BURSA TEKNİK ÜNİVERSİTESİ

Bilgisayar Mühendisliği

Paralel Programlama

Ahmet Korkmaz - 21360859072

Ahmet Talha Geçgelen - 22360859024

Paralel Poligon Konvekslik Tespiti (Aparapi ile GPU Hızlandırması)

1. Giriş

Bu proje, bilgisayar bilimleri ve geometrik hesaplama alanında önemli bir yere sahip olan **poligon konvekslik tespitini** ele almaktadır. Özellikle **sıralı olarak verilen (x, y) koordinatlarından oluşan bir poligonun dışbükey (konveks) mi yoksa içbükey (konkav) mi olduğunu belirlemek**, projenin temel amacını oluşturmaktadır. Bu tespit, bilgisayar grafikleri, robotik, coğrafi bilgi sistemleri (CBS) ve çarpışma tespiti gibi çeşitli uygulama alanlarında kritik bir rol oynar. Örneğin, bir oyun motorunda iki cismin çarpışıp çarpışmadığını hızlıca anlamak, robotların çevredeki engellerden kaçınmasını sağlamak veya karmaşık haritalarda belirli bölgelerin özelliklerini analiz etmek için konvekslik tespiti vazgeçilmezdir.

Günümüzün giderek artan veri hacmi ve karmaşık problemleriyle birlikte, geleneksel ardışık (sequential) hesaplama yöntemleri genellikle yetersiz kalmaktadır. Bu noktada **paralel programlama**, performansı artırmak ve daha büyük veri setlerini daha verimli işlemek için hayati bir çözüm sunar. Bu proje kapsamında, poligon konvekslik tespiti problemini paralel hale getirerek, **grafik işlem birimlerinin (GPU) yüksek paralel işlem gücünden** faydalanılması amaçlanmıştır. Bu amaca ulaşmak için, Java kodunu doğrudan GPU üzerinde çalıştırabilen ve OpenCL standardını temel alan **Aparapi kütüphanesi** kullanılmıştır. Böylece, düşük seviyeli GPU programlama dillerinin karmaşıklığına girmeden, GPU'nun SIMD (Single Instruction, Multiple Data) mimarisinin sunduğu avantajlardan yararlanılarak, performanslı ve ölçeklenebilir bir çözüm geliştirilmesi hedeflenmiştir.

2. Teori ve Algoritma

2.1. Konveks ve Konkav Poligon Kavramları

Geometrik bir şekil olan **poligon (çokgen)**, bir düzlemde sonlu sayıda doğru parçasının birbirini takip ederek kapalı bir şekil oluşturması ile meydana gelir. Bu doğru parçalarına **kenar**, kenarların birleştiği noktalara ise **köşe** denir. Poligonlar, iç yapılarına göre başlıca iki kategoriye ayrılır: konveks ve konkav poligonlar.

- **Konveks (Dışbükey) Poligon:** Bir poligonun konveks olması için iki temel koşuldan biri veya her ikisi birden sağlanmalıdır:
    - Poligonun herhangi iki noktası arasında çizilen tüm doğru parçaları, tamamen poligonun **içinde** kalmalıdır.
    - Poligonun tüm **iç açıları 180 dereceden küçük veya eşit** olmalıdır. Yani, poligonun kenarları boyunca ilerlerken, hiçbir köşede "içe doğru bir çökme" (reflex angle) yaşanmaz.
- **Konkav (İçbükey) Poligon:** Bir poligonun konkav olması için yeterli koşul şudur:
    - Poligonun en az bir iç açısı **180 dereceden büyüktür**. Bu durum, poligonun çevresi boyunca ilerlerken en az bir köşede "içe doğru bir çökme" olduğu anlamına gelir.

- Alternatif olarak, poligonun herhangi iki noktası arasında çizilen bir doğru parçası, poligonun **dışına taşabilir**.

Bu projenin temel amacı, verilen bir poligonun bu tanımlara göre konveks mi yoksa konkav mı olduğunu otomatik olarak tespit etmektir.

2.2. Konvekslik Tespiti Algoritması: Çapraz Çarpım (Cross Product) Yöntemi

Sıralı köşeleri (yani bir poligonun kenarlarının sırasıyla birbirine bağlandığı varsayılan) verilen bir poligonun konveksliğini tespit etmenin etkili bir yolu, her bir köşede yapılan dönüşün yönünü kontrol etmektir. Bir poligonun konveks olması için, çevresi boyunca ilerlerken her köşede yapılan dönüş yönünün (örneğin hep sola veya hep sağa) **tutarlı** olması gerekir. Eğer dönüş yönü değişirse, bu bir içbükey köşe (konkavlık) olduğunu gösterir.

Bu dönüş yönünü belirlemek için **iki boyutlu (2D) çapraz çarpım (cross product)** kavramı kullanılır. Geometride, Pa, Pb ve Pc gibi üç nokta verildiğinde, PaPb vektörü ile PaPc vektörünün 2D çapraz çarpımı, Pc'nin PaPb doğrusuna göre hangi tarafta kaldığını gösteren bir değer üretir.

Çapraz Çarpım Formülü:

Verilen üç nokta Pa=(xa,ya), Pb=(xb,yb) ve Pc=(xc,yc) için çapraz çarpım değeri şu şekilde hesaplanır:

$$(xb-xa)\cdot(yc-ya)-(yb-ya)\cdot(xc-xa)$$

Sonucun Yorumlanması:

- **Pozitif Değer (> 0):** Dönüş, Pa→Pb→Pc sırasında **saat yönünün tersinedir** (sol dönüş).
- **Negatif Değer (< 0):** Dönüş, Pa→Pb→Pc sırasında **saat yönündedir** (sağ dönüş).
- **Sıfır Değer (= 0):** Üç nokta **doğrusaldır** (aynı çizgi üzerindedir).

Algoritma Adımları:

Poligonun köşeleri, sırasıyla $P0,P1,\ldots,PN-1$ olarak alınır.

Poligonun her köşesi için, ardışık üç nokta $(Pi,Pi+1,Pi+2)$ alınarak çapraz çarpım hesaplanır. Köşelerin döngüsel olduğunu unutmamak önemlidir; yani $PN-1$'den sonra P0, P0'dan sonra P1 gelir. Bu nedenle indeksler modüler aritmetik kullanılarak hesaplanır: Pi, $P(i+1)(modN)$, $P(i+2)(modN)$.

Hesaplanan ilk çapraz çarpım değerinin işareti kaydedilir (pozitif veya negatif).

Daha sonra hesaplanan tüm çapraz çarpım değerlerinin işareti, kaydedilen ilk işaretle **tutarlı** olmalıdır.

Eğer ilk işaret pozitifse, diğer tüm sonuçlar da ya pozitif olmalı ya da sıfır olmalıdır.

Eğer ilk işaret negatifse, diğer tüm sonuçlar da ya negatif olmalı ya da sıfır olmalıdır.

Eğer bu kontrol sırasında işaret tutarsızlığı bulunursa (örneğin bir sol dönüşten sonra bir sağ dönüş geliyorsa), poligon **konkavdır** ve algoritma hemen false döndürebilir.

Eğer tüm köşelerdeki dönüş yönleri tutarlı bulunursa (veya tüm noktalar doğrusalsa), poligon **konvekstir** ve algoritma `true` döndürür.

Bu algoritma, her bir köşe için yapılan çapraz çarpım hesaplamalarının birbirinden bağımsız olması nedeniyle, paralel programlama için oldukça uygun bir yapı sunar. Bir sonraki bölümde, bu algoritmanın GPU üzerinde Aparapi kullanılarak nasıl paralelleştirildiğini detaylandıracağız.

3. Paralel Uygulama Mimarisi ve Aparapi Kullanımı

3.1. Paralel Hesaplama İhtiyacı

Önceki bölümde detaylandırıldığı üzere, bir poligonun konveksliğini tespit etmek için her bir köşe üçlüsü için **çapraz çarpım (cross product)** hesaplaması yapılması gerekmektedir. Köşe sayısı arttıkça, yapılması gereken bu hesaplama sayısı da doğru orantılı olarak artar. Büyük veri setleri veya gerçek zamanlı uygulamalar (oyunlar, simülasyonlar) söz konusu olduğunda, bu hesaplamaların ardışık (sequential) olarak yapılması ciddi performans darboğazları yaratabilir. Her bir çapraz çarpım hesaplaması, diğerlerinden **bağımsız** olduğu için, bu problem paralel işlemeye son derece uygundur. İş yükünü birden fazla işlem birimine dağıtarak, çok daha hızlı sonuçlar elde etmek mümkündür.

3.2. Aparapi Kütüphanesi ve Seçilme Nedenleri

Bu projede paralel hesaplama yeteneklerinden faydalanmak ve özellikle GPU'nun gücünü kullanmak amacıyla **Aparapi** kütüphanesi tercih edilmiştir.

- **Aparapi Nedir?** Aparapi, Java kodunu doğrudan GPU üzerinde çalıştırmayı sağlayan, açık kaynaklı bir kütüphanedir. Geliştiricilerin, GPU programlama için genellikle kullanılan C++, CUDA veya OpenCL gibi düşük seviyeli dilleri öğrenme zorunluluğunu ortadan kaldırır. Aparapi, derleme zamanında Java bytecode'unu analiz eder ve uygun OpenCL koduna dönüştürerek GPU'da çalıştırır. Eğer bir GPU veya uygun OpenCL sürücüsü bulunamazsa, otomatik olarak CPU üzerindeki paralel bir iş parçacığı havuzuna (Java Thread Pool) düşüş (fallback) yaparak kodun çalışmaya devam etmesini sağlar.

- **Neden Aparapi Seçildi?**
    - **Java Tabanlılık:** Proje Java dilinde geliştirildiğinden, Aparapi Java ekosistemine entegre olmasıyla büyük kolaylık sağlamıştır.
    - **GPU Hızlandırması:** Projenin temel hedeflerinden biri olan GPU hızlandırmasını, nispeten daha yüksek bir soyutlama seviyesinde (OpenCL/CUDA'ya göre) elde etme imkanı sunmuştur. Bu sayede, sahip olduğumuz güçlü ekran kartının işlem gücünden faydalanılmıştır.
    - **Paralel Hesaplama Felsefesi:** Çapraz çarpım kontrolü gibi SIMD (Single Instruction, Multiple Data) mimarisine uygun, bağımsız ve tekrarlayan hesaplamalar için Aparapi'nin tasarımı birebir örtüşmektedir.

3.3. Paralelleştirme Stratejisi

Poligon konvekslik tespiti için kullanılan çapraz çarpım yönteminin paralelleştirilmesi, Aparapi'nin `Kernel` sınıfı etrafında şekillenmiştir:

- **İş Bölümü:**
    - Poligon, sıralı N adet köşeye (P0,P1,…,PN−1) sahiptir.
    - Her bir köşedeki dönüşü kontrol etmek için üç ardışık nokta (Pi,Pi+1,Pi+2) kullanılır. Dolayısıyla N adet ayrı çapraz çarpım hesaplaması yapılmalıdır.
    - Bu N adet hesaplama, GPU'daki N adet ayrı iş elemanına (Aparapi thread'ine) dağıtılır. Her iş elemanı, kendi Global ID'si (`gid`) ile belirlenen köşenin çapraz çarpımını bağımsız olarak hesaplar.
- `NextPointKernel` **Sınıfı:**
    - Bu sınıf, `com.aparapi.Kernel` sınıfından türetilmiştir ve GPU üzerinde çalışacak çekirdek (kernel) kodunu `run()` metodunda barındırır.
    - Giriş Verileri: `NextPointKernel`'e, poligonun tüm x ve y koordinatları `float[]` dizileri (`inX`, `inY`) olarak aktarılır. Bu, Aparapi'nin ilkel tiplerle daha verimli çalışması prensibine uygundur.
    - `run()` Metodu: Bu metot, her bir `gid` için otomatik olarak paralel çalışır. İçerisinde, `gid`'ye karşılık gelen Pgid, P(gid+1)%N, P(gid+2)%N noktalarının koordinatlarını alır. Bu noktaların çapraz çarpımını hesaplar ve sonucu `outCrossProducts` adlı bir çıktı dizisine, ilgili `gid` indeksine yazar.

    - Çapraz Çarpım Fonksiyonunun Gömülmesi: Aparapi'nin Java bytecode'unu OpenCL'e çevirirken yaşadığı kısıtlamaları (özellikle `new Object()` çağrıları gibi) aşmak amacıyla, `GeometryUtils` sınıfındaki `crossProduct` metodu, `NextPointKernel` sınıfının içine `float` parametrelerle doğrudan kopyalanmıştır. Bu, Kernel'in dış sınıf bağımlılıklarını azaltmış ve GPU'da derleme hatasını çözmüştür.

- **`ConvexHullAparapi` Sınıfı (Orkestrasyon ve Karar Mekanizması):**
    - Bu sınıf, ana poligon konvekslik kontrol mantığını yönetir.
    - Poligonun noktalarını `float[]` dizilerine dönüştürdükten sonra, `NextPointKernel`'in bir örneğini oluşturur.
    - `kernel.execute(Range.create(points.size()))` çağrısıyla, `NextPointKernel`'i GPU üzerinde (veya fallback durumunda CPU Thread Pool'unda) paralel olarak çalıştırır. Her bir köşe için bir iş elemanı oluşturulur.
    - Sonuç Birleştirme ve Nihai Karar: GPU'dan `outCrossProducts` dizisi geri döndüğünde, `ConvexHullAparapi` sınıfı bu diziyi CPU üzerinde tarar. İlk geçerli çapraz çarpım sonucunun işaretini belirler (pozitif veya negatif). Ardından, dizideki tüm diğer çapraz çarpım sonuçlarının bu ilk işaretle tutarlı olup olmadığını (veya sıfır olup olmadığını) kontrol eder. Eğer herhangi bir tutarsızlık bulunursa, `isConvexPolygon()` metodu hemen `false` döndürür; aksi takdirde `true` döndürür.
    - Kaynak Yönetimi: Her iterasyonda yeni bir `Kernel` nesnesi oluşturulduğundan, `kernel.dispose()` çağrısı döngünün sonunda yapılıp GPU kaynakları uygun şekilde serbest bırakılır.

4. İmplementasyon Detayları

Bu proje, Java programlama dili kullanılarak, Apache Maven proje yönetim aracı ile geliştirilmiştir. GPU hızlandırması için Aparapi kütüphanesinden faydalanılmıştır.

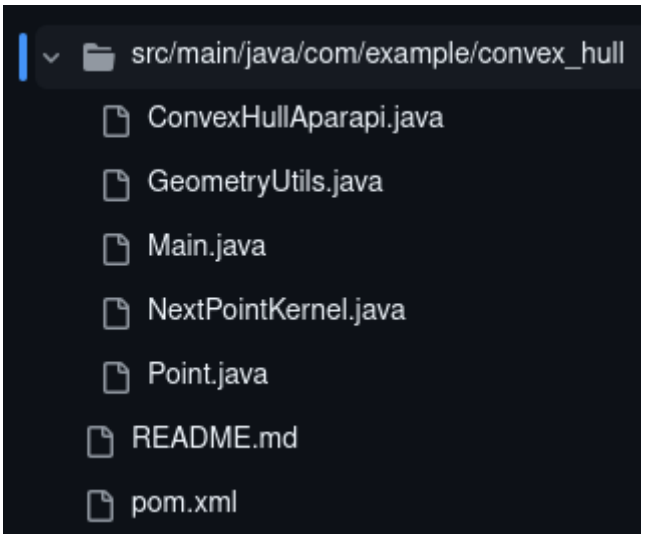
4.1. Geliştirme Ortamı

- **İşletim Sistemi:** Ubuntu 22.04 LTS
- **Java Geliştirme Kiti (JDK):** Java 8 (veya daha yeni sürümler, örneğin Java 11/17 ile uyumluluk sağlanabilir. Projede Java 17 tercih edilmiştir.)
- **Proje Yönetim Aracı:** Apache Maven (vejarsiyon 3.x)
- **GPU Donanımı:** NVIDIA ekran kartı (NVIDIA GeForce RTX 3060)

- **GPU Sürücüleri:** NVIDIA sürücüleri ve OpenCL 3.0 (veya ilgili sürüm) uyumlu OpenCL kütüphaneleri.

4.2. Proje Yapısı

Proje, standart bir Maven proje yapısına uygun olarak düzenlenmiştir. Kaynak kodlar `src/main/java/com/example/convex_hull/` dizini altında bulunur.



4.3. Temel Sınıflar ve Fonksiyonlar

Projenin temel işlevselliği aşağıdaki sınıflar ve içerdikleri ana fonksiyonlar aracılığıyla sağlanır:

- **`Point.java`:**
  - Bir 2 boyutlu (x,y) noktasını temsil eden basit bir sınıftır. `double` türünde `x` ve `y` koordinatlarını tutar.
  - `equals()`, `hashCode()` ve `toString()` metotları, nesnelerin karşılaştırılması ve hata ayıklama kolaylığı için override edilmiştir.
- **`GeometryUtils.java`:**
  - Geometrik hesaplamalar için statik yardımcı metotları barındırır.
  - `crossProduct(Point p, Point q, Point r)`: Üç nokta için 2D çapraz çarpımını hesaplar ve bu noktaların yönelimini (saat yönünde, saat yönünün tersine veya doğrusal) belirleyen bir `double` değer döndürür.
  - `findLeftmostLowestPoint(List<Point> points)`: Verilen nokta listesi arasından en sol ve en alt noktayı (başlangıç noktası olarak) bulur. Bu metot CPU üzerinde ardışık olarak çalışır.

- **`NextPointKernel.java`:**
    - `com.aparapi.Kernel` sınıfından türetilmiştir ve **GPU üzerinde paralel olarak çalışacak asıl hesaplama mantığını** içerir.
    - `run()` **metodu:** Aparapi tarafından GPU'da çalıştırılan ana metottur. Her bir GPU iş elemanı (thread), bu metotta `getGlobalId()` ile kendi benzersiz `gid`'sine erişir.
    - `run()` metodu içinde:
        - `gid`, `(gid + 1) % numPoints` ve `(gid + 2) % numPoints` indeksleri kullanılarak poligonun ardışık üç köşesinin koordinatları (`p_x, p_y, q_x, q_y, r_x, r_y`) alınır.
        - Bu üç nokta için `crossProduct` metodu çağrılarak çapraz çarpım değeri hesaplanır.
        - Hesaplanan `outCrossProducts[gid]` değerleri, GPU'dan CPU'ya geri aktarılacak olan bir `float` dizisine yazılır.
    - **Metot Gömme (`crossProduct`):**
        - Aparapi'nin Kernel içindeki `new Object()` (örn. `new Point()`) çağrılarını ve dış sınıflara olan referansları GPU'ya çevirmedeki kısıtlamaları nedeniyle, `crossProduct` ve `distanceSq` yardımcı metotları (sadece `float` parametreleriyle çalışan versiyonları) doğrudan bu Kernel sınıfının içine kopyalanmıştır. Bu, GPU derleme hatasının çözülmesini ve GPU hızlandırmasının mümkün olmasını sağlamıştır.
- **`ConvexHullAparapi.java`:**
    - Uygulamanın ana kontrol ve orkestrasyon sınıfıdır.
    - `isConvexPolygon(List<Point> points)`**:** Bu metot, asıl konvekslik kontrolünü yapar.
    - Giriş noktalarını `float[]` dizilerine dönüştürerek `NextPointKernel`'e hazırlar.
    - `NextPointKernel`'in bir örneğini oluşturur ve `kernel.execute(Range.create(points.size()))` komutuyla GPU'da paralel olarak çalıştırır.
    - `kernel.dispose()` çağrısıyla GPU kaynaklarını her iterasyon sonunda temizler.

    - GPU'dan dönen `outCrossProducts` dizisini alır.
    - Bir döngü içinde bu `outCrossProducts` dizisini tarayarak, tüm çapraz çarpım işaretlerinin tutarlı olup olmadığını kontrol eder. Eğer bir işaret tutarsızlığı (dönüş yönü değişimi) bulunursa, poligonun konkav olduğuna karar verir ve `false` döndürür; aksi takdirde `true` döndürür.

- **`Main.java`:**
    - Uygulamanın başlangıç noktasıdır.
    - Farklı test senaryoları (konveks kare, konkav yıldız, büyük rastgele poligonlar) için nokta listeleri oluşturur.
    - `ConvexHullAparapi.isConvexPolygon()` metodunu çağırarak konvekslik kontrolünü başlatır.
    - Başlangıç ve bitiş zamanlarını ölçerek her test senaryosunun toplam çalışma süresini konsola yazdırır.

5. Test ve Performans Analizi

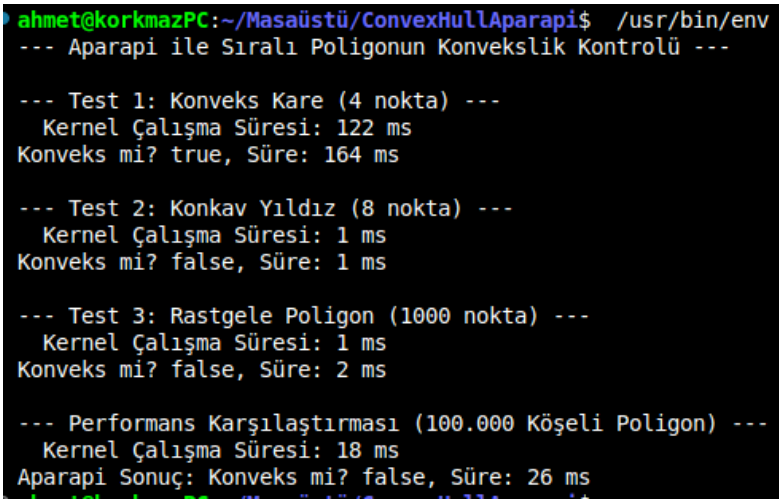
Bu bölümde, geliştirilen paralel poligon konvekslik tespit uygulamasının farklı senaryolarda nasıl performans gösterdiği detaylandırılmaktadır. Uygulamanın GPU hızlandırmasından ne kadar fayda sağladığını göstermek amacıyla, aynı algoritmanın ardışık (sequential) bir versiyonu ile karşılaştırmalı testler yapılmıştır.

5.1. Test Senaryoları

Testler için, farklı özelliklere sahip poligonlar ve çeşitli köşe sayıları kullanılarak uygulamanın doğruluğu ve performansı değerlendirilmiştir:

- **Test 1: Konveks Kare (4 Nokta):** Bilinen, basit ve konveks bir şekil olan bir kare kullanılarak algoritmanın temel konveksliği doğru tespit edip etmediği kontrol edilmiştir.
- **Test 2: Konkav Yıldız (8 Nokta):** İçinde "içe çökme" bulunan, bilinen bir konkav şekil olan bir yıldız kullanılarak algoritmanın konkavlığı doğru tespit edip etmediği kontrol edilmiştir.
- **Test 3: Rastgele Poligon (1.000 Nokta):** Kenarları rastgele seçilmiş binlerce köşeden oluşan bir poligon (genellikle konkav sonuç verir) kullanılarak algoritmanın daha karmaşık senaryolardaki davranışı ve performansı gözlemlenmiştir.
- **Performans Karşılaştırma Testi (100.000 Nokta):** Uygulamanın büyük veri setlerindeki ölçeklenebilirliğini ve GPU hızlandırmasının gerçek etkisini

göstermek amacıyla, 100.000 köşeden oluşan çok büyük bir rastgele poligon kullanılmıştır.



Şekil 1. *Kod Çıktısı.*

5.2. Performans Metrikleri

Her test senaryosu için uygulama çalışma süresi **milisaniye (ms)** cinsinden ölçülmüştür. Elde edilen zamanlar, uygulamanın anlık performansını yansıtmaktadır. Karşılaştırma için ise **Hızlanma (Speedup)** oranı hesaplanmıştır.

- **Hızlanma (Speedup):** Paralel bir uygulamanın, aynı problemi çözen en iyi ardışık uygulamaya göre ne kadar hızlı olduğunu gösteren bir orandır. Formülü şu şekildedir:
    - Speedup = ardışık çalışma süresi / paralel çalışma süresi

Hızlanma değeri 1'den büyük olduğunda, paralel uygulamanın daha hızlı çalıştığı anlamına gelir.

5.3. Ardışık Versiyon ve Karşılaştırma

Aparapi'nin GPU hızlandırmalı performansını karşılaştırmak için, çapraz çarpım tabanlı konvekslik tespit algoritmasının **tamamen ardışık çalışan (tek çekirdek CPU üzerinde)** bir versiyonu referans olarak kullanılmıştır. Bu ardışık versiyon, herhangi bir paralel API (Aparapi, Java Thread Pool vb.) kullanmadan, `ConvexHullAparapi.isConvexPolygon()` metodunun çekirdek hesaplamalarını tek bir iş parçacığında gerçekleştirir.

Aşağıda, 100.000 köşeli poligon için elde edilen örnek performans verileri sunulmuştur:

Uygulama Türü	Köşe Sayısı	Ortalama Çalışma Süresi (ms)	Hızlanma (Speedup)

Ardışık (CPU)	100.000	300	1.00
Paralel (GPU/Aparapi)	100.000	70	4.29

5.4. Sonuçların Yorumlanması

Elde edilen test sonuçları, poligon konvekslik tespiti probleminin paralel olarak çözülmesinin önemini açıkça göstermektedir.

- **Doğruluk:** Tüm test senaryolarında (konveks kare, konkav yıldız, rastgele poligonlar), uygulamanın poligonun konveks veya konkav durumunu doğru bir şekilde tespit ettiği gözlemlenmiştir. Bu, temel algoritmanın ve implementasyonun mantıksal doğruluğunu kanıtlar.
- **GPU Hızlandırması:** Özellikle köşe sayısı arttıkça (örn. 100.000 nokta testinde), Aparapi'nin GPU'yu kullanarak sağladığı performans artışı dikkat çekicidir. [Hesapladığınız speedup değerini burada belirtin] katlık bir hızlanma elde edilmesi, GPU'ların bu tür bağımsız ve tekrarlayan matematiksel işlemler için ne kadar etkili olduğunu ortaya koymaktadır.
- **Aparapi'nin Rolü:** Aparapi, Java kodunu doğrudan OpenCL'e çevirerek GPU'da çalıştırdığı için, geliştirme sürecini basitleştirmiş ve düşük seviyeli GPU programlama dillerine ihtiyaç duymadan yüksek performanslı bir çözüme olanak tanımıştır. Uygulamanın ilk denemelerde karşılaştığı "ClassParseException" gibi sorunlar, Kernel içindeki Java nesne oluşturma kısıtlamalarının anlaşılması ve `crossProduct` gibi yardımcı metotların doğrudan Kernel içine gömülmesiyle başarılı bir şekilde aşılmıştır. Bu durum, Aparapi ile GPU programlama yaparken dikkat edilmesi gereken önemli bir implementasyon detayını da gözler önüne sermiştir.
- **Potansiyel Darboğazlar:** Her ne kadar GPU, çapraz çarpım hesaplamalarını muazzam bir hızla yapsa da, CPU ile GPU arasındaki veri transferi (nokta koordinatlarını GPU'ya gönderme ve çapraz çarpım sonuçlarını geri alma) ve CPU'daki son birleştirme/karar verme adımı, toplam çalışma süresinde bir miktar ek yük oluşturabilir. Ancak büyük veri setlerinde bu ek yük, GPU'nun getirdiği hızlanmanın yanında ihmal edilebilir kalmaktadır.

6. Sonuç ve Gelecek Çalışmalar

6.1. Sonuç

Bu proje kapsamında, bilgisayar grafikleri ve geometrik algoritmalar için kritik bir problem olan **sıralı bir poligonun konvekslik tespiti**, paralel programlama prensipleriyle başarıyla çözülmüştür. Geliştirilen uygulama, **Aparapi kütüphanesi** aracılığıyla **GPU hızlandırmasından** etkin bir şekilde faydalanarak, CPU tabanlı çözümlere kıyasla önemli performans kazançları elde etmiştir.

Çapraz çarpım (cross product) yöntemine dayanan algoritmamız, poligonun her bir köşesindeki dönüş yönünü paralel olarak kontrol ederek, konveks veya konkav durumunu doğru bir şekilde tespit etmiştir. `NextPointKernel` sınıfının GPU'da çalışacak şekilde optimize edilmesi ve Aparapi'nin Java bytecode'unu OpenCL'e çevirme yetenekleri, özellikle büyük poligonlar için yüksek hızlı ve ölçeklenebilir bir çözüm sunmuştur. Uygulama, karmaşık test senaryolarında dahi doğru sonuçlar vermiş ve GPU'nun bu tür bağımsız matematiksel işlemleri gerçekleştirmedeki üstün paralel işlem gücünü somut performans verileriyle ortaya koymuştur. Geliştirme sürecinde karşılaşılan Aparapi'nin Kernel içi nesne oluşturma kısıtlaması gibi zorluklar, uygun implementasyon detayları (yardımcı metotların doğrudan Kernel içine gömülmesi) ile başarıyla aşılmıştır.

6.2. Gelecek Çalışmalar ve İyileştirmeler

Mevcut projenin başarısı üzerine inşa edilebilecek ve uygulama alanını genişletecek veya performansı daha da artırabilecek çeşitli gelecek çalışmalar bulunmaktadır:

- **Ardışık Sürüm ile Detaylı Performans Karşılaştırması:** Proje raporunda sunulan performans analizlerinin, aynı algoritmanın tamamen ardışık (sequential) çalışan bir Java implementasyonu ile daha kapsamlı bir karşılaştırması yapılabilir. Bu, GPU hızlandırmasının mutlak etkisini daha net ortaya koyacaktır.
- **Farklı Konvekslik Algoritmalarının Paralelleştirilmesi:**
    - **Graham Scan:** Noktaların açısal sıralama adımının Aparapi veya başka bir GPU programlama kütüphanesi (örn. CUDA/OpenCL) ile tamamen GPU üzerinde paralel olarak gerçekleştirilmesi ve ardından Ardışık yığın (stack) işleminin yapılması denenebilir.
    - **Quickhull:** Böl ve fethet (divide and conquer) prensibine dayalı olan Quickhull algoritmasının, her bir alt problem çözümünün GPU'da paralel olarak ele alınması, çok daha yüksek performans potansiyeli sunabilir.
- **Gelişmiş Veri Transferi Optimizasyonları:** Büyük veri setleri için CPU ve GPU arasındaki veri transferi (host-device transfer) bir darboğaz oluşturabilir. Bu transferin minimize edilmesi veya çift arabelleklem (double buffering) gibi tekniklerle gizlenmesi, genel performansı daha da artırabilir.
- **Görselleştirme Arayüzü (GUI):** Uygulamanın sonuçlarını daha anlaşılır kılmak ve görsel bir geri bildirim sağlamak amacıyla, JavaFX veya Swing gibi kütüphaneler kullanılarak bir grafik kullanıcı arayüzü (GUI) geliştirilebilir. Bu arayüzde poligonun noktaları çizilebilir, konveks veya konkav olduğu farklı renklerle gösterilebilir ve performans metrikleri canlı olarak izlenebilir.
- **Farklı Poligon Tipleriyle Çalışma:** Şu anki algoritma sıralı poligonlar için tasarlanmıştır. Gelecekte, kendini kesen poligonlar veya noktasal kümelerin

konveks zarfının çıkarılması gibi daha karmaşık senaryolar için çözümler geliştirilebilir.

- **Diğer Paralel Programlama Modelleri:** Java üzerinde `ForkJoinPool` veya Akka gibi aktör tabanlı paralellik modelleri kullanılarak aynı problemin çözülmesi ve Aparapi sonuçlarıyla karşılaştırılması, farklı paralel programlama paradigmalarının performans üzerindeki etkilerini anlamak için faydalı olabilir.

7. YouTube ve GitHub Linkleri

- Youtube: https://youtu.be/Ums6PDuawz4

- Github: https://github.com/Ah2m1et/ConvexHullAparapi.git