ile karşılaştırılmasından oluşur. Vektör boyutları 48×48, 96×96, 128×128 ve 192×192 olarak belirlenmiş, her vektör boyu için 512 ile 20480 arasında değişen sayıda ve bellek miktarlarının elverdiği ölçüde farklı vektör ile karşılaştırma yapılmıştır. Vektörler sayısal olarak 32bit kayan noktalı gösterim ile ifade edilmişlerdir. Cihazlar arası bellek transferlerinin süreleri karşılaştırmalarda dikkate alınmamıştır.

C++ gerçekleştirimler yalnızca CPU üzerinde tek çekirdek kullanımında ve hız için eniyilenmiş olarak çalışmaktadırlar. Bu mimarilerdeki teorik en yüksek hızlar, tek çekirdek hızlarıyla, sistemdeki CPU çekirdeği sayısının çarpılmasıyla ile bulunabilir. C++ gerçekleştirim SSE ve benzeri teknolojiler es geçilerek, önbellek farkındalığında, Linux altında *g++* ve Windows altında Microsoft 2008 derleyici araç takımı ile derlenmiştir.

OpenCL ve CUDA gerçekleştirimler, tamamen evrensel bellek kullanan ve evrensel belleğin yanında paylaşımlı bellek de kullanan olarak iki farklı şekilde hazırlanıp sınanmıştır. Bu gerçekleştirimlerde her test için en hızlı sonucu veren çalışma öbeği boyu seçilmiştir.

İlk deneyde C++ gerçekleştirimler ile en yüksek çekirdekli işlemci üzerindeki OpenCL gerçekleştirim karşılaştırılmıştır. Şekil 2'de görüldüğü üzere Intel Core i7-860 OpenCL gerçekleştirimi, sisteminin *aşırı-akış* (HT) özelliklerinden de faydalanarak, diğer tüm CPU gerçekleştirimlere göre en az dört katın üzerinde hızlandırma sağlamıştır. Bu sebeple, bu gerçekleştirim, GPU üzerindeki diğer gerçekleştirimler için sonraki deneylerde referans kabul edilmiştir.

GPU üzerinde, sadece evrensel bellek kullanan gerçekleştirimlerin hız kazanımları Şekil 3'te, paylaşımlı bellek yöntemi kullananların hız kazanımları ise Şekil 4'te gösterilmiştir. Bu iki deneyden çıkarılan sonuç, GPU üzerinde paralel ilinti katsayısı işini tüm işlem kaynaklarından faydalanıp en verimli seviyede yapabilmek için karşılaştırılacak vektör sayısının belli bir sayının üzerinde olması gerekliliğidir. Elde edilen bir diğer sonuç ise ATI Radeon HD 5870'in en fazla çekirdekli GPU olarak görünmesine rağmen, en hızlı hesaplamayı yapmamış olmasıdır. Bu sonuç [9]'da da gözlemlenmiş olup, donanımın OpenCL desteğinin yeni olmasına bağlanabilir.

Hem OpenCL hem de CUDA çalıştırabilen iki mimarinin karşılaştırılması Şekil 5'te verilmektedir. Farklı uygulamalarda olduğu gibi [9] ilinti katsayısı hesabında da CUDA ile OpenCL'e göre daha fazla kazanım elde edilmiştir. Bu sonuç, CUDA derleyicisi ve sürücülerinin daha olgun olmasına bağlanabilir. Paylaşımlı bellek kullanmanın getirdiği hız kazanımı en açık halde Şekil 6'de görülebilir. Paylaşımlı bellek yönteminin ek getirisinin GPU ana bellek hızı ile ters bağıntılı olduğu gözlemlenmiştir.

5. Sonuçlar

Bu çalışmada dev örüntü veritabanlarında, hızlı hesaplanması büyük öneme sahip olan Pearson İlinti Katsayısı'nı GPU'lar üzerinde paralelleştirilerek daha hızlı hale getiren bir yöntemden bahsedilmiştir. GPU'lar üzerinde hesap yeteneği bütünüyle kullanılan çok çekirdekli CPU'lardan bile 100 kata kadar hızlı sonuçlar elde edilmiştir. Denenen hesap birleşik mimari altyapıları arasında CUDA'nın, OpenCL'e göre ilinti hesabı için daha uygun olduğu görülmüştür. Ayrıca, özellikle GPU ana bellek hızının düşük olduğu donanımlarda olmak üzere, paylaşımlı bellek kullanımının önemi ortaya konmuştur. Bu çalışmadaki karşılaştırma yöntemi ile birlikte kullanılmak üzere, GPU üzerinde gerçekleştirilecek sıralama ve getirme yöntemleri sayesinde verimli arama sistemlerinin oluşturulması planlanmaktadır.

6. Kaynakça

[1] Owens, J.D.; Luebke, D.; Govindaraju, N.; Harris, M.; Krüger, J.; Lefohn, A.E.; Purcell, T.J, "A Survey of General-Purpose Computation on Graphics Hardware," *Computer Graphics Forum*, vol.26, no.1, pp. 80-113, 2007

[2] Owens, J.D.; Houston, M.; Luebke, D.; Green, S.; Stone, J.E.; Phillips, J.C., "GPU Computing," *Proceedings of the IEEE* , vol.96, no.5, pp.879-899, May 2008

[3] Zach, C., Bischof, H., Karner, K.: "Hierarchical disparity estimation with programmable 3D hardware," *WSCG (International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision). Short Communications*, pp. 275–282, Plzen, Slowakei (2004)

[4] Woetzel, J., Koch, R.: "Real-Time Multi-Stereo Depth Estimation on GPU with Approximative Discontinuity Handling,". *European Conf. on Visual Media Production* (2004)

[5] R. Yang, M. Pollefeys, H. Yang, and G. Welch. "A Unified Approach to Real-Time, Multi-Resolution, Multi-Baseline 2D View Synthesis and 3D Depth Estimation Using Commodity Graphics Hardware,". *International Journal of Image and Graphics (IJIG)*, 4(4):627–651, 2004.

[6] Shuai Che; Michael Boyer; Jiayuan Meng; David Tarjan; Jeremy W. Sheaffer; Kevin Skadron; "A performance study of general-purpose applications on graphics processors using CUDA," *Journal of Parallel and Distributed Computing*, vol.68, pp. 1370-1380, October 2008

[7] Ryoo, S.; Rodrigues C. I.; Baghsorkhi, S. S.; Stone, S. S.; Kirk, D. B.; Hwu, W. W, "Optimization principles and application performance evaluation of a multithreaded GPU using CUDA," In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, February, 2008.

[8] Nottingham, A. and Irwin, B. 2009. "*GPU packet classification using OpenCL: a consideration of viable classification methods,*". *Proceedings of the 2009 Annual Research Conference of the South African institute of Computer Scientists and Information Technologists*, October, 2009.

[9] Zhang, Wenyu; Zhang, Li; Sun, Shangmin; Xing, Yuxiang; Wang, Yajie; Zheng, Juan, "A preliminary study of OpenCL for accelerating CT reconstruction and image recognition," *Nuclear Science Symposium Conference Record (NSS/MIC), 2009 IEEE* , vol., no., pp.4059-4063, Oct. 24 2009-Nov. 1 2009

[10] Leskela, J.; Nikula, J.; Salmela, M., "OpenCL embedded profile prototype in mobile device," *Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on* , vol., no., pp.279-284, 7-9 Oct. 2009

[11] Pulli, K., "Mobile visual computing, " *Systems, Architectures, Modeling, and Simulation, 2009. SAMOS '09. International Symposium on* , vol., no., pp.i-i, 20-23 July 2009.

[12] Nickolls, J.; Buck, I.; Garland, M.; Skadron, K.; "Scalable parallel programming with CUDA," *ACM SIGGRAPH 2008 Classes*, August, 2008

[13] Khronos OpenCLWorking Group. "*OpenCL Specification, v1.0. Technical Report*", Khronos Group. http://www.khronos.org.

[14] A. Munshi, "OpenCL: Parallel computing on the GPU and CPU," *presentation at SIGGRAPH 2008*, http://s08.idav.ucdavis.edu/munshi-opencl.pdf

[15] NVIDIA. "NVIDIA CUDA Programming Guide 2.3 edition," 2009.