KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

KOŞUT PROGRAMLAMA İLE GRAFİK İŞLEMCİ ÜZERİNDE ÖRNEK BİR UYGULAMA GELİŞTİRME

YÜKSEK LİSANS TEZİ

Bilgisayar Müh. Mengü DEMİR

MAYIS 2012
TRABZON

KARADENİZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

KOŞUT PROGRAMLAMA İLE GRAFİK İŞLEMCİ ÜZERİNDE ÖRNEK BİR UYGULAMA GELİŞTİRME

Bilgisayar Mühendisi Mengü DEMİR

Karadeniz Teknik Üniversitesi Fen Bilimleri Enstitüsünce
"BİLGİSAYAR YÜKSEK MÜHENDİSİ"
Unvanı Verilmesi İçin Kabul Edilen Tezdir.

Tezin Enstitüye Verildiği Tarih : 3.05.2012
Tezin Savunma Tarihi : 23.05.2012

Tez Danışmanı : Yrd. Doç. Dr. Mustafa ULUTAŞ

Trabzon 2012

Karadeniz Teknik Üniversitesi Fen Bilimleri Enstitüsü

Bilgisayar Mühendisliği Anabilim Dalında

Mengü Demir tarafından hazırlanan

KOŞUT PROGRAMLAMA İLE GRAFİK İŞLEMCİ ÜZERİNDE ÖRNEK BİR UYGULAMA GELİŞTİRME

başlıklı bu çalışma, Enstitü Yönetim Kurulunun 24 / 01 / 2012 gün ve 1440 sayılı kararıyla oluşturulan jüri tarafından yapılan sınavda

YÜKSEK LİSANS TEZİ

olarak kabul edilmiştir.

Jüri Üyeleri

Başkan: Doç. Dr. Murat EKİNCİ **…...……**

Üye: Doç.Dr. Ali GANGAL **…...……**

Üye : Yrd. Doç.Dr. Mustafa ULUTAŞ **……...……**

Prof. Dr. Sadettin KORKMAZ

Enstitü Müdürü

ÖNSÖZ

Bilgisayar donanımlarında her geçen gün yaşanan iyileşmelerle birlikte bilgisayar hızları artsa da bu gelişmeye eş zamanlı olarak onlardan umulan beklentiler de yükselmektedir. Bu yüzden pek çok durumda koşut programlama yöntemlerine başvurmak zorunlu hale gelmektedir.

Öteden beri merkezi işlem birimi kullanarak koşut programlama yapılabilecek pek çok araç bulunmaktadır ancak grafik işlem birimi kullanılarak koşut programlama yapılması nispeten daha yeni bir fikirdir. Bu çalışmada yeni sayılabilecek bu düşünceden hareketle merkezi işlem biriminde çalışması uzun zaman alan bir görüntü işleme yazılımı grafik işlem birimi üzerinde icra edilmiş ve işlem süresi bakımından kayda değer bir iyileşme elde edilmiştir.

Çalışmamda danışmanlığımı üstlenen hocam Yrd. Doç. Dr. Mustafa ULUTAŞ'a, değerli yardımlarını esirgemeyen arkadaşım Cihat KELEŞ'e, Mustafa YAZICI, Aynur ŞİMŞEK ve Sayın. Hayati TÜRE'ye uygarlığın bugünkü düzeyine ulaşmasında büyük emeği geçmiş bilim insanlarına ve beni yetiştirip bugünlere getiren aileme sonsuz teşekkürlerimi sunarım.

Mengü DEMİR

Trabzon 2012

TEZ BEYANNAMESİ

Yüksek Lisans Tezi olarak sunduğum "KOŞUT PROGRAMLAMA İLE GRAFİK İŞLEMCİ ÜZERİNDE ÖRNEK BİR UYGULAMA GELİŞTİRME" başlıklı bu çalışmayı baştan sona kadar danışmanım Yrd. Doç. Dr. Mustafa ULUTAŞ'ın sorumluluğunda tamamladığımı, örnekleri kendim topladığımı, deneyleri ilgili laboratuarlarda yaptığımı, başka kaynaklardan aldığım bilgileri metinde ve kaynakçada eksiksiz olarak gösterdiğimi, çalışma sürecinde bilimsel araştırma ve etik kurallara uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul ettiğimi beyan ederim. 15/06/2012

Mengü DEMİR

İÇİNDEKİLER

Sayfa No

Yüksek Lisans Tezi

ÖZET

KOŞUT PROGRAMLAMA İLE GRAFİK İŞLEMCİ ÜZERİNDE ÖRNEK BİR UYGULAMA GELİŞTİRME

Mengü DEMİR

Karadeniz Teknik Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı
Danışman: Yrd. Doç. Dr. Mustafa ULUTAŞ
2012, 87 Sayfa, 5 Sayfa Ek

Kullanıcıların karmaşık problemleri bilgisayar ile çözme beklentisi sürekli artış göstermektedir. Karmaşık problemleri çözmek için geliştirilen algoritmaları genel amaçlı mikroişlemciler üzerinde gerçek-zamanlı olarak çalıştırmak da çoğu durumda mümkün olmamaktadır. Bu gerekçelerden yola çıkan donanım üreticileri, algoritmaların çalışma zamanını azaltacak ve birçok işlem elemanı üzerinde koşut olarak çalıştırılabilecek yeni mikroişlemciler tasarlamaktadır.

Bu tezde NVIDIA firması tarafından geliştirilen ve grafik işlem birimleri üzerinde çalışan CUDA teknolojisi tanıtılmıştır. Daha sonra CUDA teknolojisinin etkinliğini kanıtlamak için bulanık mantık kuralları ile hareketli bir taşıyıcı bant üzerindeki zarlı fındıkların gerçek-zamanda tanınması problemi için bir algoritma kodlanmıştır. Bu algoritmanın hem koşut hem de seri çalışma süreleri rapor edilmiştir.

Anahtar Kelimeler: *Koşut Programlama Merkezi İşlem Birimi, Grafik İşem Birimi, CUDA,* Bütünleşik Aygıt İşleme Mimarisi, Bulanık Mantık

Master Thesis
SUMMARY

PARALLEL PROGRAMMING ON GRAPHICS PROCESSORS AND AN APPLICATION

Mengü DEMİR

Karadeniz Technical University
The Graduate School of Natural and Applied Sciences
Computer Engineering
Supervisor: Assoc. Prof. Mustfafa ULUTAŞ
2010, 87 Pages, 5 Pages Appendix

Users' expectation to solve complex problems by computers is an ever-increasing trend. Algorithms developed to solve complex problems often have high algorithmic complexity and can not run on general purpose microprocessors in real-time. Hardware manufacturers inspired by this demand design innovative microprocessors with many processing elements to run algorithms in parallel to reduce run time.

A parallel programming technique, Compute Unified Device Architecture (CUDA) by Nvidia is introduced in this thesis. CUDA is then used to program an algorithm based on the fuzzy logic rules to identify skinned hazelnuts on a moving conveyor belt in real time to prove the effectiveness of the technique. Run times for both parallel and serial implementation of the same algorithm is reported.

Key Words: *Parallel programming, Central Processing Unit, Graphics Processing Unit CUDA, Compute Unified Device Architecture, Fuzzy Logic*

ŞEKİLLER DİZİNİ

Sayfa No

TABLOLAR DİZİNİ

Sayfa No

SEMBOLLER DİZİNİ

BM	Bulanık Mantık
CUDA	Compute Unified Device Architecture
GİB	Grafik İşlem Birimi
HY 1.1	Hesaplama Yeteneği 1.1
HY 2.1	Hesaplama Yeteneği 2.1
MİB	Merkezi İşlem Birimi
MPI	Message Passing Interface
OKA	Ortalama Kaydırma Algoritması
ÖDED	Ölçeklndirilebilir Değişmez Öznitelik Dönüşümü

1. GENEL BİLGİLER

1.1. Giriş ve Çalışmanın Amacı

Bilgisayarlar günümüzde tıptan, güvenliğe, eğlenceden üretime insan yaşamının pek çok anında kendine yer bulmuş aygıtlardır. Başlangıçta temel matematiksel işlemleri yapmak, veri saklamak gibi basit işlemleri gerçekleştirmek için tasarlanan bilgisayarlar günümüzde çevresinde olup biten olayları kavrayan, akıllı çözümler üretebilen, zeki cihazlar olma yolunda ilerlemektedir. İnsanlar makinelerden kat kat zeki, algılama noktasında ileride ve fikir yürütme noktasında önde olsalar da yorulma, kapris, kıskançlık gibi birçok doğal dezavantaja sahip canlılardır. Bütün bu sebeplerle bilgisayarlardan insan yerine düşünme, karar verme ve hiç olmazsa insanlara belirli noktalarda danışmanlık yapması beklenmektedir. Deyim yerindeyse günümüz bilgisayarlarından beklenen hızlı bir köle olmaları değil anlayışlı ve yorulmaz bir danışman, yoldaş olmalarıdır.

Bilgisayarın gerçek dünyadaki görüntüleri kavrayıp değerlendirmesi görüntü işleme, bilgisayar görmesi ve makine öğrenmesi disiplinlerinin ilgi alanına girmektedir. Bu disiplinler daha önceden yapay zekânın bir alt kolu olarak değerlendirilse de bugün ayrı birer bilim dalı konumuna gelmişlerdir. Görüntü işleme iki boyutlu sayılar dizisinden ibaret olan sayısal görüntüler üzerinde gürültü giderme, kenar bulma gibi ön işleme adımlarının genel adıdır. Bilgisayar görmesi ise görüntü işlemenin bir sonraki adımı durumundadır. Ön işlenmiş görüntülerin bölütlenmesi ve bilgiler çıkarılması bilgisayar görmesinin başlıca uğraşı alanlarıdır. Bu çalışmada kullanılan yöntemler daha çok bu iki disiplinin ilgi alanı içerisindeki algoritmalardan meydana gelmektedir.

Fındık, üretiminde Türkiye'nin dünyada birinci sırada olduğu çok önemli bir tarım ürünüdür. Ayrıca günümüz çikolata sanayisinin önemli bir katkı maddesi konumundadır. Kabuğundan soyulan ve dışındaki kahverengindeki zardan ayıklanan fındık çikolata fabrikalarına satılmaktadır. Fındığın dışında bir miktar ya da tamamen zar kalması çikolatanın kalitesini düşüren bir olumsuzluktur. Bu yüzden çikolata fabrikaları fındık üreticilerinden tamamen zarsız fındıklar talep etmektedir. Her ne kadar zar ayıklama makineleri bu zarları büyük oranda ayıklasalar da yine de kısmen ya da tamamen zarlı fındıklar kalabilmektedir. Bu durumun önüne geçmek için fındık fabrikaları ayıklama makinesinden çıkan fındığı bir de işçilere kontrol ettirmektedir. Bu çalışmada insanların yaptığı bu kontrolü bilgisayarların yapabilmesi için görüntü işleme, bilgisayar görmesi

ve bulanık mantık disiplinleri içerisinde yer alan algoritmalarda çözüm üretilmeye gayret gösterilmiştir. Bunun için geliştirilen kırmızı renkli bir bant üzerinde hareket halindeki fındıkların görüntüleri içerisinde fındıklar tespit edilmiş ve bulanık mantık yöntemleriyle zarlı ya da zarsız oldukları karar verildikten sonra konum kestirim algoritmalarının da yardımıyla izlenmiş ve zarlı fındıkların varlıkları kullanıcıya bildirilmiştir. Bütün bu işlemler hareketi tespit ve takip kapsamında düşünüldüğü için hızın büyük bir önem taşıdığı işlemlerdir. Bundan ötürü algoritmalar grafik işlem birimi üzerinde koşut programlamaya olanak veren CUDA teknolojisi yardımıyla icra edilmiş ve kayda değer bir hız artışı sağlanmıştır.

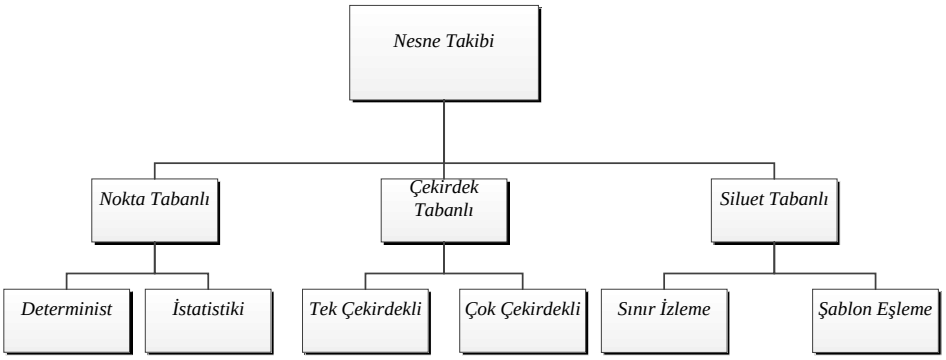
1.2. Hareket Algılama ve İzleme Algoritmaları

Hareketin algılanması ve izlenmesi birbiriyle yakından ilişkili disiplinlerdir. Hareketin algılanması göreceli olarak kolay bir işlemdir ancak ikinci aşama olan izleme daha güç bir süreçtir. Çünkü tespit edilen her hareketlinin gerçekten cisim olup olmadığı, eğer cisim ise hangi yörüngeyi takip ettiği gibi başat sorunlarla uğraşır.

Nesne izlenmesi ya da diğer bir adıyla takibi hayli güç bir iştir. Bu güçlükler Bettencourt ve Soumers tarafından şu başlıklar altında sıralanmıştır [1].

1. Görüntü üzerinde meydana gelen gürültüler.
2. Nesnelerin tam olarak ayırt edilmelerini sağlayacak fiziksel bir yapıya sahip olmamaları
3. Değişik nesne şekilleri
4. Işık şiddetinin değişimi
5. Gerçek zamanlı izleme için hızlı işlem zorunluluğu
6. 3 boyutlu video verilerinin 2 boyutlu görüntü alanına izdüşümünden kaynaklanan bilgi yitimi

Çoklu nesne izlemede bunlara ek olarak cisimlerin birbiri ardına geçip gözlenemez olması ve belirli bir zaman sonra tekrar görünür olmalarından kaynaklanan bilgi kirliliğini de ekleyebiliriz. Bilim dünyasında nesne izlemeyle ilgili birçok çalışma yapılmıştır. Bu çalışmalar Talu tarafından Şekil 1.1’deki gibi sınıflandırılmıştır [2]



Şekil 1.1. Nesne izleme algoritmalarının sınıflandırılması

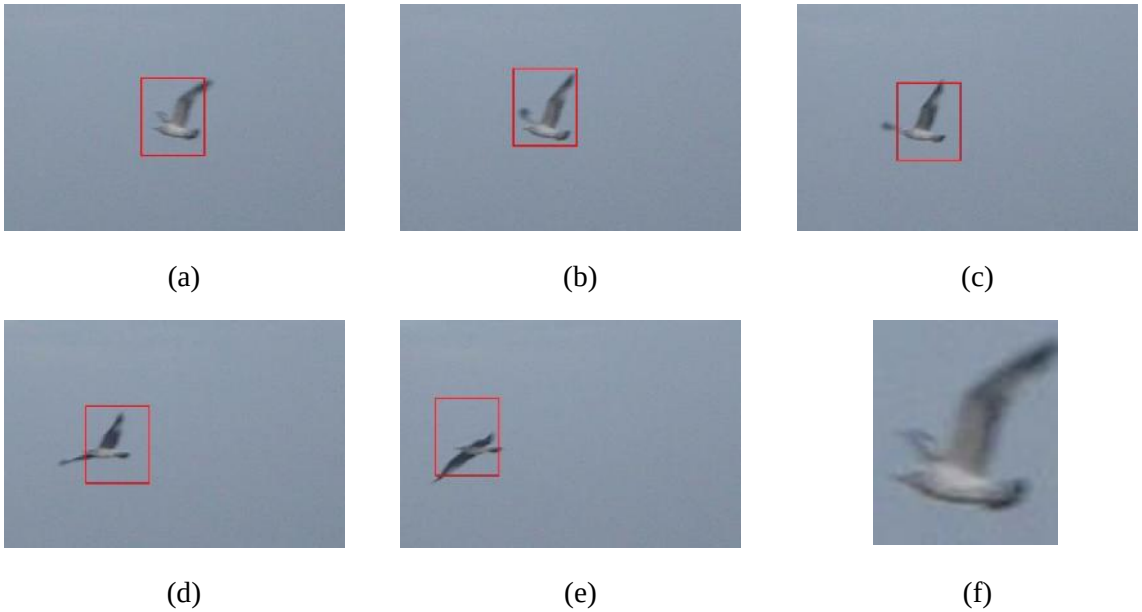
Bu sınıflandırma içerisindeki nokta tabanlı yöntemler takip edilen her nesnenin tek bir nokta ile ifade edildiği yöntemlerdir. Çekirdek tabanlı yöntemler nesnelerin düzgün geometrik bir çekirdek içerisindeki olasılık yoğunluk işlevleriyle takip edildiği yöntemlerdir. Siluet tabanlı yöntemler ise düzgün geometrik bir şekle sahip olmayan hedefleri takipte önceki sahnelerde ortaya çıkan hedeflerin özelliklerinin aranması temeline dayanır [2].

Bu çalışmada tek bir sabit kameradan gelen görüntüler üzerinde hareketli cisimlerin izlenmesi üzerinde durulmuştur. Çalışmada kullanılan algoritmalar incelendiğinde görülmektedir ki pek çok algoritmanın karmaşıklığı hayli yüksektir. Bu durum algoritmalar bir arada kullanıldığında makinenin hızında ciddi yavaşlamalara sebep olmaktadır. Dünya üzerinde bu tür yavaşlıkları gidermek için birçok hızlandırma tekniği geliştirilmektedir. Bunu gerçekleştirmek için yapılan çalışmaları donanım tabanlı ve yazılım tabanlı olarak ikiye ayırmak olasıdır. Yazılım tabanlı çözümlerin de en başta geleni koşut programlamadır. Koşut programlama uygun olan bütün işlem birimlerini hesaplamaya eş zamanlı olarak dâhil ederek zamandan kazanmayı hesaplayan yöntemlerin genel adıdır [3]. Çalışmada grafik işlem birimi üzerinde koşut programlamayı sağlayan CUDA teknolojisi kullanılmıştır.

Bu çalışmada kaynak taraması kapsamında çok sayıda hareket algılama ve nesne takibi algoritması incelenmiştir. Bunlar arasında çekirdek tabanlı yöntemlerden ortalama kaydırma [4] ve CamShift [5], siluet tabanlı yöntemlerden ölçeklenebilir değişmez öznitelik dönüşümü [6], nokta tabanlı yöntemlerden basit fark [7] ve optik akış [8,9,10] algoritmaları sayılabilir.

1.2.1. Ortalama Kaydırma Algoritması

Ortalama kaydırma algoritması (OKA) hareket takibinde sıklıkla başvurulan çekirdek tabanlı izleme algoritmalarından birisidir. Fukunaga ve Hostetler tarafından önerilmiş bir yöntemdir. Ortalama kaydırma algoritması kayma yöneyini hesaplayarak nesne merkezinin gerçekleştirmiş olduğu hareketi değerlendirir. Karşılaştıracağı bölge ve hedef arasında benzerlik çekirdeği kullanılır. Kullanılan benzerlik çekirdeğine göre çeşitli türlerde Ortalama Kaydırma algoritmaları mevcuttur. Hedefteki öteleme, kaydırma, dönme, boyut değişimi gibi etkenlere karşı dayanıklı yeni ortalama kaydırma algoritmaları üzerinde çalışmalar [11,12] vardır. Şekil 1.2'de örnek bir hareketli görüntü dizisi içindeki aranan cisim ve OKA ile elde edilen sonuçlar görülmektedir.



(a) (b) (c)

(d) (e) (f)

Şekil 1.2. Örnek görüntüler üzerinde ortalama kaydırma algoritması ile hedef takibi (a), (b), (c), (d), (e) videonun değişik zamanlardaki sahneleri (f) her bir sahnede takip edilen cisim

1.2.1.1. Bhattacharyya Katsayısı

Bhattacharyya yöntemi iki ayrık ya da sürekli olasılık dağılımının birbirine ne kadar benzediğini ölçen istatistikî bir yöntemdir [13]. Bhattacharyya katsayısı b, p ve q olasılık yoğunluk dağılımlarını temsil etmek üzere (1) 'deki gibi hesaplanır. Tablo 1'de iki tane 4

bitlik gri düzeyli görüntünün olasılık yoğunluğu için Bhattacharyya katsayısı hesabı görülmektedir

$$b = \sum_{u=1}^{m} \sqrt{\bar{p}_u \bar{q}_u} \quad (1)$$

Tablo 1. Bhattacharyya katsayısı hesabı örneği

Bit	*1.Görüntünün Piksel Sayısı*	*2.Görüntünün Piksel Sayısı*	*1.Görüntünün Olasılık Yoğunluğu*	*2.Görüntünün Olasılık Yoğunluğu*	*Bit Başına Bhattacharyya*
1	52	17	0.0601	0.0247	0.0015
2	10	39	0.0116	0.0566	0.0007
3	82	83	0.0948	0.1205	0.0114
4	82	80	0.0948	0.1161	0.0110
5	72	6	0.0832	0.0087	0.0007
6	15	40	0.0173	0.0581	0.0010
7	66	53	0.0763	0.0769	0.0059
8	52	42	0.0601	0.0610	0.0037
9	97	66	0.1121	0.0958	0.0107
10	65	63	0.0751	0.0914	0.0069
11	80	29	0.0925	0.0421	0.0039
12	45	43	0.0520	0.0624	0.0032
13	43	2	0.0497	0.0029	0.0001
14	83	98	0.0960	0.1422	0.0136
15	8	17	0.0092	0.0247	0.0002
16	13	11	0.0150	0.0160	0.0002
Bhattacharyya Katsayısı					0.0748

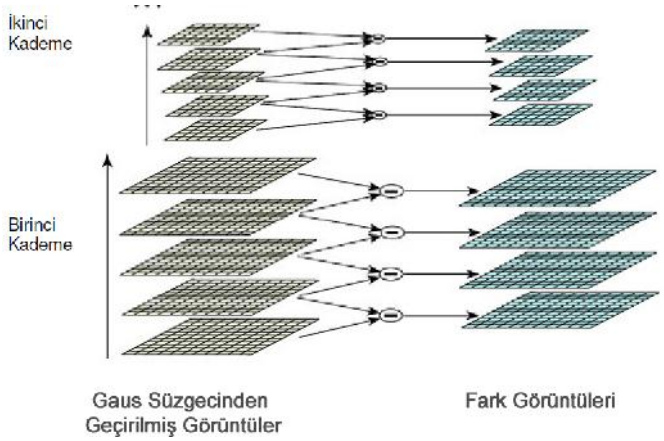
1.2.2. Ölçeklenebilir Değişmez Öznitelik Dönüşümü

Ölçeklenebilir değişmez öz nitelik dönüşümü [6] (ÖDÖD) 1999 yılında David G. Lowe tarafından ortaya atılmış bir yöntemdir. Özellikli nokta (köşe, kenar) eşlemeye dayalı bir yöntemdir. Dört temel aşamadan oluşur [14]. Bunlar sırasıyla ölçeksel uzaydaki uç noktaların elde edilmesi, kilit nokta konumlarının belirlenmesi, döngüsel değişime dayanıklılık kazandırılması ve kilit nokta tanımlayıcılarının bulunmasıdır. ÖDÖD'nün en büyük artısı döndürme, öteleme gibi işlemlere karşı son derece dayanıklı olması ve iyi sonuçlar üretmesidir. Ayrıca işbu yöntem ile arka plan çıkarım yönteminde olduğu gibi

sabit bir kamera gereksinimi yoktur. Fakat yavaş çalışması ve köşe içermeyen cisimlerin bulunmasında başarısız olması olumsuz tarafıdır.

1.2.2.1. Ölçeksel Uzaydaki Uç Noktaların Elde Edilmesi

Bu aşamada resim önce değişik standart sapma değerlerine sahip Gauss süzgeçlerinden geçirilir ve oluşan yeni görüntüler birbirinden çıkarılır. Bu işleme Gausslar'ın Farkı adı verilmektedir. Bu işlem resim boyutları her seferinde yarıya indirilerek - yeni kademeler oluşturularak- yinelenir. Şekil 1.3'te bu kademe oluşturma işlemi görülmektedir [14].



Şekil 1.3. Gauss'lar farkı yöntemi

1.2.2.2. Kilit Nokta Konumlarının Belirlenmesi

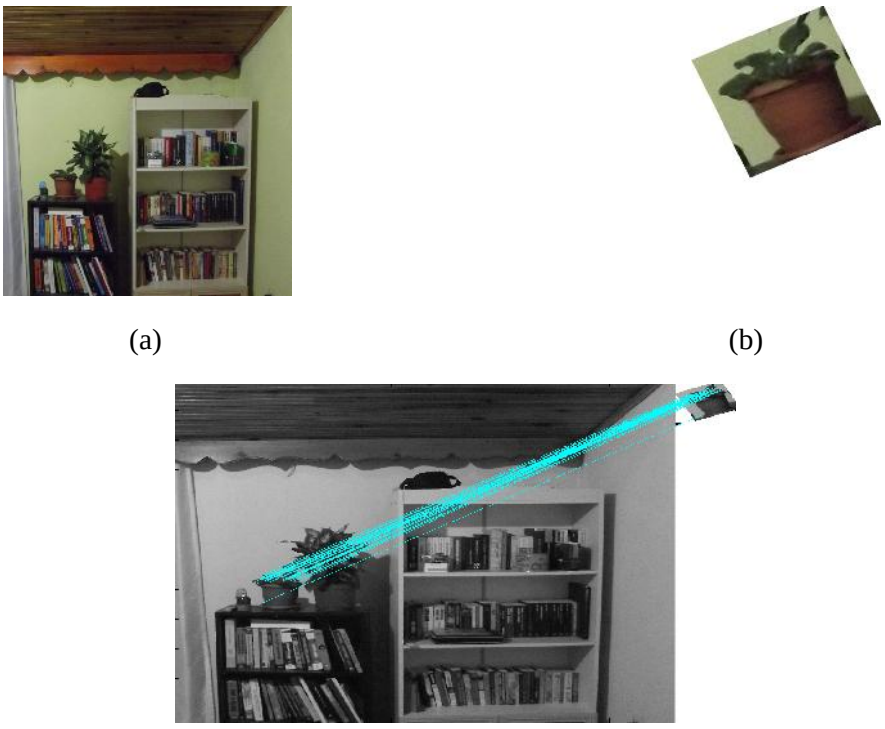
Ölçeksel uzaydaki uç noktaların belirlenmesi sonrasında elde edilen uç noktaların çoğu düşük karşıtlığa sahip ve daha çok kenarlarda tespit edilen gürültü olarak adlandırılabilecek noktalardır. Konum belirlenirken yapılacak ilk iş bu gereksiz noktaların elenmesi ve asıl kilit noktaların ortaya çıkarılmasıdır. Bunun için ilk aşamada bulunan aday noktaların konumları interpolasyon yardımıyla atanır. Bu işlem sırasında Taylor serisi kullanılır. Sapması belirli bir değerden küçük olan noktalar tasfiye edilir.

1.2.2.3. Döngüsel Değişime Dayanıklılık Kazandırma

Kilit noktaların konumları düzgün olarak belirlendikten sonra sıradaki işlem döngüsel değişime dayanıklılık kazandırmaktır ki bu ÖDED algoritmasına asıl değerini kazandıran kısımdır. Daha önce Gauss süzgecinden geçirilmiş resmin gradyan büyüklüğü ve açısı hesaplanır. Bu işlem kilit noktaların komşu her bir pikseli için tekrarlanır.

1.2.2.4. Kilit Noktalara Ait Tanımlayıcıların Tespit Edilmesi

Bu aşama ÖDED'nün son aşamasıdır. Bunun için uygun ölçekte noktaların çevresindeki enerji yoğunlukları örneklenir ve ilinti (korelasyon) kullanılarak bunlar arasında eşleştirme yapılır. Şekil 1.4'te Örnek bir resim için ÖDED algoritmasının sonuçları görülmektedir.



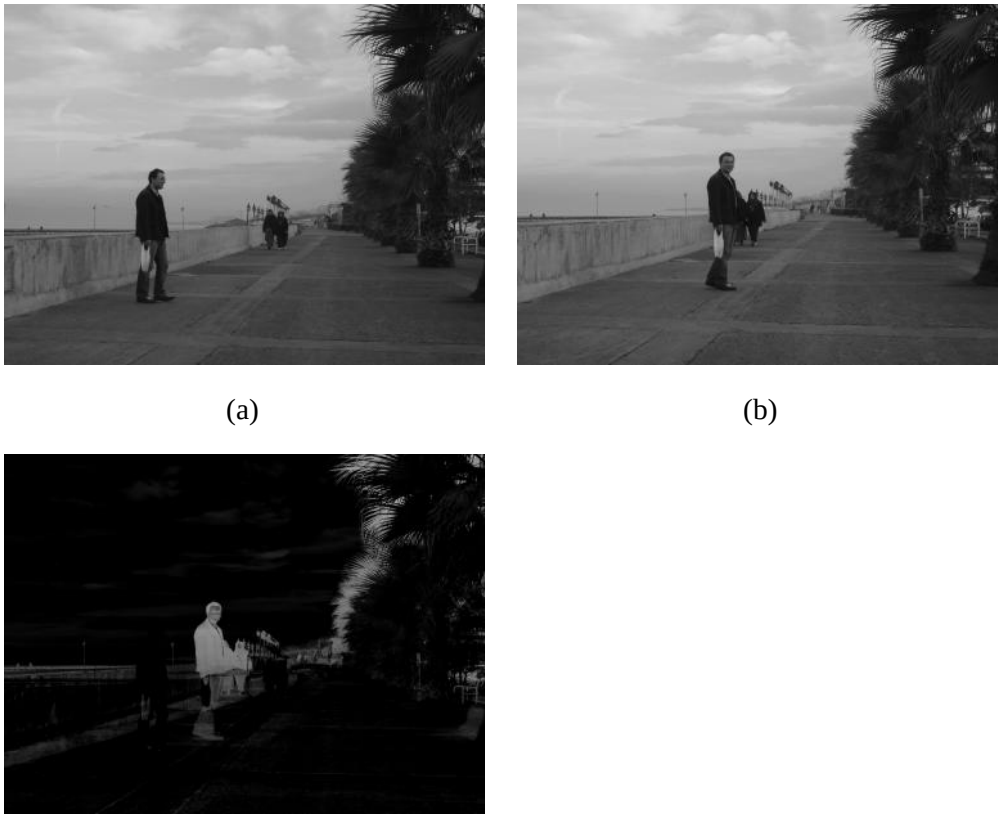
(a) (b)

(c)

Şekil 1.4. ÖDÖD ile iki görüntünün karşılaştırılması (a) ana görüntü (b) aranacak görüntü (c) birbiriyle eşleşen uç noktalar

1.2.3. Basit Fark Yöntemi

Eğer elimizde iki farklı görüntü varsa hareket tespitinin en kolay yolu iki görüntünün farkını almak olacaktır [15]. Basit fark alma yöntemi hareket tespitinde en kolay ve hızlı yöntemdir ancak olumsuz tarafı gürültüye karşı aşırı duyarlı olmasıdır. Bir diğer dezavantajı da kameranın bu yöntemde sabit kalmak zorunda olmasıdır. Kamera hareketinin meydana gelmesi durumunda arka plan da değişeceğinden yanlış bölgeler hareketli olarak tanımlanabilir. Yöntem gürültüye karşı duyarlı olduğu için gürültü giderme işlemleri uygulanmış görüntüler üzerinde kullanılmıştır. Şekil 1.5'te bir video görüntüsünden alınan iki farklı resim ve bunların farkları görülmektedir.



(a) (b)

(c)

Şekil 1.5. Basit fark(arka plan çıkarımı) ile hareket tespiti (a) birinci sahne (b) ikinci sahne (c) iki sahnenin farkı

1.2.4. Optik Akış Yöntemleri

Gerçek yaşamdaki üç boyutlu hareketli cisimlerin iki boyutlu cisimler üzerinde iz düşümleri yapıldığında aynı cisimler farklı sahnelerde birbirinin eşleniği olan pikseller üzerine denk gelirler. Bu da hareketin -bir şekilde önü kapanmadıysa- görüntü düzlemindeki noktaların yer değişimi olarak tanımlanabileceği anlamına gelir. Bu düşünceden yola çıkarak hareket tespit eden yöntemlere genel olarak optik akış yöntemleri adı verilir [16]. Optik akış yöntemleri kamera hareketinden bağımsız yöntemlerdir. Ancak çok yavaş çalışırlar ve gürültülü sonuç üretirler. Dizinde bazı optik akış yöntemleri bulunmaktadır. Burada bu yöntemlerden alan tabanlı yaklaşım ve fark tabanlı yaklaşım üzerinde durulmuştur.

1.2.4.1. Alan Tabanlı Yaklaşım

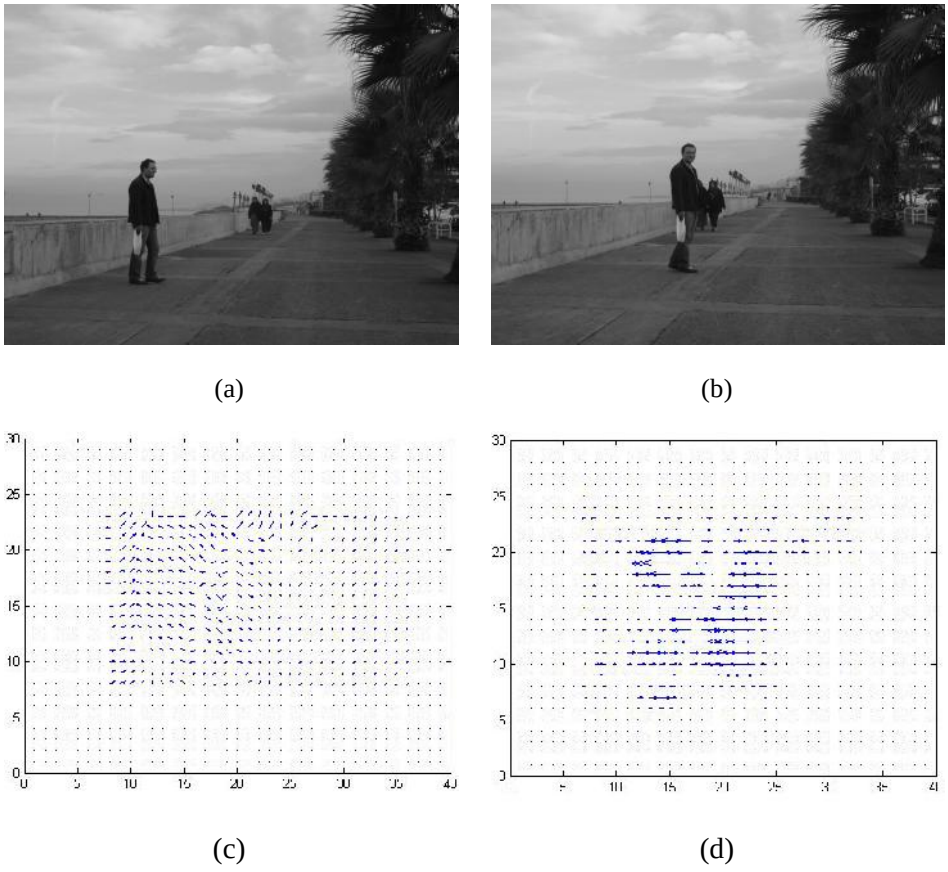
Barnard ve Fisher tarafından geliştirilen bu yöntemde piksellerin komşuluğundaki piksellerle birlikte benzerliği üzerinde durulur [16]. Alan tabanlı yaklaşımın genel formülü (2)'de olduğu gibidir.

$$P(t+1)_{x+\partial x, y+\partial y} = P(t)_{x,y} + H(t)_{x,y} \quad (2)$$

Burada $H(t)_{x,y}$ işlevi görüntüler arasındaki şiddet farkını, x+ ∂x, y+ ∂y de ilgili pikselin t+1 anındaki konumunu temsil etmektedir. Şekilde alan tabanlı yakfaşım sonucu elde edilmiş hareket yöneyleri görülmektedir.

1.2.4.2. Fark Tabanlı Yaklaşım

Fark tabanlı yaklaşım iteratif bir optik akış yöntemidir. Fark tabanlı yaklaşım ile optik akış yöntemleri arasında [9,17 ,18] sayılabilir. Bu yöntemler arasından en yaygın kullanılan Horn ve Schunk'un yöntemidir. Pikselin yeni ve eski konumlarındaki parlaklık değerinin aynı olacağı varsayımıyla hareket tespiti yapar. Şekil 1.6'da fark tabanlı yaklaşım sonucu elde edilmiş hareket yöneyleri görülmektedir.



(a) (b)

(c) (d)

Şekil 1.6. Optik akış yöntemleriyle hareket tespiti (a) Birinci sahne (b) İkinci sahne (c) Alan tabanlı optik akış algoritması sonucu elde edilen hareket yöneyleri (d) Fark tabanlı optik akış algoritması sonucu elde edilen hareket yöneyler

1.3. Koşut Programlama

Bilgisayar bilimlerinde her geçen gün yaşanan gelişmeler ışığında yeni algoritmalar geliştirilmektedir. Özellikle görüntü işleme, makine öğrenmesi, bilgisayar görmesi gibi alanlarda geliştirilen algoritmalar ağır matematiksel işlemlere dayalı olduğu için bilgisayar üzerinde icra edilmesi zaman alan, yüksek hesaplama karmaşıklığına sahip algoritmalardır. Bu durum bilhassa parmak izi tanıma, hedef takibi benzeri saniyelerin dahi önemli olduğu işlemlerde ciddi bir zamanlama sorununu beraberinde getirmektedir. Zaman sorununu çözebilmek için çeşitli yöntemler üzerinde durulmaktadır. Bu yöntemleri donanım tabanlı ve yazılım tabanlı olmak üzere iki ana başlıkta incelemek olasıdır. Bu çalışmada yazılım tabanlı bir çözüm olan koşut programlama irdelenmiştir. Koşut programlama, verileri iş yapacak olan birimlere dağıtarak sonuçları daha erken alma esasına dayanan yöntemlerin genel adıdır.

Koşut programlamada görevi yapacak olan işlem birimlerinden hiç birinin boşta kalmaması, her birinin hesaplamaya başından sonuna kadar katılması istenir. Ayrıca işlemciler arasındaki iletişimin en aza indirilip haberleşme yükünün azaltılması salık verilen diğer bir noktadır. Yapılmak istenen işin boyutu büyüdükçe koşut programlamadan alınan verim de artacaktır. Eğer iş yükü az ise birimler arasındaki gereksiz haberleşme yükünden dolayı koşut programlama ardışık programlamadan daha geç çözüm üretebilir.
Bunların ötesinde yürütme zamanını etkileyen en önemli parametrelerden biri algoritmanın koşut programlamaya yatkınlığıdır. Bazı algoritmalar koşut programlamaya son derece uygun iken bazen kısmi bir koşutlama yapılabilmektedir. Bir önceki adımda üretilen sonucun sonraki adımda girdi olarak kullanıldığı algoritmalar – bu durum veri bağımlılığı olarak adlandırılır – başta olmak üzere bir takım algoritmaları kullanarak koşut programlama yapmak olanaksızdır. Aşağıda koşut programlamayı değerlendirmede kullanılan temel kavramlar incelenmektedir.

1.3.1. Koşut Programlamada Temel Kavramlar

Koşut programlama sonuçlar değerlendirilirken en çok kullanılan ölçütler hızlanma ve verimliliktir.

1.3.1.1. Hızlanma

Hızlanma, işlem birimi artışıyla zaman arasındaki bağıntıdır[19]. Bir işlemciyle hesaplama süresinin çok sayıda işlemciyle yapılan hesaplamanın süresine oranıdır. İşlemcinin artmasıyla orantılı olarak işlem zamanının da azalması beklenir. Hızlanmadaki bu tür bir artışa doğrusal hızlanma ya da ideal hızlanma denir. Hızlanma (3) formülüyle hesaplanır.

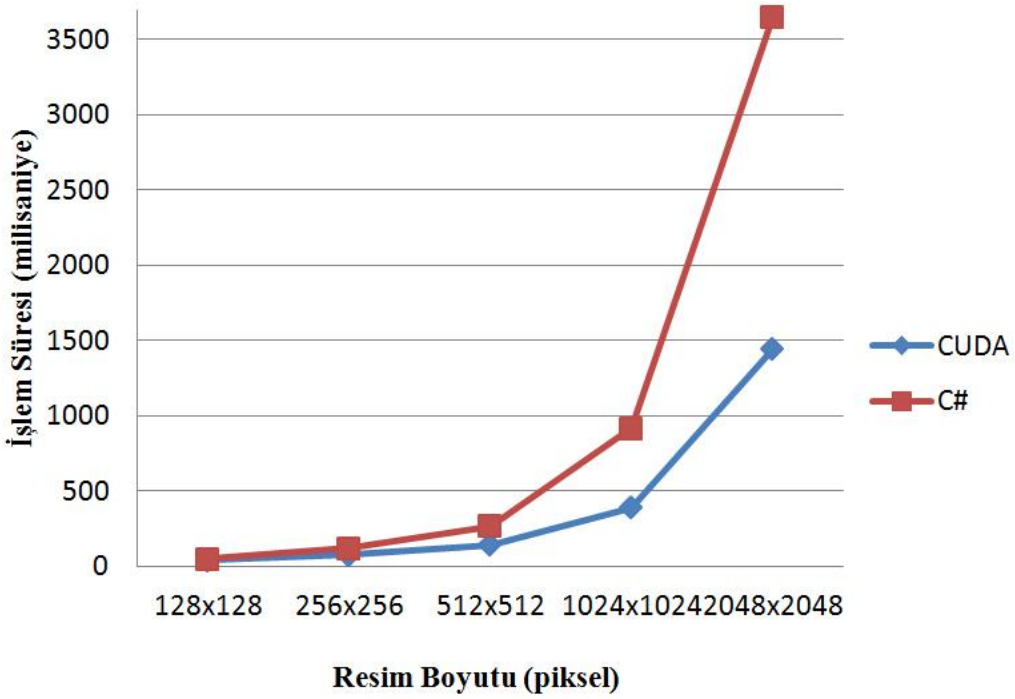
$$Hızlanma = \frac{1\ işlem\ birimiyle\ Hesaplama\ süresi}{n\ işlem\ birimiyle\ hesaplama\ süresi} \quad (3)$$

1.3.1.2. Verimlilik

İşlem birimi artışıyla elde edilen faydayı ölçmeye yarar. Hızlanmanın işlemci sayısına oranıyla elde edilir. (4) formülüyle hesaplanır. İdeal bir hızlanmada verim %100'dür.

$$Verimlilik = \frac{100 \times Hızlanma}{işlem\ birimi\ sayısı} \quad (4)$$

Koşut programlamadan tam olarak yarar sağlayabilmek için veri boyutunun büyük olması gerekmektedir. Şekil 1.7'de koşut programlamayla işletilmiş bir algoritmanın (Asmuth Bloom Sır Paylaşım Algoritması) hem ardışık hem de koşut olarak işleme süresi yer almaktadır [20].



Şekil 1.7. Koşut ve ardışık programlama yöntemleriyle yürütülmüş örnek bir algoritmanın (Asmuth Bloom Sır Paylaşım Yöntemi) değişik veri boylarındaki çalışma süreleri

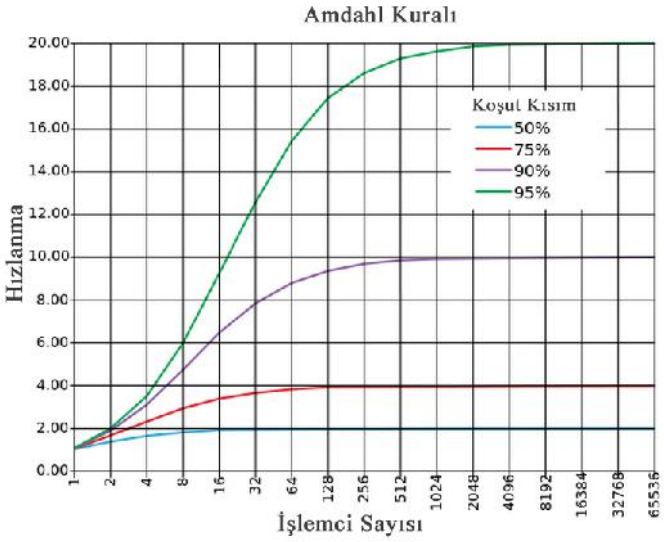
Görüldüğü gibi işlenecek verinin boyutu artıkça koşut programlama ile ardışık programlama arasındaki işlem süresi de artmaktadır. Bu durumun sebebi veri taşıma ve haberleşme zamanıdır. Ardışık programlamada verilerin işlem birimleri arasında taşınması söz konusu olmadığı için koşut programlamadaki haberleşme yükü yoktur. Yani koşut programlamada küçük boyutlu veriler için kısa olan hesaplama süresine haberleşme süresi de gereksiz yere eklenmiş olmaktadır.

1.3.1.3 Veri Bağımlılığı

Veri bağımlılığı hesaplama sırasında herhangi bir adımda ortaya çıkan sonucun sonraki adımlarda girdi olarak kullanılması halinde oluşur. Koşut programlama için olumsuz bir etkendir. Bazı durumlarda koşut programlamayı olanaksız hale getirir.

1.3.1.4. Amdahl Kuralı

Amdahl Kuralı [21] sistemin parçasının hızlandırılması sonucunda ve sistemin bir bütün olarak ele alındığında toplam hızlanmasının ne olacağını hesaplamak için kullanılır. Çoğu zaman birden fazla işlemci kullanıldığında azami hızlanmayı kestirmede kullanılır. Genel olarak Amdahl Kuralı koşut olarak programlanan bir algoritmada işlemci sayısının artışıyla birlikte hızın da artacağını ancak algoritmadan algoritmaya değişen bir işlemci sayısından sonra artışın duracağını öne sürmektedir. Şekil 1.8'de Amdahl Kuralı'na uygun bir hızlanma görülmektedir [20]. .

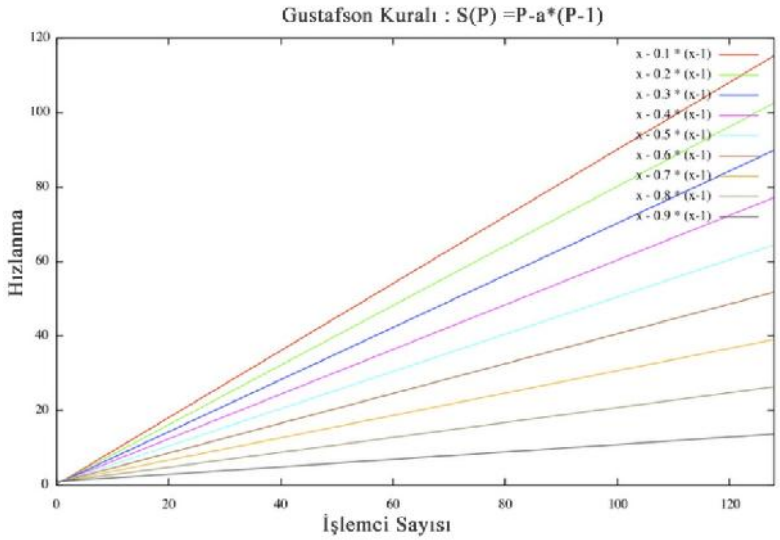


Şekil 1.8 Amdahl Kuralı'na uygun hızlanma

1.3.1.5. Gustafson Kuralı

Gustafson Kuralı[23] yeterince büyük bir sorunun verimli bir şekilde koşutlaştırlabileceğini öne süren bir modeldir. P işlemci sayısı, S hızlanma a' işlemin koşutlaştırlamayan bölümü olmak üzere formülüyle hesaplanır**.**

$$S(P) = P - \alpha.(P - 1) \quad (5)$$



Şekil 1.9 Gustafson Kuralı'na uygun hızlanma

Gustafson Kuralı'na göre hızlanma Şekil 1.9'da görüldüğü tarzda olmalıdır [24]. Gustafson Kuralı iyi bir şekilde koşutluk sağlandıktan sonra işlemci sayısının artışıyla hız artışının da sürekli olacağını öne sürmektedir.

1.3.1.6. Tek Program Çoklu Veri (SPMD)

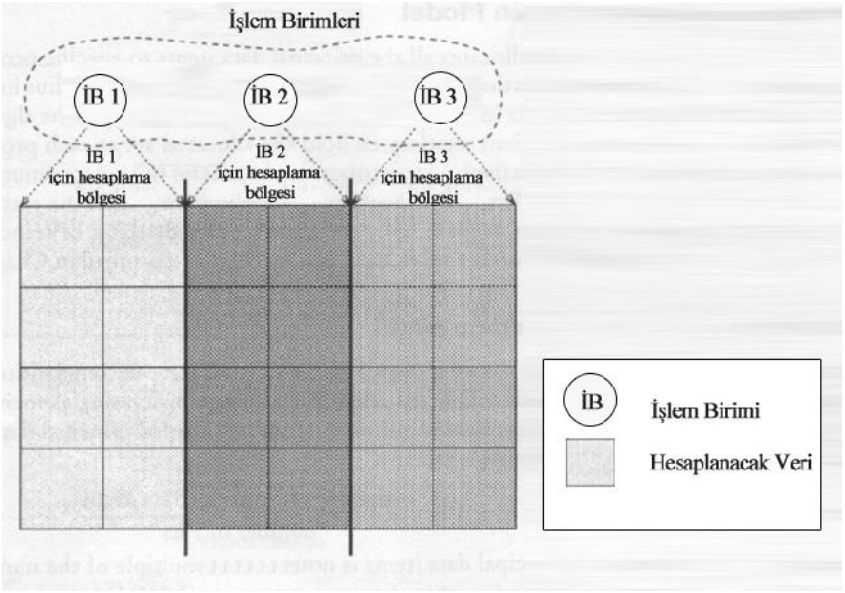
Tek program çoklu veri (single program multiple data - TPÇV) koşut programlamada önemli kavramlardan birisidir. TPÇV mantığında her bir işlem birimi için ayrı ayrı kod yazıp derlemek yerine bütün işlem birimlerinin yapacağı işi içeren tek bir kod derlenerek bütün işlem birimleri üzerinde icra edilir.

1.3.2. Hesaplama Modelleri

Değişen algoritma, işlem birimi sayısı ve hesaplama yüküne göre çeşitli hesaplama modelleri geliştirilmiştir. Bunlar veri güdülen, istek güdülen ve karma hesaplama modelleri olmak üzere üç tanedir.

1.3.2.1. Veri Güdülen Hesaplama Modeli

Veri güdülen hesaplama modeli görevin her bir işlem birimine en başta dağıtılıp hesaplama bittikten sonra sonuçların toplanması esasına dayanır. Bu yönteme Böl ve Yönet yöntemi de denir. Veri Güdülen Hesaplama Modeli şekil 1.10'da gösterilmiştir [21]. Şekilde her bir kare birbirinden ayrı ve bağımsız bir görevi ya da bir diğer deyişle hesaplanacak veriyi temsil etmektedir.



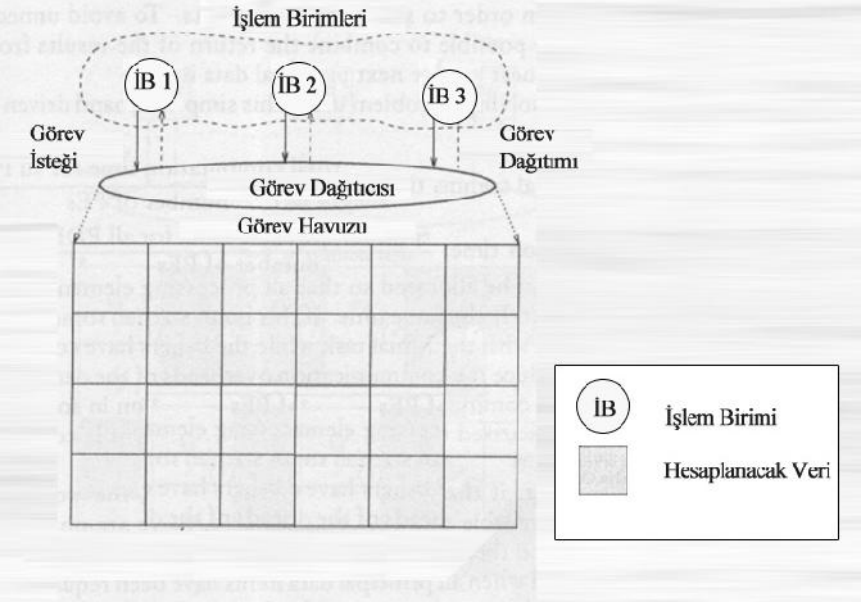
Şekil 1.10. Veri güdülen hesaplama modeli

Bu model hesaplanacak verinin boyutunun önceden bilindiği durumlarda kullanılır. İşlem birimi sayısı da önceden bilindiği için görevler işlem birimine eşit olarak dağıtılır. İşlem birimi güçlerinin eşit olmadığı durumlarda ise her işlem birimine gücüne göre dağıtım yapılabilir fakat bu durum pek faydalı bir çözüm değildir. Her işlem birimi kendine gelen işi sadece bir sefer aldığı, sonucu ya da sonuçları bir sefer gönderdiği ve

hesaplama sırasında diğer işlem birimleriyle iletişim kurmadığı için haberleşme yükü en aza iner.

1.3.2.2. İstek Güdülen Hesaplama Modeli

İstek Güdülen Hesaplama Modeli'nde görev dağıtımı işlem biriminin iş talebine bağlı olarak gerçekleştirilir. Bu yüzden bu yönteme Dinamik Yük Dengeleme [25] adı da verilir. Görevler görev havuzu adı verilen sanal bir birim üzerinde biriktirilir. İşlem birimlerinden birisi ya da birkaçı görev dağıtıcısı olarak belirlenir. Görev dağıtıcısı başlangıç görevlerini görev havuzundan seçerek alt işlem birimlerine gönderir. Daha sonra kendi payına düşen görevi bitiren işlem birimi sonucu merkeze göndererek yeni görev isteğinde bulunur. Eğer görev kaldıysa iş dağıtıcısı yeni görev gönderir ancak görev kalmadıysa iş bitti bildirimi gönderir. Bu yapı şekil 1.11'de [24] anlatılmaktadır.



Şekil 1.11 İstek güdülen hesaplama modeli

Bu modelde hesaplanacak verinin boyutunun önceden bilinmesine gerek yoktur. Her işlem birimi en etkili şekilde kullanılacağından dolayı işlem birimlerinin aynı güçte olmasına da gerek yoktur. Bununla birlikte görev dağıtıcısı ya da alt işlemciler arasında sürekli görev gönderme ve görev isteme yaşanacağından iletişim yükü fazladır.

1.3.2.3. Karma Hesaplama Modeli

Adından da anlaşılacağı üzere istek ve veri sürülen modelleri harmanlayan bir yöntemdir. İki yöntemin artılarını da bünyesinde barındıran bir yöntemdir. Başlangıç görevlerinin bilinip sonraki görevlerin bilinmediği durumlarda kullanılır. Önce veri sürülen modele göre eşit dağıtım yapılır böylece haberleşme maliyeti en aza indirilmiş olur. Daha sonra bilinmeyen görevler geldiğinde de istek sürülen modele geçilerek dinamik dağıtım sağlanır. Bu yöntemin olumsuz yönü ise programlamanın önceki iki yöntemden daha güç olmasıdır.

1.3.3. Kullanılan Donanım ve Teknolojiler

Günümüzde bilgisayarlar üzerinde koşut programlama merkezi işlem birimleri ya da grafik işlem birimleri üzerinde yapılmaktadır.

1.3.3.1. Merkezi İşlem Birimi

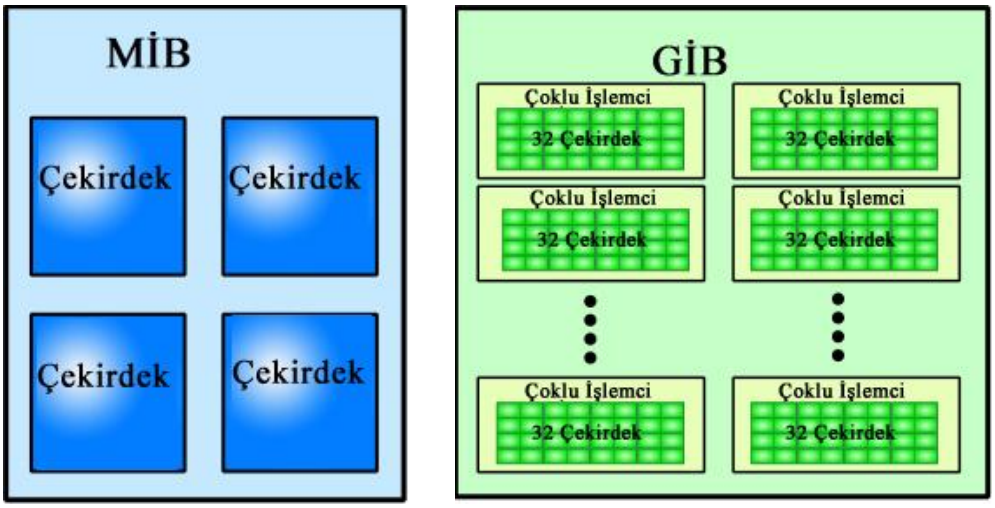
Merkezi işlem birimi (MİB) bilgisayarın beyni olarak adlandırılabilecek bir devredir. Genellikle işlemci deyimi merkezi işlem birimini tanımlamak için kullanılır. Çoklu çekirdek mimarisi de merkezi işlem birimine hız katan bir yapıdır. Günümüzdeki çoğu kişisel bilgisayarın merkezi işlem birimlerinde az sayıda ancak çok güçlü çekirdekler bulunur.

1.3.3.2. Grafik İşlem Birimi

Grafik işlem birimleri (GİB) gelişen oyun teknolojisiyle birlikte merkezi işlem birimlerinin yetersiz kalmalarıyla birlikte bu eksikliği gidermek amacıyla tasarlanmış devrelerdir. Matematiksel işlemlerde MİB'lerinden daha başarılı bir yapıya sahiptirler.

1.3.3.3. MİB ve GİB Karşılaştırması

Şekil 1.12'de temel MİB ve GİB mimarileri karşılaştırmalı olarak yer almaktadır.



(a) (b)

Şekil 1.12 (a) merkezi işlem birimi mimarisi (b) grafik işlem birimi mimarisi

Şekilde de görüldüğü üzere MİB ve GİB mimarileri arasında bir hayli fark vardır. MİB az sayıda ve güçlü çekirdeğe sahipken grafik işlem GİB güçsüz ama çok sayıda çekirdeğe sahiptir. GİB mimarisi gereği koşut programlama çok elverişlidir. GİB'lerin bu yapısı sayesinde günümüzde koşut programlama alanında yükselen bir kullanım grafiğine sahiptir. Tablo 2'de 2010 Kasımı itibariyle dünya üzerinde kullanılan en güçlü beş süper bilgisayar görülmektedir [**26**].

Tablo 2. 2010 Kasım ayı itibariyle dünya üzerindeki en güçlü beş bilgisayar

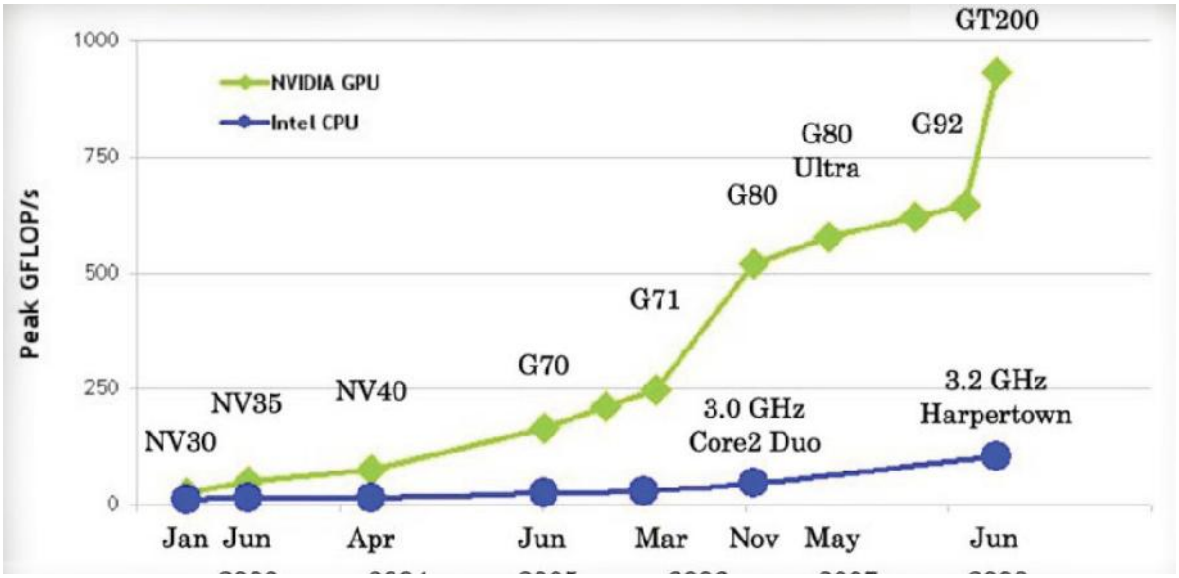
Kullanan Kuruluş	Bilgisayar	Çekirdek Sayısı
National Supercomputing Center in Tianjin China	Tianhe-1A - NUDT TH MPP, X5670 2.93Ghz 6C, NVIDIA GPU, FT-1000 8C / 2010 NUDT	186368
DOE/SC/Oak Ridge National Laboratory United States	Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz / 2009 Cray Inc.	224162
National Supercomputing Centre in Shenzhen (NSCS) China	Nebulae - Dawning TC3600 Blade, Intel X5650, NVidia Tesla C2050 GPU / 2010 Dawning	120640
GSIC Center, Tokyo Institute of Technology Japan	TSUBAME 2.0 - HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows / 2010 NEC/HP	73278
DOE/SC/LBNL/NERSC United States	Hopper - Cray XE6 12-core 2.1 GHz / 2010 Cray Inc.	153408

Listenin bir üç ve dördüncü sırasındaki bilgisayarlar GİB birimi kullanan bilgisayarlardır. 2011 yılı sonu itibariyle dünya üzerindeki en güçlü beş süper bilgisayar incelendiğinde de durumun pek değişmediği fark edilmektedir. Tablo 3'de 2011 yılı sonu itibariyle dünya üzerindeki en güçlü bilgisayarlar görülmektedir.

Tablo 3. 2011 yılı sonu itibariyle dünya üzerindeki en güçlü beş süper bilgisayar

Kullanan Kuruluş	Bilgisayar	Çekirdek Sayısı
RIKEN Advanced Institute for Computational Science (AICS)	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect	705024
National Supercomputing Center in Tianjin	NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050	186368
DOE/SC/Oak Ridge National Laboratory	Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz	224162
National Supercomputing Centre in Shenzhen (NSCS)	Dawning TC3600 Blade System, Xeon X5650 6C 2.66GHz, Infiniband QDR, NVIDIA 2050	120640
GSIC Center, Tokyo Institute of Technology	HP ProLiant SL390s G7 Xeon 6C X5670, Nvidia GPU, Linux/Windows	73278

2011 sonu itibariyle de beş tane bilgisayar içerisinden üç tanesi GİB tabanlıdır. programlama Görüldüğü üzere dünyada GİB ile koşut programlama gittikçe yaygınlaşmaktadır. GİB'nin bir diğer artısı ise performans artışının firmalar tarafından MİB'ne kıyasla daha hızlı bir şekilde yükseltilmesidir. Şekil 1.13'te [**27**]de GİB ve MİB üretiminde küresel ölçekte üretim yapan NVIDIA ve Intel firmalarınca üretilen işlem birimlerindeki hız artışı görülmektedir.



Şekil 1.13 NVIDIA tarafından üretilen grafik işlem birimleriyle Intel tarafından üretilen merkezi işlem birimlerinin hızlarında yıllara göre meydana gelen artış.

1.3.4. Merkezi İşlem Birimi Üzerinde Koşut Programlama

MİB üzerinde koşut programlama uzun zamandır yapıla gelen bir uygulamadır. Bu amaçla geliştirilmiş platformlar arasında MPI (Message Passing Interface) ve PVM (Parallel Virtual Machine) sayılabilir.

1.3.4.1. MPI Teknolojisi

MPI (Message Passing Interface , İleti Geçme Ara Yüzü) merkezi işlem birimi üzerinde koşut programlama yapabilmek için geliştirilmiş C, C++, C#, Java , Python ve FORTRAN dilleriyle birlikte kullanılan bir kütüphanedir. Koşut programlamada yaygın olarak kullanılan tekniklerden birisidir. 1992 yılında profesyonel anlamda çalışmaya başlanan MPI teknolojisini ilk sürümü 1994 yılında bilgisayar dünyasına sunulmuştur. Temel çalışma mantığı işlemcileri haberleştirme üzerin kuruludur. Yüz yirmiden fazla -halen artmaya devam etmektedir- yordama sahip olsa da altı temel yordamla özetlenebilecek bir yapıya sahiptir [25]. MPI ile program geliştirilirken çoğunlukla tek program çoklu veri mantığı göz önüne alınır bu mantığa göre tek bir kod derlenerek bütün işlem birimleri üzerinde icra edilir. Her bir işlem birimi kendisinin hangi numaraya sahip

olduğunu öğrenerek komutları icra etmektedir. Aşağıda rastgele üretilmiş 100 adet sayının en büyüğünü bulan MPI kodu yer almaktadır.

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <conio.h>
#include <time.h>
#include <math.h>


#define baslik1 1
#define baslik2 2
#define sayi 100
MPI_Status durum;
void isci(int islemciNumarasi, int toplamIslemciSayisi)
{

      int gelenSayilar[sayi];
      // 1 numaralı gönderme yordamına karşılık gelen
      // 2 numaralı alma yordamı. Bütün dizi elemanları burada alınıyor...
      MPI_Recv(&gelenSayilar, sayi, MPI_INT, 0,baslik1, MPI_COMM_WORLD,
&durum);

      // toplam işlem birimi saıysı ve şu anda icra edilen
      // işlem birimi numarası göz önünde bulundurularak
      // dizinin hangi aralığındaki en büyük sayının
      // bulunacağı hesaplanıyor
      int pay = (int) (sayi /(toplamIslemciSayisi-1));
    int bas = (islemciNumarasi -1) * pay;
      int son = bas + pay;

      // hesaplanan aralıktaki en büyük
      // sayı bulunuyor
      int eb = gelenSayilar[bas];
      for(int i=bas; i<=son; i++)
            if(gelenSayilar[i]>eb)
                  eb = gelenSayilar[i];

      //en büyük sayı 1 numaralı alma yordamına
      // karşılık olarak göndeliyor. Burada
      MPI_Send(&eb, 1, MPI_INT, 0,baslik2, MPI_COMM_WORLD);
}

int main(int argc, char *argv[])
{
  int i;
  int islemciSayisi, simdiki;
  MPI_Init(&argc, &argv);  // MPI çevresi başlatılıyor
  MPI_Comm_rank(MPI_COMM_WORLD, &simdiki);  // programın hangi işlem birimi
  // üzerinde icra edildği belirleniyor
  MPI_Comm_size(MPI_COMM_WORLD, &islemciSayisi);
  // sistemdeki toplam işlem birimi sayısı belirleniyor
  if(simdiki ==0)  // eğer işlem yapan  ana işlem birimi ise
```

```
  {
  // rasgele 100 tane sayı üretiliyor...
    int sayilar[sayi] ;
      for(int i=0; i<sayi; i++)
            sayilar[i]=rand();
    // üretilen sayılar bütün işlemcilere gönderiliyor...
      // buraya 1 numaralı gönderme yordamı adı verilmiş olsun
      for(int i=1; i<islemciSayisi; i++)
            MPI_Send(&sayilar, sayi, MPI_INT, i, baslik1, MPI_COMM_WORLD);


      int gelenSayi=0;
      int eb = -999999;
      // burada her işlem biriminde hesaplanan en büüyk değerler
      // alınıyor. Gelen değerler arasından en büyüğü dizinin de
      // en büyük elemanı oluyor... bu noktaya 1 numaralı alma
      // yordamı adı veilmiş olsun
      for(int i=1; i<islemciSayisi; i++)
      {
            MPI_Recv(&gelenSayi,1, MPI_INT, i,baslik2, MPI_COMM_WORLD,
&durum);
            if(gelenSayi>eb)
                  eb = gelenSayi;
      }

      printf("Üretilen Sayılar İçerisinde En Büyük Olan %d\n",eb);
  }
  else
  {
       isci(simdiki,islemciSayisi);
  }
     MPI_Finalize();// MPI çevresi sonlandırılıyor
       return 0;

}
```

Kodlar incelendiğinde ana yapının bir C++ kodundan pek de farklı olmadığı görülmektedir. Öncelikle MPI çevresi oluşturulmakta ve bu çevrede yer alan işlem birimi sayısı belirlenmektedir. Bu kodlar aracılığıyla üretilmiş olan programlar bütün işlem birimleri üzerinde aynen icra edilecektir. Her bir işlem birimi üzerinde ‘simdiki’ adlı değişkenin aldığı değer farklı bir tamsayı olacaktır. 0 değerini alan işlem birimi kodun ilk icra edildiği işlem birimidir. Bu işlem birimi ana işlem birimi olma özelliğini taşımaktadır. Bu işlem biriminde rastgele 100 adet sayı üretilmekte ve diğer bütün işlem birimlerine gönderilmektedir. Her bir işlem birimi MPI_Recv yordamıyla kendine gelen sayı dizisini almaktadır. Bloklama yeteneği sayesinde işlem birimi veri almadan işleyişine devam edemez ki böylece güvenli hesaplama gerçekleşmiş olur. Her bir işlem birimi toplam işlem birimi sayısını göz önüne alarak kendilerine düşen aralığı hesaplar ve belirlediği aralıktaki en büyük sayıyı bularak MPI_Send yordamı aracılığıyla ana işlem birimine yollar. Ana

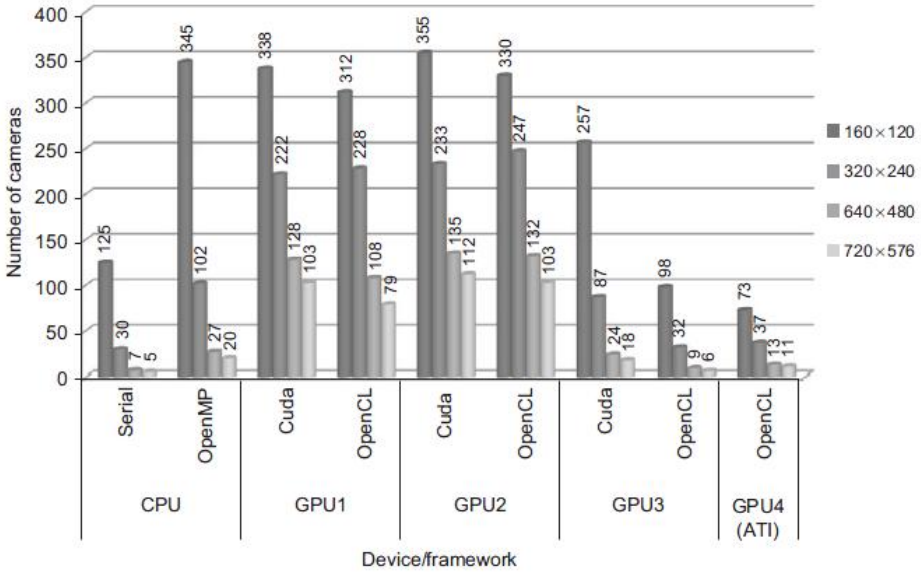
işlem birimi de kendine gelen sayılar içerisinden en büyük sayıyı bularak kullanıcıya bildirir. Koşut programlama mantığı sayesinde ana işlem birimi yüz tane sayı yerine işlem birimi sayısı kadar sayı içerisinden en büyük sayıyı hesaplamış olur.

1.3.4.2. PVM Teknolojisi

PVM (Parallel Virtual Machine, Koşut Sanal Makine) teknolojisi bilgisayarlar arasında koşut ağ kurmaya yarayan bir teknolojidir. En belirgin özelliği Linux ve Windows işletim sistemleri bulunduran makineler arasında taşınabilir ve dağıtık yazılım geliştirme olanağı sağlamasıdır. Bu teknolojide de tıpkı MPI'da olduğu gibi C , C++ ve FORTRAN dilleriyle yazılım geliştirilebilir.

1.3.5. Grafik İşlem Birimi Üzerinde Koşut Programlama

Bölüm 1.3.3.3'de de belirtildiği üzere GİB içerisinde pekçok güçsüz çekirdek barındırır. Zamanla matematiksel işlemler yapmaya son derece elverişli olan ve oyunlar için geliştirilmiş bulunan bu çekirdeklerin diğer hesaplamalar için de kullanılabileceği düşüncesi belirmiş ve bu amaçla bazı teknolojiler ortaya çıkmıştır. Başını NVIDIA firması tarafından geliştirilmiş olan CUDA'nın çektiği bu teknolojiler arasında açık kaynaklı OpenCL ve Microsoft tarafından geliştirilmiş olan DirectCompute teknolojileri de sayılabilir. GİB üzerinde programlama yapmakta kullanılacak teknolojiyi seçmek için birçok etken dikkate alınabilir. Bu etkenler arasında hız, işletim sistemi ve donanım bağımsızlığı, geliştirici desteği, kullanım kolaylığı ve yaygınlık sayılabilir. Bu parametreler arasında en önemli olanı işlem süresi ya da diğer bir deyişle hızdır. Teknolojilerin hızlarını karşılaştırılan çalışmalar arasında Temizel, Halıcı, Logoğlu, Taşkaya Temizel, Ömrüuzun ve Karaman'ın ortak çalışması [28] bu tez kapsamında incelenmiştir. İlgili çalışmada örnek bir görüntü işleme algoritması hem CUDA hem de OpenCL ortamında kodlanmış ve değişik donanımlar üzerinde test edilmiştir. Bu test sonuçlarıyla CUDA ortamında geliştirilen yazılımın OpenCL ortamında geliştirilenden daha hızlı çalıştığı Şekil 1.14'teki grafikte de görüldüğü üzere kanıtlanmıştır[28].



Şekil 1.14 Örnek algoritmanın CUDA ve OpenCL ortamlarında geliştirilen yazılımlarda sonuçlanma süreleri,

Tablo 4'te adı geçen teknolojilerin değişik parametrelere göre değerlendirilmesi görülmektedir.

Tablo 4. GİB üzerinde programlama teknolojilerinin karşılaştırılması

	CUDA	OpenCL	DirectCompute
İşletim Sistemi Bağımsızlığı	Var	Var	Sadece Microsoft Tarafından Geliştirlen İşletim Sistemleri
Donanım Bağımsızlığı	Sadece NVIDIA tarafından üretilen donanımlar üzerinde	Var	Var
Geliştirici Desteği	Yüksek	Orta (açık kaynak)	Düşük
Hız	Yüksek	Orta	Düşük
Kullanım Kolaylığı ve Yaygınlık	Yaygın ve kolay kullanım	CUDA kadar yaygın olmamakla birlikte yaygın	Düşük

Bu parametreler arasında en önemlisi işlem süresi ya da diğer bir adlandırmayla işlem hızdır. Görüldüğü gibi en kısa işlem süresi CUDA teknolojisiyle elde edilmiştir. CUDA sadece donanım bağımlılığı noktasında diğer teknolojilerden geride olup diğer bütün parametrelerde ya rakiplerinden üstün konumdadır ya da onlarla aynı düzeydedir. İşbu sebepten ötürü bu çalışmada CUDA teknolojisi kullanılmıştır.

1.3.5.1. OpenCL Teknolojisi

OpenCL teknolojisi Khronos Grubu tarafından geliştirilmiş ve merkezi işlem birimi ile grafik işlem biriminin eş güdüm içerisinde çalışmasını hedefleyen bir platformdur[29]. Bu grup AMD/ATI, Apple Inc, ARM Holdings, Creative Labs, id Software, Ericsson, Google, Intel Corporation, Motorola, Mozilla,Nokia, Nvidia , Samsung Electronics, Sony Computer Entertainment, Oracle/Sun Microsystems, Texas Instruments gibi dev firmaların oluşturduğu bir ortaklıktır[30]. Bununla birlikte işbu grupta başı AMD firması çekmektedir. OpenCL teknolojisinin en önemli tarafı işletim sistemi ve donanımdan bağımsız olmasıdır. Bununla birlikte NVIDIA tarafından geliştirilmekte olan CUDA teknolojisi kadar hızlı bir gelişim grafiğine sahip olduğu söylenemez.

1.3.5.2. Direct Compute Teknolojisi

Direct Compute Microsoft tarafından geliştirilmiş bir koşut programlama teknolojisidir. GİB üzerinde koşut programlama sağlayan bu sistemin en büyük artısı donanımdan bağımsız olmasıdır. Fakat sadece Microsoft firmasında geliştirilen işletim sistemlerinde çalıştırılabilir olması, gelişim grafiğinin yavaş seyretmesi ve performanstaki yavaşlık bu teknolojinin yazılın sektöründeki kullanım oranını kısıtlamaktadır.

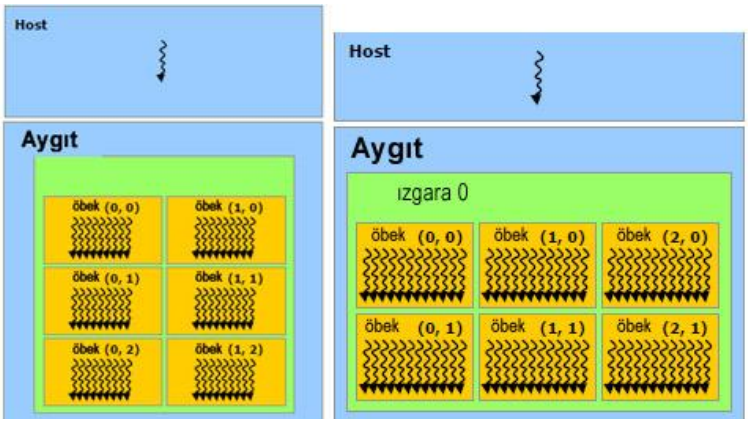
1.3.5.3. CUDA Teknolojisi

CUDA (Compute Unified Device Architecture, Bütünleşik Aygıt İşleme Mimarisi); NVIDIA firması tarafından üretilen, ölçeklendirilebilir koşut programlama modelidir. Çok çekirdekli grafik işlemcileri kullanarak genel amaçlı hesaplama işlemlerinin yapılmasını sağlamak için geliştirilmiştir [20]. CUDA, geliştirici firma tarafından üzerinde önemle durulan ve sürekli yenilenen bir teknolojidir. Bu geliştirmeler iyi bir değişiklik olsa da

sürekli değişiklikler yazılımcıları sıkıntıya sokmaktadır. İşbu teknoloji sadece NVIDIA tarafından üretilen merkezi işlem birimleri üzerinde çalışabilmektedir. Bellek yetersizliği ve ardışık programlamaya göre daha güç programlanması da CUDA teknolojisinin olumsuz yönleri arasındadır. Bununla birlikte sürekli geliştirilen CUDA'da programlama güçlüğü her yeni sürümde daha da azaltılmaktadır. Kullanılan donanımın değişkenliğine göre hesaplamadan elde edilen verim de değişmektedir. NVIDIA firması tarafından, üretilen her grafik işlem birimine hesaplama yeteneği olarak adlandırılan bir değer verilmektedir. Bu çalışmada hesaplama yeteneği 1.1 olan NVIDIA GeForce GT 130M ve hesaplama yeteneği 2.1olan NVIDIA GeForce G5 540M model donanımlar kullanılmıştır. Elde edilen sonuçlar verilirken NVIDIA GeForce GT model donanımdan elde edilen sonuçlar 'HY 1.1' NVIDIA GeForce G5 540M model donanımdan elde edilen sonuçlar 'HY 2.1' olarak adlandırılacaktır.

CUDA teknolojisinde Grafik işlem birimi 'aygıt' olarak adlandırılır. Aygıt üzerinde çok sayıda çekirdek yer almaktadır. Bu çekirdeklerin her biri merkezi işlem birimi çekirdeklerine kıyasla çok güçsüz olsa da bir araya geldiklerinde matematiksel işlemler başta olmak üzere pekçok işlemde MİB'nden daha hızlı çalışabilmektedirler zaten CUDA mimarisinde hedeflenen de tam olarak budur. CUDA'da yazılan bir işlev tüm iş parcacıkları (iplikler) üzerinde eş zamanlı olarak yürütülür. Bu işlevlere 'kernel' ya da Türkçe ifadesiyle 'çekirdek' adı verilir.

CUDA mimarisine göre iş parçacıkları (iplik) birleşerek öbekleri (block) , öbekler de birleşerek ızgarayı(grid) meydana getirir. Izgaralar da birleşerek aygıtı oluşturur. Ayrıca 32 iş parçacığından oluşan birime çözgü (warp) adı verilir. İşbu yapı Şekil 1.15'te [32] gösterilmiştir. Her iş parçacığı üzerinde aynı komut icra edilir ki bu yapısıyla CUDA tek program çok veri mantığıyla hareket etmiş olur. Her iş parçacığı, blok ve ızgaranın eşsiz kimliği vardır. Yazılım geliştirilirken verinin hangi çekirdek tarafından işleneceğini belirlemek için bu kimlikler kullanılır ki programlamanın en güç tarafı bu kimlikleri doğru olarak kullanabilmektir.

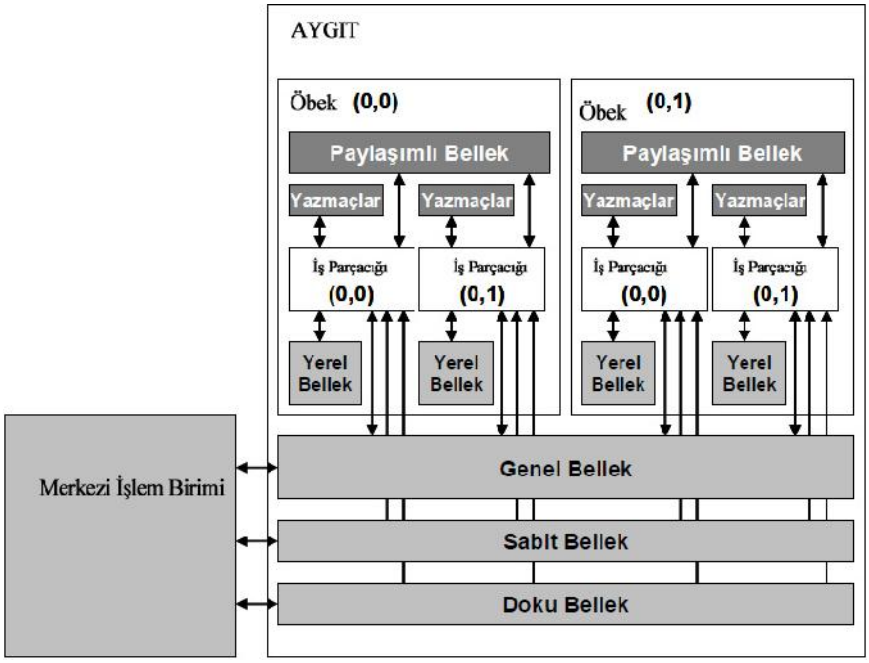


Şekil 1.15. CUDA mimarisi

CUDA kullanılırken karşılaşılan güçlüklerden biri de bellek sorunudur. Aygıt belleği host belleğine göre daha küçüktür ayrıca (GİB) üzerinde giriş/çıkış işlemleri merkezi işlem birimine göre daha yavaştır. Bu sebeplerden ötürü program verileri çoğunlukla merkezi işlem birimi tarafından alınır ve aygıt üzerinde gerektiği kadar yer açılıp aygıta aktarılır. Doğal olarak bu süreç de zaman alır.

1.3.5.3.1. CUDA Bellek Çeşitleri

CUDA üzerinde yer alan belli başlı bellek türleri aşağıda açıklanmıştır. Bu bellek yapıları ulaşabildikleri noktalara göre bütün halinde Şekil 1.16'da [25] görülmektedir.



Şekil 1.16 CUDA bellek mimarisi

1.3.5.3.1.1. Genel Bellek

Adından da anlaşılabileceği üzere en yaygın bellek türüdür. Doğrudan DRAM üzerinde yer alır. Merkezi işlem birimi tarafından ulaşılabilen bir bellek olması neredeyse her programda kullanılmasını zorunlu kılar. Farklı öbek, ızgara ve iş parçacıkları tarafından da ulaşılabilir. Hem okunup hem de yazılabilir. Uzun gecikme süresine sahip olması bu bellek türünün önemli bir dezavantajıdır. Bu yüzden verinin çok daha hızlı erişilebilen paylaşımlı bellek üzerine taşınıp oradan işlenmesi salık verilen bir çözümdür.

1.3.5.3.1.2. Paylaşımlı Bellek

Aynı öbek üzerinde bulunan iş parçacıkları tarafıdan erişilebilir. İlgili iş parçacıkları tarafından veri paylaşımını sağlar. Büyüklüğü kısıtlı olmasına karşın erişim hızı gayet yüksektir. Yonga içi bir bellek olduğu için yerel ve genel belleklerden daha hızlıdır. Bu nitelikleri sayesinde veri tamponlamak için sıklıkla başvurulan bir bellek çeşididir. Çoğu zaman ham veri genel bellekten paylaşımlı belleğe aktarılır, burada elde edilen sonuçlar tekrar genel belleğe döndürülür.

1.3.5.3.1.3. Sabit Bellek

Sabit bellek de genel bellek gibi DRAM üzerinde yer alır. Erişim hızı yavaştır. Bunula birlikte tamponlanabilir. Bu da sadece okunan veriler üzerinde olumlu bir özelliktir.

1.3.5.3.1.4. Yerel Bellek

Yerel bellekler öbekler içerisinde iş parçacıklarıyla iletişim kurabilmek için geliştirilmiştir. Yavaş bir bellektir.

1.3.5.3.1.5. Doku Bellek

Doku bellek bir çok açıdan sabit belleğe benzer o da sadece okunan veriler için avantaj sağlar. Sabit bellekle birlikte daha çok grafik işlemlerinde kullanılırlar.

1.3.5.3.1.6. Yazmaçlar

Yerel belleklerin yavaşlığı dolayısıyla geliştirilmiş hızlı belleklerdir. Ancak her bir öbekteki yazmaç miktarının sabit olması iş parçacığı arttıkça parçacık başına yazmaç miktarının azalmasına yol açar.

1.3.5.3.2. CUDA Programlama Ara Yüzü ve Performans

CUDA ile programlama yapabilmek için C programlama dili uzantısı olarak yazılım geliştirme ortamlarıyla bütünleştirlebilen (Visual Studio gibi) ve bizzat NVIDIA tarafından üretilen CUDA sürücü programlama arayüzü olmak üzere tip arayüz vardır. CUDA programlama modeli, C diline uygun bir ifade kümesi içermektedir. Bu şekilde programlamacıların koşut programlama için gerekli işlevleri tanımlaması amaçlanmıştır. Ek olarak, yazılım geliştirirken aygıt üzerindeki belleğin yönetimi, bilgisayar ile ekran kartı arasındaki veri iletişimi, birden fazla aygıtın yönetilmesi gibi gereksinimler için özel işlevler tanımlanmıştır. C dili ile programlama yapılırken bu işlevlerin bulunduğu kütüphane (CUDA Runtime API) kullanılabilir [26]. CUDA ortamında yazılım geliştirilirken işlevleri ve değişkenleri niteleyen niteleyeciler kullanılır. Bu niteleyiciler

ilgili kod parçasının nerede çalışacağını ve kod parçasına nereden ulaşılabileceğini gösterir. Niteleyiciler Tablo 5'te gösterilmektedir.

Tablo 5. CUDA'da kullanılan niteleyiciler

Niteleyici Türü	Adı	Açıklama
İşlev niteleyicisi	__device__	Sadece aygıt üzerinde yer alır
İşlev niteleyicisi	__global__	Hem aygıt hem de merkezi işlem birimi üzerinde işler
İşlev niteleyicisi	__host__	Sadece merkezi işlem birimi üzerinde işler
Değişken niteleyicisi	__device__	Genel bellek üzerinde saklanır. Her iş parçacığının ulaşımına ve üzerine yazmasına açıktır.
Değişken niteleyicisi	__shared__	Paylaşımlı bellek üzerinde saklanır ve sadece ilgili öbek içerisinden erişime açıktır

CUDA mimarisi önceki bölümlerde de belirtildiği üzere çeşitli kodlama zorluklarını da beraberinde getirir. Hız kazanmak amacıyla paylaşımlı bellek kullanmak bu güçlükleri daha da artırmaktadır. Aşağıda paylaşımlı bellek kullanmadan sadece genel bellek üzerinde ortanca süzgeci gezdirme işlemi için grafik işlem biriminin hazırlanmasına ve merkezi işlem biriminden veri aktarmaya yarayan kod parçası yer almaktadır.

```
 void ortanca( float *resim ,int gen, int yuk, int M, int N)
{
	int boy  = gen * yuk;
	float *ay_resim ;
	float *ay_sonuc;
	int boyut = sizeof(float) * boy;
	cudaMalloc((void**)&ay_resim, boyut);
	cudaMalloc((void**)&ay_sonuc, boyut);
	cudaMemcpy(ay_resim, resim, boyut, cudaMemcpyHostToDevice);
	 ortanca3x3<<<M,N>>>(ay_resim, ay_sonuc,boy,  gen, yuk);
	cudaMemcpy( resim, ay_sonuc, boyut, cudaMemcpyDeviceToHost);
	cudaFree(ay_resim);
	cudaFree(ay_sonuc);
}
```

Bu koda göre kullanıcının gönderdiği resim GİB'nde ay_resim adıyla, süzme işlemi sonucu elde edilen görüntü de ay_sonuc adıyla yine GİB'nde saklanacaktır. 'cudaMalloc' yordamıyla ay_resim ve ay_sonuc işaretçilerine GİB'nde boş alan açılmaktadır. cudaMemcpy yordamı ile MİB üzerinde yer alan resim işaretcisinin gösterdiği veriler GİB üzerine aktarılmaktadır. Daha sonra ortanca3x3 adlı çekirdek yordamıyla GİB üzerinde veri işlenmektedir. İşbu yordamın söz diziminin klasik C ve C++ kodlarından farklı olduğu hemen göze çarpacaktır. '<<<,>>>' ifaresi içerisinde kodun kaç öbek ve kaç iş parçacığı içerisinde icra edileceği görülmektedir. İş parçacıklarının öbekler içerisinde bulunduğuna daha önce değinilmişti. Dolayısıyla toplamda öbek sayısı x iş parçacığı sayısı kadar iş parçacığı üzerinde kod icra edilmiş olmaktadır. Söz gelimi <<<10,20>>> ifadesi için 10 x 20 = 200 tane iş parçacığı çalışmış olur. Çekirdek işlevi icra edildikten sonra cudaMemcpy yordamıyla GİB üzerindeki veriler MİB üzerine aktarılmış olur. 'cudaFree' yordamlarıyla GİB üzerinde ayrılmış olan yer iade edilir. 'ortanca3x3' çekirdeği içerisindeki kodlar aşağıda görülmektedir.

```
#define indis(a,b,c) ( a* b + c)
#define siraBul(a,b,c,d) ( a + b * c * d)

__device__ float deger(float *resim, int indis, int boyut)
{
	if(indis>= 0 && indis<boyut)
		return resim[indis];
	else
		return 0;
}

__global__ void ortanca3x3(float *resim , float *sonuc,  int boy, int gen, int yuk)
{
	unsigned int i = indis(blockIdx.x,blockDim.x , threadIdx.x);
	unsigned int j = indis(blockIdx.y,blockDim.y , threadIdx.y);
	int sira = siraBul(i , j, blockDim.x , gridDim.x);
	float dizi[9];
	if(sira>=0 && sira< boy)
	  {
		dizi[0] =  deger(resim, sira, boy);
		dizi[1] =  deger(resim, sira - 1, boy);
		dizi[2] =  deger(resim, sira + 1,boy);
		dizi[3] =  deger(resim, sira - yuk, boy);
		dizi[4] =  deger(resim, sira + yuk, boy);
		dizi[5] =  deger(resim, sira - yuk-1,boy);
		dizi[6] =  deger(resim, sira - yuk+1, boy);
		dizi[7] =  deger(resim, sira + yuk-1,boy);
		dizi[8] =  deger(resim, sira + yuk+1, boy);

		for(int i=0; i<8; i++) // kabarcık sıralamasıyla sayılar sıralanıyor
			for(int j=i+1; j<9; j++)
				if(dizi[i]> dizi[j])
				{
					float ara = dizi[j];
					dizi[j]= dizi[i];
					dizi[i] = ara;
				}
		sonuc[sira] = dizi[4];
	}
	 else
		return ;
		__syncthreads();
}
```

Kod içersinde yer alan ‘blockIdx’, ‘threadIdx’ gibi ifadelerin farklılığı hemen dikkat çekecektir. Bu yapılar çekirdeğin üzerinde çalıştığı iş parçacığına ait bilgileri taşıyan ifadelerdir. Bu bilgilerden yola çıkılarak hangi piksel işlem üzerinde işlem yapılması gerektiği hesaplanmaktadır. Örneğin çalışan grafik işlem birimi üzerinde 512 tane ızgara, her ızgara üzerinde 256 tane öbek (x ve y eksenlerinde) ve her bir öbekte 128 iş parçacığı (x ve y eksenlerinde) yer alsın. 0 numaralı öbeğin 100 numaralı iş parçacığı için i = 0 *

256 + 100 = 100 ve j = 0* 256 + 100 = 100 değerini alır. Bu durumda sıra = 100 + 100 * 256 * 128 = 3276900 değerini alır. Buna göre ortanca3x3 adlı yordam görüntü üzerindeki 3276900. numaralı piksel üzerinde işlem yapacaktır. ‘deger’ işlevi gelen bilgilere göre işaretçinin resmin değerini döndürmektedir. Parametre olarak gelen sayı resmin boyutları içerisinde ise resmin ilgili pikseline ait değeri döndürmekte yoksa 0 değerini döndürmektedir. Bu sayede görüntünün kenarları için 0 doldurma işlemi de kendiliğinden yapılmış olmaktadır. Daha sonra kabarcık sıralaması ile ortanca süzgeci çekirdeği içerisindeki değerler sıralanmakta ve ortadaki değer sonuç işaretçisine aktarılmaktadır. Burada bir tamponlama işlemi yapılmadığı için sonuç değerlerinin yine aynı büyüklükte fakat ayrı bir işaretçi içerisinde tutulması gerekmiştir. Bu da hem ek bellek maliyeti hem de sadece genel bellek üzerinde işlem yapılıp tamponlamaya başvurulmadığı için işlem yavaşlığı anlamına gelir.

Aynı işlemin tamponlama yapılarak icra edilmesi ise biraz daha farklıdır. Aşağıda paylaşımlı bellek yapmak için gerekli kodlar verilmiştir.

```
void  ortancaSuzgeci(float *resim, int gen, int yuk, int parca)
{
  float *ay_resim;
  cudaMalloc((void**)&ay_resim, sizeof(float) * gen * yuk);
  cudaMemcpy(ay_resim, resim, sizeof(float) * gen * yuk, cudaMemcpyHostToDevice);

  dim3 grids(yuk/parca,gen/parca);
  dim3 threads(parca,parca);

  int boy = gen * yuk;
  ortanca3x3Paylasimli<<<grids, threads>>>(ay_resim, gen, yuk);
  cudaMemcpy(resim, ay_resim, sizeof(float) * gen * yuk, cudaMemcpyDeviceToHost);
  cudaFree(ay_resim);
}
```

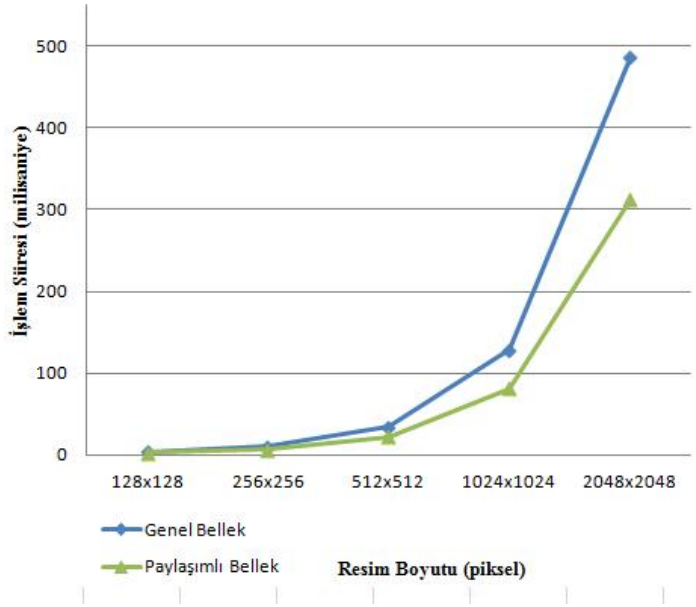
Görüldüğü gibi kodlama genel bellek için programla ile hemen hemen aynıdır fakat aradaki en önemli fark çekirdek işlevi çağırırken grids ve threads adlı değişkenlerdir bu çekirdek işlevin CUDA’ya ait bir yapı olan dim3’e özgü şekilde çalışmasına olanak verir. ‘dim3’ yapısı x, y ve z adlı üç tane işaretsiz tam sayıdan oluşan bir yapıdır ve öbek boyutlarını belirtir. Tamponlama yaparak ortanca süzgeci gezdiren kod aşağıda verilmiştir.

```
#define BLOCK_W 8
#define BLOCK_H 8
// paylaşımlı bellek yardımıyla tamponlama ile ortanca süzgeci gezdirme

 __global__ void ortanca3x3Paylasimli( float *resim, int gen, int yuk)
{
	__shared__ float cerceve[BLOCK_W*BLOCK_H][9];

	unsigned int x=blockIdx.x*blockDim.x+threadIdx.x;
	unsigned int y=blockIdx.y*blockDim.y+threadIdx.y;
	unsigned int tid=threadIdx.y*blockDim.y+threadIdx.x;

	if(x>=gen && y>=yuk)
		return;

	cerceve[tid][0]= (y==0||x==0)?0.0f:resim[(y-1)*gen+x-1];
	cerceve[tid][1]= (y==0)?0.0f:resim[(y-1)*gen+x];
	cerceve[tid][2]= (y==0||x==gen-1)?0.0f:resim[(y-1)*gen+x+1];
	cerceve[tid][3]= (x==0)?0.0f:resim[y*gen+x-1];
	cerceve[tid][4]= resim[y*gen+x];
	cerceve[tid][5]= (x==gen-1)?0.0f:resim[y*gen+x+1];
	cerceve[tid][6]= (y==yuk-1||x==0)?0.0f:resim[(y+1)*gen+x-1];
	cerceve[tid][7]= (y==yuk-1)?0.0f:resim[(y+1)*gen+x];
	cerceve[tid][8]= (y==yuk-1||x==gen-1)?0.0f:resim[(y+1)*gen+x+1];

	__syncthreads();

	for ( int j=0; j<5; ++j)
	{
		for ( int l=j+1; l<9; ++l)
			if (cerceve[tid][l] < cerceve[tid][j])
			{
				const float ara=cerceve[tid][j];
				cerceve[tid][j]=cerceve[tid][l];
				cerceve[tid][l]=ara;
			}
		__syncthreads();
	}
	resim[y*gen + x]=cerceve[tid][4];
}
```

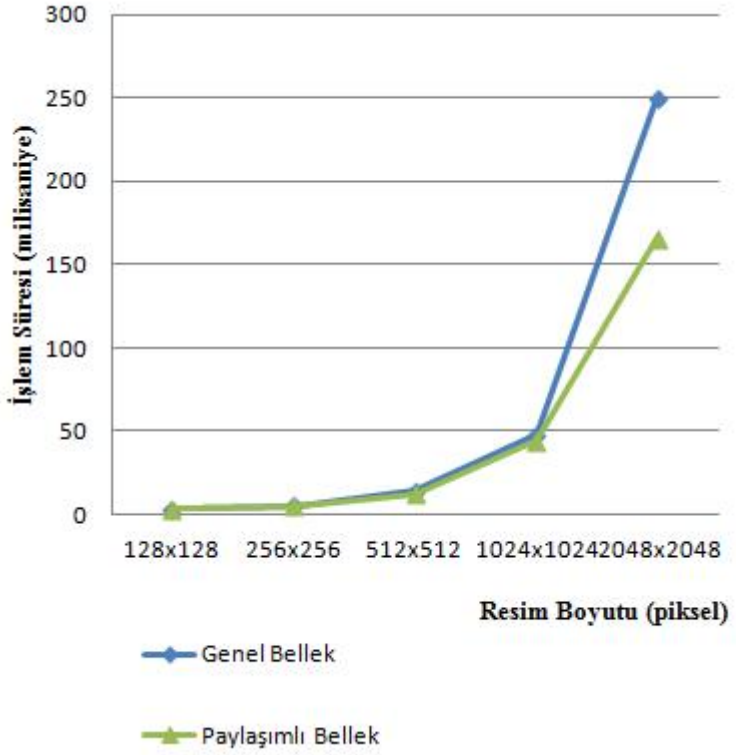
Burada '*__shared__*' niteleyicisi ile birlikte paylaşımlı bellek üzerinde dizi tanımlanmaktadır. Bir paylaşımlı bellek bölgesine sadece ortak öbeğe ait iş parçacıklarının ayrıldığı daha önce belirtilmişti. Dolayısıyla çekirdek işlevin işletildiği bütün iş parçacıklarının aynı paylaşımlı bellek bölgesine ulaşmaları beklenemez. Aynı öbekte yer alan iş parçacıklarının da aynı paylaşımlı bellek bölgesi üzerinde işlem yapması da algoritmanın yanlış çalışmasına yol açacaktır. Bu yüzden paylaşımlı bellek üzerinde iki boyutlu bir dizi ayrılarak her bir parçacığın sadece kendisiyle ilgili kısımlara veri yazması ve oradan veri okuması sağlanmıştır. Daha sonra yapılan işlemler genel bellek kullanan

koddakiyle hemen hemen aynı işlemlerdir. Tek fark verilerin genel belleğe değil de paylaşımlı bellek üzerinde ayrılan dizinini ilgili satır ve sütununa yazmasıdır. Son olarak kabarcık sıralaması algoritmasıyla sıralama yapılarak ortada kalan değer genel belleğe aktarılmıştır.

Paylaşımlı bellek kullanmanın faydasının ölçüldüğü deneyde yukarıda verilen kodlar hem 1.1 hesaplama yeteneğine sahip aygıt hem de 2.1 hesaplama yeteneğine sahip aygıt üzerinde ayrı ayrı icra edilmiştir. Şekil 1.17 ve Şekil 1.18'de bu hesaplama sonuçları grafikler halinde gösterilmektedir.



Şekil 1.17 Ortanca süzgeci algoritmasının hesaplama yeteneği 1.1 olan bir aygıt üzerinde icra edilirken paylaşımlı bellek kullanldığında ve kullanılmadığında geçen işlem süreleri

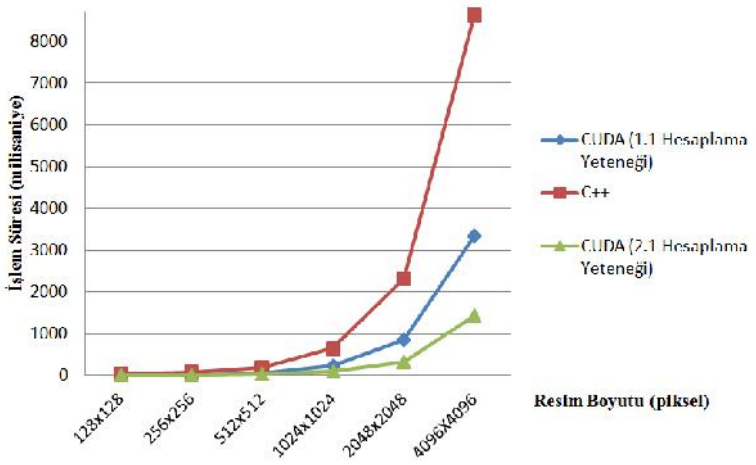


Şekil 1.18 Ortanca süzgeci algoritmasının hesaplama yeteneği 2.1 olan bir aygıt üzerinde icra edilirken paylaşımlı bellek kullanıldığında ve kullanılmadığında geçen işlem süreleri

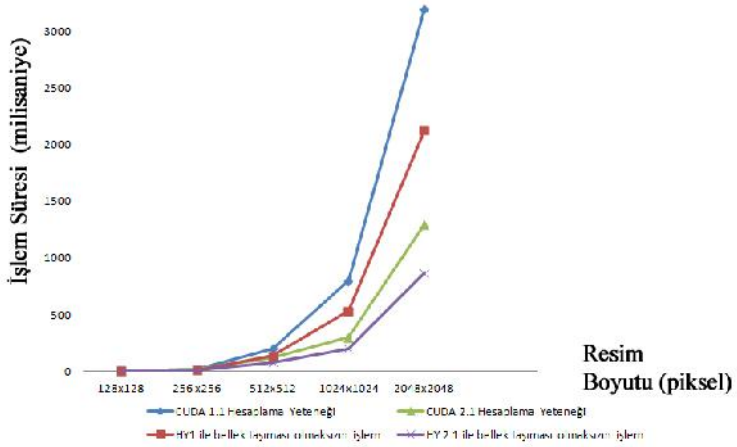
Görüldüğü gibi paylaşımlı bellek üzerinde işlem yapılması tamponlama avantajı sayesinde performans artışı sağlamaktadır. CUDA platformunda kodlamanın güçlüğünü incelemek amacıyla CUDA kodları klasik C++ kodlarıyla karşılaştırılmıştır. Aşağıda süzgecinden geçirme işleminin merkezi işlem birimi üzerinde C++ dilinin olanaklarıyla yapılması durumunda kullanılan kod verilmiştir.

```
#define sira(a,b,c)(a+ b* c)

void ortanca(float *kaynak, int resimGen, int resimYuk,int suzgecBoyu)
{
float *sonuc = new float[resimYuk * resimGen];
for(int i=0; i<resimGen; i++)
     for(int j=0; j<resimYuk; j++)

       {
float *dizi = new float[8];
int sayac = 0;
// bir piksele komşu 8 piksel sırayla taranıp diziye aktarılıyor
for(int k=-1; k<= 1; k++)
   for(int l=-1; l<= 1; l++)
      {
        dizi[sayac]= kaynak[indis(i+k, j+l,resimYuk)];
        sayac++;
       }
// kabarcık sıralamasına göre sayılar sıralanıyor
for(int k=0; k<5; k++)
     for(int l=k+1; l<9; l++)
          if(dizi[k]> dizi[l])
            {
            float ara = dizi[k];
            dizi[k]= dizi[l];
            dizi[l] = ara;
            }
        sonuc[indis(i+k, j+l,resimYuk)] = dizi[4];
    }
}
```

Burada resim genişlik ve yükseklik değerlerine göre taranır ve görüntünün üzerinde ortanca süzgeci gezdirilmiş olur. İşi, tek bir merkezi işlem birimi bütün pikseller üzerinde sırasıyla yapar. Görüldüğü üzere ardışık işlem yapan bir dille (burada C++ kullanıldı) kod yazmak CUDA ile kod yazmaktan çok daha kolay ve risksizdir. Platformlar hız bakımından değerlendirildiğinde ise CUDA ile koşut programlama bu örnekler için hız bakımından bariz verimli görülmektedir. Ayrıca CUDA ortamında yazılım geliştirme sırasında verilerin MİB belleğinden GİB belleğine taşınması da zaman almaktadır. Şekil 1.19'da 1.1 ve 2.1 hesaplama yeteneğine sahip aygıtlarlar kaydedilen işlem süresi ile ardışık programlamada kaydedilen işlem süreleri ile bellekler arası veri taşınmadığı zaman gerçekleşen işlem süreleri görülmektedir.



(a)



(b)

Şekil 1.19. (a)Algoritmanın değişik teknolojilerdeki gerçekleşme Süreleri
(b) CUDA teknolojisinde belleğe veri taşıma olmaksızın hesaplama süreleri

Şekil 1.12'de ortanca alma işleminin değişik donanım ve algoritmalarda yürütüldüğünde elde edilen sonuçlar, Tablo 6'da ise HY 1.1 ve HY 2.1 ile elde edilen sonuçların ardışık işleme göre zaman kazancı oransal olarak yer almaktadır. Bu ölçümlerde elde edilen bulgulara göre işlenen verinin boyutu büyüdükçe alınan verimin de arttığı görülmektedir. Uygun bir grafik işlem birimi teknolojisi kullanıldığı zaman işlem süresi 8 kata kadar kısalabilmektedir.

Tablo 6. Değişik hesaplama yeteneğine sahip aygıtlarda elde edilen zaman kazançları

Teknoloji /Veri Boyu	128x128	256x256	512x512	1024x1024	2048x2048	4096x4096
H.Y 1.1 /C++	4,83	4,68	3,1	2,85	2,67	2,609
H.Y 2.1 / C++	5,1	5,0	6,3	8,09	7,03	7,96

1.3.6. Kullanılan Donanıma Göre Koşut Programla Teknolojilerini Karş.

Son yıllarda GİB ve MİB'nin performanslarını karşılaştırmak için önemli çalışmalar [33,34,35] yapılmıştır. Bu incelemelerde genelde görüntü işleme alanında sık kullanılan algoritmalardan birkaçı seçilmiş ve değişik GİB mimarilerine sahip aygıtlar üzerinde performans kıyaslanmıştır. [34] gibi bazılarında ise GİB ve MİB lerin ortak kullanıldığı karma çalışmalar üzerinde durulmuştur. Karma işlem yapabilen sistemlerin iki teknolojinin de gelişmiş yönlerini bünyesinde barındırarak önemli bir avantaj elde ettikleri görülmektedir. Bugün GİB kullanılarak yapılan işlemlerin çoğunda ilk verilerin MİB tarafından okunarak GİB'ne iletildiği bilinen bir gerçektir. [35] ise bu taşıma işlemi sırasında harcanan zamanı ve bellek verimli kullanıldığında elde edilen kazancı göstermesi bakımından dikkate değer bir çalışmadır.

Bütün bu veriler ışığında örnek bir GİB üzerinde koşut program yürütmeyi sağlayan teknolojiler (GİBT) ile MİB üzerinde koşut program yürütmeyi sağlayan teknolojiler (MİBT) karşılaştırılmış ve şöyle sonuçlar elde edilmiştir. MİBT'leri istek güdülen model başta olmak üzere değişik hesaplama modellerine GİB teknolojilerinden daha elverişlidir. GİB teknolojileri mükemmel derecede koşutlaştırlabilen algoritmalar üzerinde daha başarılıdır. MİB teknolojileri programlama kolaylığı açısından GİB teknolojilerinden daha üstün konumdadır çünkü yıllardır kullanılan programlama dilleri MİB üzerinde işlem yapar. İlgili teknolojiler bu diller üzerine yeni komutlar eklenmesiyle oluşurlar ve C, C++, FORTRAN gibi dillerde program geliştiren insanlar için öğrenmesi kolay teknolojilerdir. GİB teknolojileri ise birinci gruba göre çok sonradan yaygınlaşan platformlardır ve söz dizimleri MPI, PVM gibi teknolojilere kıyasla daha karmaşıktır. Maliyet açısından

incelendiğinde ise GİB teknolojilerin daha üstün olduğu görülmektedir. MİB'leri GİB'lerine göre daha ucuzdur ve MPI, PVM gibi teknolojileri kullanmak için çok sayıda işlemciye gerek duyulmaktadır. GİB üreten firmaların ürünlerinin yaygınlaşmasını sağlamak istemeleri CUDA, OpenCL gibi teknolojilerin arkasında güçlü bir üretici desteğinin doğmasına yol açmıştır. Tablo 7'da MİB ve GİB teknolojilerinin ana hatlarıyla karşılaştırılması görülmektedir.

Tablo7. MİB ve GİB üzerinde koşut programlamada kullanılan teknolojilerin karşılaştırılması

	MİB Kullanan Teknolıjiler	GİB Kullanan Teknolojiler
Değişik Hesaplama Modellerine Uygunluk	Yüksek	Düşük. (Veri güdülen modele daha uygun)
Maliyet	Pahalı	Ucuz
Programlama Kolaylığı	Kolay	Güç
Geliştirici Desteği	Orta	Yüksek
Mükemmel Derecede Koşutlaştırılabilen Algoritmalardaki Hız	Orta	Yüksek

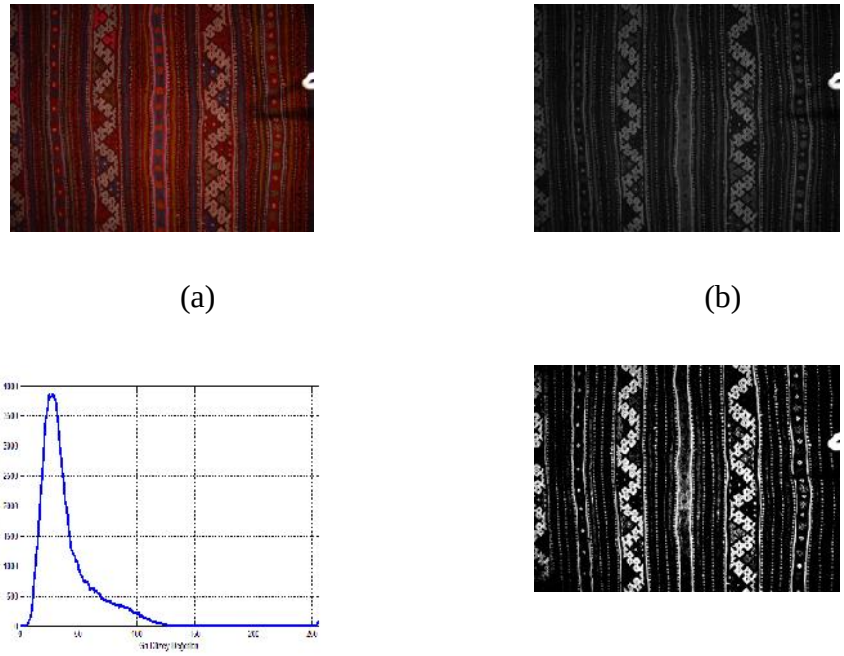
Sonuç olarak eğer tek komutun çok veri üzerinde uygulanabileceği bir algoritma yürütülmek isteniyorsa GİB teknolojilerine, dinamik yük dengelemenin esas alındığı bir algoritma icra edilmek isteniyorsa MİB teknolojilerine başvurulmalıdır.

1.4. Otsu Eşik Değer Bulma Algoritması

Gri düzeyli görüntüyü ikili görüntüye dönüştürme işlemi cisim tanımada önemli aşamalardan birisidir. İkili resme dönüştürme diğer adıyla görüntü eşikleme işlemi genellikle görüntü üzerindeki bölgelerin belirlenmesinde kullanılır. Dizinde eşikleme işlemini yapan birçok algoritma vardır. Bu algoritmalar arasında eşik değerin pikselin

komşu piksellerinine göre yerel olarak hesaplandığı Nibblack Yöntemi [36], histogramın entropisine göre genel bir eşik değer öneren Kapur Yöntemi [37], yumuşatma işlemine dayalı olan Bütünleşik İşlevler [38] yöntemleri anılabilir. Bu çalışmada sık kullanılan eşikleme yöntemlerinden birisi olan Otsu algoritması [39] kullanılmıştır. İlgili algoritma resim üzerindeki bölgeler belirlendikten sonra gürültü olabilecek bölgelerin elenmesi sırasında da kullanılmıştır.

Otsu Algoritması görüntülerin ön plan ve arka plandan oluşan iki ayrı öbekten oluştuğunu varsayar ve bu iki bölgeyi birbirinden ayıran bir eşik değer bulmaya çalışır. Bunun için görüntünün histogramını esas alır ve dağıtılmış varyansın en az olduğu noktayı eşik değer olarak belirler [40]. Otsu algoritması, hızlı çalışma ve kolay kodlama gibi avantajlara sahiptir. Bununla birlikte takip edilen nesneye özel bir eşikleme yöntemi belirlemez. Şekil 1.20'te örnek bir resim üzerinde Otsu Algoritması kullanılarak eşikleme işlemi ve sonucu görülmektedir.



(a) (b)

(c) (d)

Şekil 1.20. Otsu Algoritması'na göre görüntü eşikleme (a) Orijinal görüntü (b) a'nın gri düzeyli hali (c) b'deki görüntünün histogramı (d) b'nin Otsu Algoritması'ndan elde edilen değere göre eşiklenmiş ikili hali

1.5. Optimal (İteratif) Eşikleme Yöntemi

İteratif eşikleme yöntemi da dizinde önemli bir yere sahip algoritmalardan birisidir [41]. Histograma dayalı bir eşikleme algoritmasıdır. Görüntü üzerindeki ön plan ve arka plan değerlerin ortalamalarına göre eşik değerde kaydırma yapar. Eşik değerin bir önceki durumla aynı olduğu halde işlemi sonlandırır. Aşağıda optimal (iteratif) eşik değer bulma algoritması yer almaktadır.

1. Cisimlerin konumları hakkında hiçbir bilgi olmadığını varsayalım. Sadece dört köşedeki piksellerin arka plana ait olduğu düşünülsün
2. t adımında ön planın ve arka planın ortalamalarını $\mu_B^t = \frac{\sum(i,j)\in arka plan\, f(i,j)}{\#arka plan\, pikselleri}$ ve $\mu_O^t = \frac{\sum(i,j)\in nesneler\, f(i,j)}{\#nesne\, pikselleri}$ formülleriyle hesapla.
3. Eşik değeri $T^{(t+1)} = \frac{\mu_B^t + \mu_O^t}{2}$
4. Eğer $T^{(t+1)} = T^{(t)}$ ise dur yoksa 2. Adıma geç

1.6. Biçim Bilimi (Morfoloji)

Matematiksel biçim bilim ya da diğer adıyla morfoloji 1960'lı yılların sonuna doğru geliştirilmeye başlanan ve görüntü işleme alanında önemli bir yere sahip olan bir disiplindir [40]. Ön işleme, bölütleme ve nesne şeklinin belirlenmesi gibi birçok aşamada kullanılan yöntemlerden oluşur. Tek renkli görüntüler bit sayısına göre değişen bir yelpazede renk değerlerinden oluşur. Sözgelimi 8 bitlik bir gri düzeyli görüntü 2'nin 8. kuvveti olan 256 tane değer alabilir. 1 bitlik gri düzeyli resim ise 2'nin 1. Kuvveti olan 2 tane değer alabilir. Bu iki değer de 0 ve 1 sayılarıdır. 1 beyaz rengi 0 ise siyah rengi temsil eder. İşte bu 0 ve 1'lerden oluşan görüntülere ikili görüntü denir. Gri düzey görüntüden ikili görüntüye dönüştürme işlemine *eşikleme* adı verilir. Eşikleme, eşik değer olarak belirlenen bir sayıya göre yapılır. İkili görüntüdeki piksellerden eşik değerin üzerinde değere sahip olanlar 1'e, altında değere sahip olanlar ise 0'a çevrilir. Biçim biliminde yayma ve aşındırma olmak üzere iki temel işlem vardır.

1.6.1. Aşındırma (Eroision)

Aşındırma işlemi yapısal eleman olarak adlandırılan bir çekirdeğe göre yapılır. Yapısal eleman süzme işleminde olduğu gibi görüntü üzerinde gezdirilir. Eğer merkez piksel 0 değerini taşıyor ise ana görüntü üzerinde çekirdek içindeki 0 değerlerine karşılık düşen piksel değerleri 0 olarak belirlenir. Aşındırma, görüntü üzerindeki siyah bölgelerin genişliğini artıran bir işlemdir.

1.6.2. Yayma (Dilation)

Yayma işlemi aşındırma işleminin tam tersidir. Ana görüntüdeki merkez piksel 1 değerini taşıyor ise ana görüntü üzerinde çekirdek içindeki 1 değerlerine karşılık düşen piksel değerleri 1 olarak belirlenir. Yayma görüntü üzerindeki beyaz bölgelerin genişliğini arttıran bir işlemdir.

1.6.3. Kapama (Closing)

İkili görüntü üzerinde önce aşındırma sonra da yayma işleminin uygulanmasıyla elde edilir. Kapama işleminde temel amaç gürültü olabilecek bölgeleri elemektir.

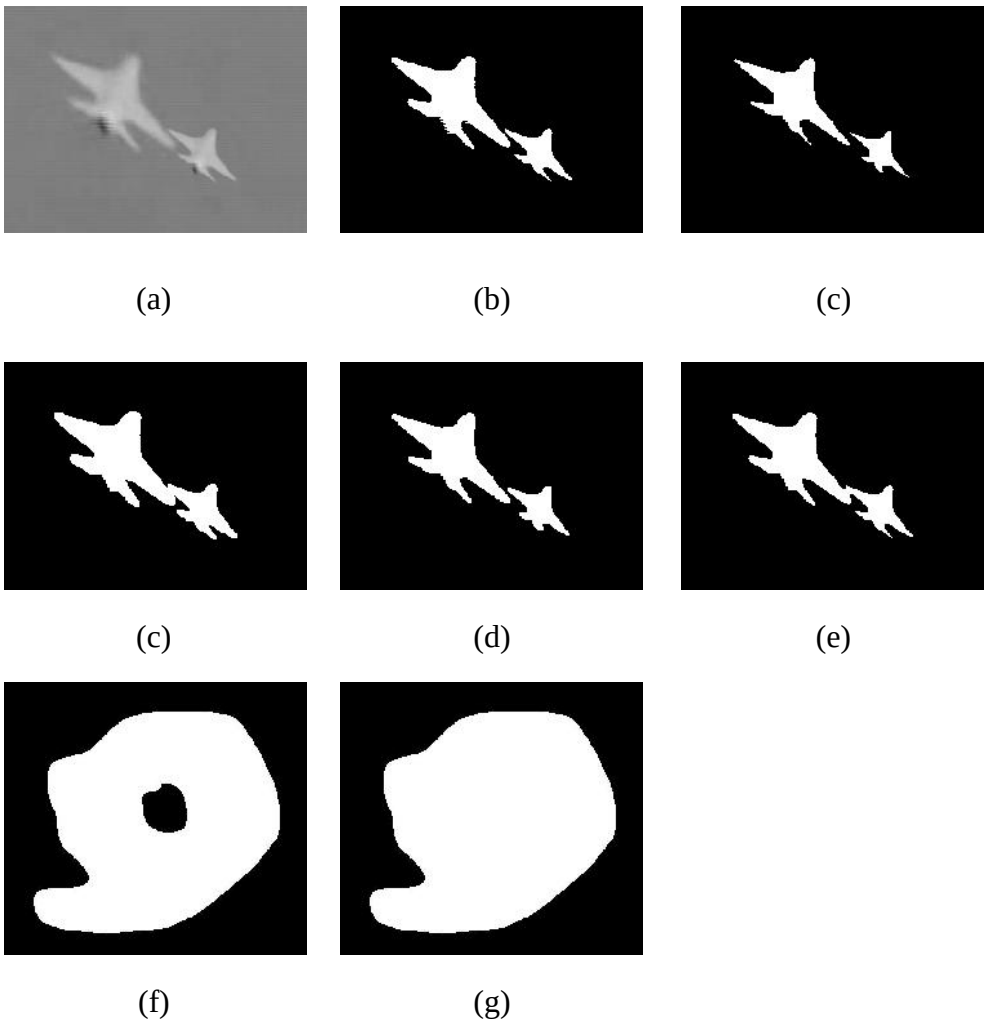
1.6.4. Açma (Opening)

İkili görüntü üzerinde önce yayma sonra da aşındırma uygulanmasıyla yapılır. Açma işleminin amacı eşikleme sırasında gereksiz yere elenmiş bilgileri geri kazanmaktır. Şekil 1.16'da biçim bilimde kullanılan algoritmaların örnek bir ikili görüntü üzerinde uygulanmış hali görülmektedir

1.6.5. Sel Doldurma Algoritması

Flood fill algoritması [42] boşluk kalan görüntülerin boşluklarını doldurmak amacıyla geliştirilmiş bir yöntemdir. Ayrıca Microsoft Paint türü görüntü işleme yazılımlarında da kendine yer bulmuş bir algoritmadır. Bu çalışmada ikili görüntüler

üzerinde doldurma yapmak için başvurulmuştur. Şekil 1.21'de biçim bilim algoritmalarının sonuçları örnek görüntüler üzerinde gösterilmektedir.



(a) (b) (c)

(c) (d) (e)

(f) (g)

Şekil 1.21 Çeşitli görüntüler üzerinde biçim bilimsel işlemler (a) Orijinal görüntü (b) a'nın ikili hali (c) b üzerinde aşındırma işlemi (d) b üzerinde yayma işlemi, (e) b üzerinde kapama işlemi, (f) b üzerinde açma işlemi, (g) örnek bir ikili görüntü, (h) g'nin doldurma ile doldurulmuş hali

1.7. Kapalı Bileşen Etiketleme

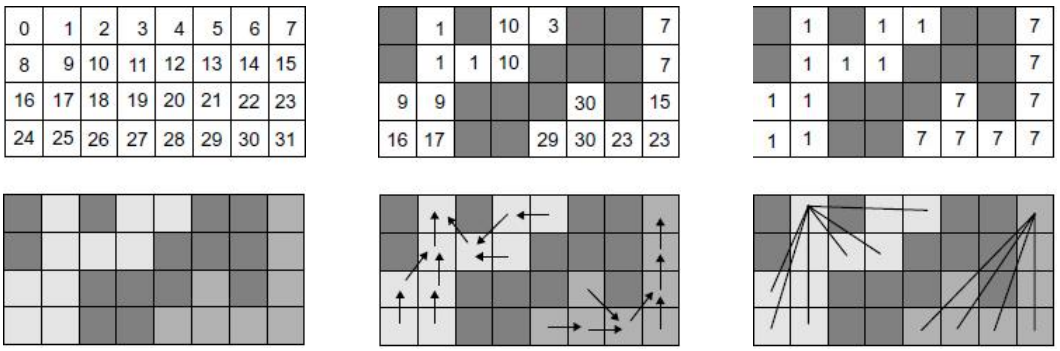
Gri düzeyli görüntünün ikili görüntüye dönüştürülmesinden sonraki aşama ikili görüntüyü oluşturan bölgeleri etiketlendirmek diğer bir deyişle bölgelerin kimliklerini belirlemektir. Her bir bölge birbiriyle komşu, bütünleşik ve aynı değeri taşıyan piksellerden oluşur. Bu bölgelerin her birine kapalı bileşen adı verilir. Kapalı bileşenlerin

etiketlenmesiyle her bir bölgeye ait alan, en, boy, ağırlık merkezi ve konum gibi değerleri elde etmek mümkün olur. Dizinde etiketleme işlemini yapmak için geliştirilmiş birçok algoritma vardır [43,44,45,46]. Bu çalışmalardan özellikle Jung- Me Park, Carl G Looney ve Hui-Chuen Chen tarafından önerilen Böl ve Yönet yöntemi koşut olarak uygulanabilmesi açısından dikkat çekicidir. Bu yönteme göre işlem birimi sayısı arttıkça işlem süresi kısalmakta fakat işlem birimi sayısı 10'a ulaştıktan sonra işlem süresinde kayda değer bir düşüş gözlenmemektedir. Bu çalışmada CUDA teknolojisi kullanıldığı için ve CUDA teknolojisinde çok sayıda işlem birimi kullanıldığı için bu yöntem yeğlenmemiştir. Bununla birlikte dizinde CUDA teknolojisi ile kapalı bileşen etiketleme hakkında yapılan çalışmalar mevcuttur. Bu çalışmalar arasında [47][48] sayılabilir. Wu, Otto ve Suziki tarafından üretilen çalışmanın paylaşımlı bellek üzerinde işlem yapmaya daha uygun olması dolayısıyla bu çalışmada kullanılmıştır. Bu algoritma bileşenlerin komşu bileşenle ilişkilendirildiği ve kümelendiği birinci faz ve kümelerin ortak köke göre etiketlendiği ikinci faz olmak üzere iki temel aşamadan meydana gelmektedir. Birinci adımda görüntü üzerinde bütün pikseller taranarak her bir piksele kendi tek boyutlu adresi verilmektedir. Bu tek boyut adres görüntü iki boyutludan bir boyutluya indirildiğinde her bir piksele karşılık gelen kendi adresidir. Söz gelimi 400 * 300 boyutunda bir resim için iki boyutlu düzenlemede i = 2 ve j = 30 adresine karşılık gelen adres tek boyutlu düzenlemede i * yükseklik + j yani 630 değerini taşımaktadır. Sonra her bir piksel komşularıyla birlikte taranır. Eğer piksel komşusuyla ilişkili ise yani aynı değeri taşıyorsa komşu pikseller arasında birleştirme işlemi uygulanır. Birleştirme işlemine ve birleştirme işleminde kullanılan kök bulma işlevine ait yalancı kod aşağıda görülmektedir.

```
kokBul(adresDizisi,elemanAdresi)
while(addresDizisi[elemanAdresi] != elemanAdresi
    elemanAdresi = addresDizisi[elemanAdresi]
return elemanAdresi

birlestir(adresDizisi, elemanAdresi0, elemanAdresi1)

 kok0 = kokBul(adresDizisi, elemanAdresi0)
kok1 = kokBul(adresDizisi, elemanAdresi1)
 if(kok0 < kok1) adresDizisi[kok1] = kok0
 if(kok1 < kok0) adresDizisi[kok0] = kok1
```

Birinci faz sonucunda her bir bileşenin bağlı olduğu herhangi bir komşusuyla bağı ortaya çıkarılmış olur. Eğer birden fazla komşusuyla ilişkili ise bu ilişkilerden sadece biri tutulur.

İkinci fazda ise bütün pikseller üzerinde kök bulma işlemi icra edilerek ortak bölgelerin kökü öbek içerisindeki en küçük adresli piksel kök seçilmek suretiyle belirlenmiş olur. Şekil 1.22'de [49] iş bu algoritmanın örnek bir görüntü parçası üzerinde icra edilmesi aşamaları görülmektedir.



(a) (b) (c)

Şekil 1.22 Örnek görüntü üzerinde kapalı bileşen etiketleme (a) İkili görüntü ve adres bilgileri (b) birinci faz sonucunda birleştirme (c) ikinci faz sonrasında ortak köke göre tamamlanmış etiketleme işlemi

Görüldüğü gibi bu algoritma koşut olarak işlemeye müsait bir algoritmadır ancak kapalı bileşenler ardışık sırada birbirini takip eden sayılardan değil de kök adres değerlerinden meydana gelmektedir.

1.8. Moment Kavramı

Moment görüntünün piksellerine ait şiddet değerlerinin ağırlıklı ortalamasıdır. Bilgisayar görmesi alanında öznitelik çıkarmak amacıyla gri düzeyli ya da ikili görüntüler üzerinde sıklıkla uygulanan istatistikî bir yöntemdir. mxn boyutunda bir I(x,y) görüntüsü için (i,j). dereceden moment (6) formülüyle hesaplanır.

$$M_{ij} = \sum_{x}^{m} \sum_{y}^{n} x^i y^j I(x,y) \qquad (6)$$

Yaygın olarak ikili görüntüler üzerinde merkez (X,Y) noktası (7) formülüyle hesaplanır.

$$X = \frac{M_{10}}{M_{00}}, \; Y = \frac{M_{01}}{M_{00}} \tag{7}$$

Moment'in yaygın bir yöntem olmasının en geçerli sebebi dönme ve öteleme işlemlerine karşı dayanıklı olmasıdır.

1.9. Hareket Kestirimi ve Kalman Süzgeci

Hareket kestirimi üzerinde çokça araştırma yapılmış konulardan biridir. Hareket kestirimi yapan başlıca süzgeçler arasında Wiener Süzgeci [50], Kalman Süzgeci [51], EKF (Genişletilmiş Kalman) Süzgeci [52] ve Parçacık Süzgeci [53] sayılabilir. Hareket kestirimi alanında kullanılan yöntemler arasında Macar matematikçi Rudolf Emil Kálmán tarafından geliştirilmiş olan Kalman Süzgeci başı çeker. Kestirim süzgeçleri Can tarafından Tablo 8 'de görüldüğü üzere sınıflandırılmıştır [54].

Tablo 8. Kestirim süzgeçlerinin sınıflandırılması

Süreç Tipi	*Kestirimci Tipi*	*Süzgeç*
Geniş anlamda durağan	Doğrusal	Wiener Süzgeci
Durağan olmayan, doğrusal, Gauss-Markov Model	Doğrusal	Kalman Süzgeci
Durağan olmayan, doğrusal olmayan Markov model	Yaklaşık Doğrusal	EKF
Durağan olmayan, doğrusal olmayan Markov model	Monte Carlo	Parçacık Süzgeci

Bu çalışmada durağan olmayan süreçleri de kapsadığı ve hızlı çalışabilen bir süzgeç olduğu için Kalman Süzgeci kullanılmıştır. Ayrıca Kalman Süzgeci bir dizi matematiksel eşitlikten oluşan bir kestiricidir ve bazı ön varsayımlar karşılığında kestirilen hata kovaryansını en aza indirdiği için en uygun süzme yöntemidir [55]. Dizinde Kalman Süzgeci ile yapılmış çeşitli çalışmalar arasında [56,57,58,59] sayılabilir.

Kalman Süzgeci (KS) doğrusal sistemler için kestirim yapabilen tekrarlı bir algoritmadır. KS'nin çalışabilmesi için başlangıç durumunun bilinmesi ve sistem gürültüsünün Gauss dağılımı şeklinde ifade edilebilmesidir.

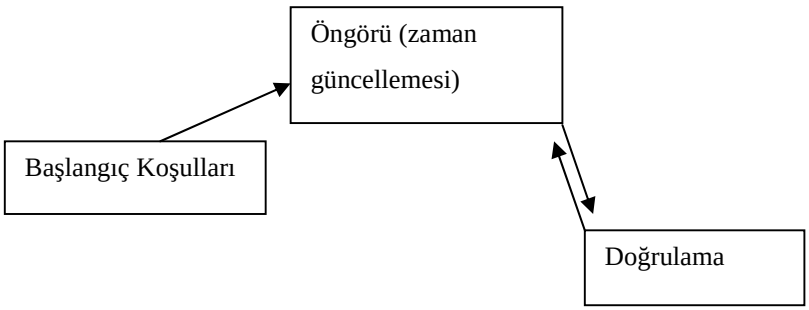
KS, kestirim ya da bir diğer deyişle tahmin yapabilmek için sistem ve ölçüm adlı iki temel modele gereksinim duymaktadır. X_k sistem durumu, X_{k-1} sistemin bir önceki durumu, A sistemin önceki durumunu şimdiki durumuna bağlayan matris ve W_{k-1} sistem gürültüsünü temsil etmek üzere sistem modeli (8) şeklinde ifade edilir.

$$X_k = AX_{k-1} + W_{k-1} \tag{8}$$

Z_k ölçüm modelinin t anındaki durumu, H matrisi de mevcut durumu ölçüme bağlama matrisi, v_k da ölçüm gürültüsü olmak üzere ölçüm modeli (9) şeklinde ifade edilir.

$$z_k = HX_k + v_k \tag{9}$$

Sistem durumu ve ölçüm modeli hesaplanan sistemde sonraki işlemler ise çevrimli olarak öngörülerin yapılması ve doğrulanmasıdır. Her adımda yapılan öngörüler doğrulama işleminden geçtikten sonra bir sonraki adımdaki öngörülere referans olmaktadır. KS'nin bu yapısı Şekil 1.23'te görülmektedir.



Şekil 1.23 Kalman Süzgeci'nin genel yapısı

KS'nde öngörü ve doğrulama için bazı temel kavramlar mevcuttur. Bu kavramlar KS başlangıç değerleri, öngörü denklemleri, doğrulama denklemleri, hata kovaryans matrisi ve kalman kazancıdır. Başlangıç değerleri ya da koşulları süzgecin çalışmasının başlangıcında sadece bir defa çalışan değerlerdir. Referans değeri oldukları ve ilk öngörü bu değerlere göre yapıldığı için önem taşırlar. Öngörü denklemleri ise iki tane denklemden meydana gelir sistemin k anındaki öngörüsü için öngörü denklemi ve öncül sistem belirsizliği (10,11)'deki gibi hesaplanır.

$$X_k^- = AX_{k-1}^+ \tag{10}$$

$$P_k^- = AP_{k-1+}^+ Q_{k-1} \tag{11}$$

Hata kovaryans matrisi (P_k) sistemdeki belirsizliği tespit eder. Sistem gürültüsünün öngörüyü bozucu etkisinin bulunmasından dolayı hata kovaryansı artar. Hata kovaryans matrisi ile birlikte Kalman kazancı (K_k) ise KS'nin en önemli bileşeni olarak kabul edilir. Kestirimi ya da diğer bir deyişle öngörüyü doğrulamak için kullanılmaktadır. Sistemin dışarıdan alınan bilgilere mi yoksa kendi öngörüsüne göre mi değerlendirme yapacağı Kalman Kazancının belirleyiciliği ile netlik kazanır. Eğer ölçüm gürültüsü artarsa öngörü kısmına, sistem gürültüsü çok ise dışarıdan alınan verilere göre kestirim yapılır. Bu parametrelere göre düzenlenmiş doğrulama denklemleri şu şekilde tanımlanır.

$$x_k^+ = x_k^- + K(z_k - Hx_k^-) \tag{12}$$

$$P_k^+ = (I - K_k H)P_k^- \tag{13}$$

$$K_k = P_k^- H^T (HP_k^- + R_k)^{-1} \tag{14}$$

Kalman Süzgeci'nin doğrusal olmayan sistemlere uyarlanmış haline Genişletilmiş Kalman Süzgeci adı verilmektedir. Bu çalışmada sistem doğrusal bir yapıda olduğu için ilerleyen bölümlerde anlatılacağı üzere Kalman Süzgeci'ne başvurulmuştur.

1.10. Bulanık Mantık

Bulanık Mantık (BM) Aristo Mantığı'nın yerine geliştirilmiş, kesin ve net sonuçlar yerine, bulanık ve tam net olmayan sonuçlar üreten bir mantıktır. BM insan düşüncesini ve mantık yürütmesini modellemeye ve karşılaşılan sorular üzerinde uygulamaya çalışır. Tıpkı insanlar gibi karşılaştığı durumlar karşısında eğer şu böyleyse, diğeri de öyleyse sonuç böyle olur gibi kurallar oluşturur ve bu kurallara göre çıkarım yapar. Çok güç bir matematiksel modele sahip sistemleri daha kolay bir şekilde modelleyebilme amacını taşır. Bu doğrultuda girilen kurallar matematiksel kurallara dönüştürülür ve sonuçlara göre çıkarım yapılır.

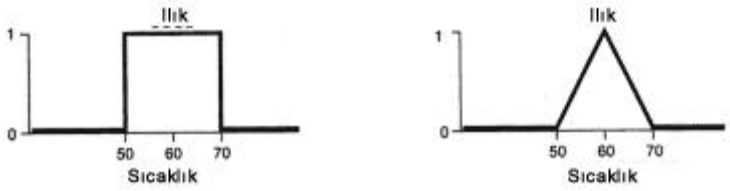
BM günümüzde özellikle kontrol mühendisliğinde kendine genişçe kullanım alanı bulmuş bir disiplindir. Günümüzde hızlı trenlerden, buzdolaplarına, çamaşır makinelerinden kameralara kadar birçok aygıt bulanık mantığa dayalı yöntemlerle kontrol edilmektedir. Bulanık mantığın tarihçesine bir bakılacak olursa; 1920 yılında Jan Lukasiewicz'in çok değerli mantık üzerine çalışması [60] bulanık mantığın başlangıcını meydana getirir. 1937 yılında Max Black'ın muğlâk küme [61] ile ilgili makalesi bulanık mantıktaki üyelik işlevini tanımlamış fakat bilim dünyasınca pek dikkate değer bulunmamıştır. 1965 yılına gelindiğinde ise Lütfi Askerzâde'nin bulanık küme kuramı [62] bugünkü BM kavramını tam anlamıyla ortaya koıymuştur. Bu bakımdan Askerzâde, Bulanık Mantık'ın babası sayılabilir. 1974 yılıunda Mamdani'nin buhar makinesini bulanık denetimle gerçekleştirmesi işbu disiplinin uygulanabilirliğini kanıtlaması bakımından ayrı bir önem taşır. 1976 yılında Danimarka'da Circle Coment ve SIRA adlı firmalar çimento fırınlarının denetiminde bulanık uygulamalar gerçekleştirmişlerdir. 1987 yılında Hitachi tarafından tasarlanan Japon Sendai Metrosu bulanık mantık denetleyicisiyle çalışmaya başlamıştır. Doksanlı yıllardan itibaren de BM iyice yaygınlaşmış ve birçok mühendislik uygulamasının temelini oluşturmuştur.

1.10.1. Bulanık Mantığın Temel Bileşenleri

Bulanık mantık dört temel bileşenden meydana gelmektedir. Bunlar bulanık kümeler, dilsel değişkenler / bulanık değerler, üyelik işlevleri ve bulanık kurallardır.

1.10.1.1. Bulanık Kümeler

Klasik kümede bir eleman ya kümenin içindedir ya da dışındadır. 1 üye olmayı 0 da üye olmamayı temsil eder. Bulanık kümede ise bir eleman [0,1] aralığında değişen değerler alabilir. Onun için 1 tam olarak üye olmayı, (0,1) üye olma derecelerini 0 ise hiç üye olmamayı temsil eder. Şekil1.24'te bulanık ve klasik kümeler karşılaştırmalı olarak sergilenmektedir.



(a) (b)

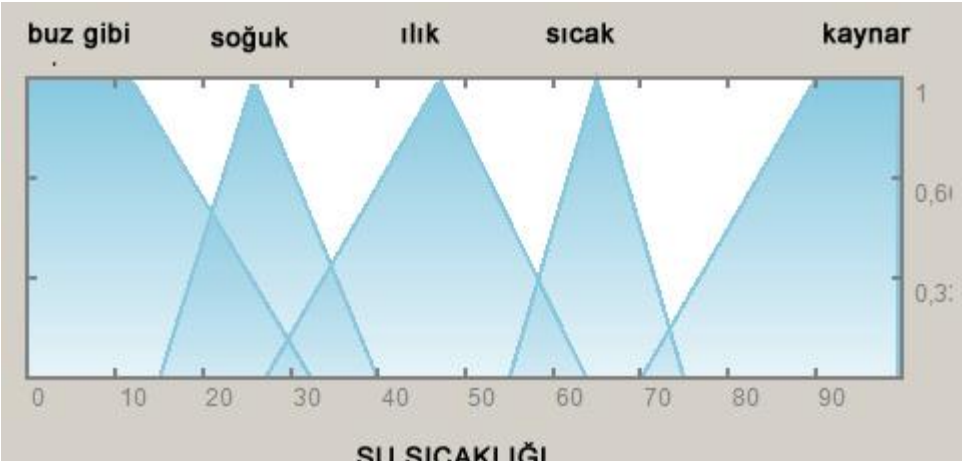
Şekil 1.24. Bulanık ve klasik küme karşılaştırması (a) klasik küme (b) bulanık küme

1.10.1.2. Dilsel Değişkenler

Bulanık mantık güç matematiksel modellere sahip sistemleri daha kolay şekilde modellemeyi hedefler. Bu amaçla klasik matematiksel formülleri insanların daha kolay anlayabileceği şekilde sözcüklerle ifade eder. Söz gelimi sıcaklığı tanımlamak için sıcak, soğuk, ılık, sıcak, kaynar gibi sözcüklerle insanların tanımlamasına uygun deyimlerle niteler.

1.10.1.3. Üyelik Dereceleri ve Üyelik İşlevleri

Üyelik derecesi dilsel değişkenin bulanık kümeye hangi oranda ait olduğunu belirtir. Bir dilsel ifadenin bütün değerler için derecelerinin matematiksel karşılığı da üyelik işlevidir. Dizince sıkça kullanılan üyelik işlevleri arasında üçgen, yamuk, gauss, sigmoid üyelik işlevleri sayılabilir. Şekil 1.25'te örnek sıcaklık kümesine ait üyelik işlevleri görülmektedir.



Şekil 1.25 Su Sıcaklığı kümesine ait üyelik işlevleri

1.10.1.4. Bulanık Kurallar

Bulanık mantığın temel bileşenlerinden biri de bulanık kurallardır. Bulanık kurallar insan dilindeki eğer, ise, ve, veya gibi ifadelerle oluşturulur. Değişik koşulları insan mantığına uygun biçimde "Eğer bugün hava yağışlı ve soğuk ise şemsiyemi ve kazağımı yanıma almalıyım", "Eğer bugün hava yağışlı ve ılık ise şemsiyemi almalıyım" örneklerinde görüldüğü gibi modeller.

1.10.2. Bulanık Sonuç Çıkarım Mekanizmaları

Bulanık Mantık disiplini içerisinde yukarıda anlatılan temel bileşenleri farklı biçimde kullanarak sonuç üreten mekanizmalar vardır. Bu mekanizmalar sonuç çıkarma mekanizmaları olarak adlandırılmaktadır. Bunlara örnek olarak Sugeno [63], Tsukomaoto [64] ve Mamdani [65] yöntemleri verilebilir.

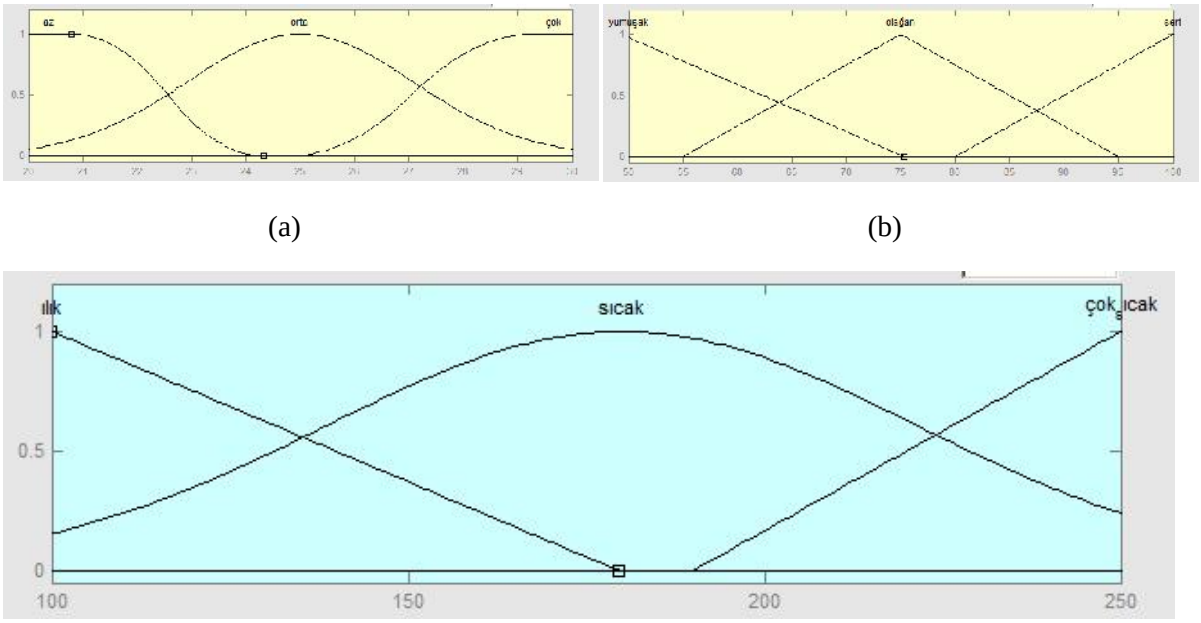
Bulanık sonuç çıkarma mekanizmaları beş temel adımdan meydana gelir. Bunlar girişlerin bulanıklaştırılması, bulanık işlecin uygulanması, bulanıklaştırma, çıktıları toplama ve durulaştırma işlemleridir. Bu mekanizmanın genel yapısı Şekil 1.26'da görülmektedir.

Şekil 1.26 Bulanık Sonuç Çıkarma Mekanizmaların Genel Yapısı

Girişlerin bulanıklaştırılması işleminde dış dünyaya ait veriler bulanık kümeler haline getirilir. Bulanık işlecin uygulanması süreci de bulanık kurallar oluşturulmaya başlanır. Bunun için VE, VEYA, DEĞİL mantıksal ifadelerinin kullanımına karar verilir. 'VE' ifadesi için en küçük ve prod, 'veya' ifadesi için en büyük ve propor işleçleri yer almaktadır. Sonraki işlem olan bulanıklaştırma işleminde her bir kural için girişlere önceki iki adımın uygulanmasıyla oluşan tavan değere göre çıkış işlevi biçimlendirilir. Böylece her bir kural için çıktılar oluşturulmuş olur. Diğer bir adım olan toplama aşamasında her bir kural için elde edilen yöntemler en büyük, genel toplam ya da propor yöntemlerinden birisiyle toplanır ve bütün kuralların katılımıyla çıkışlar elde edilmiş olur. Son adım olan bulanıklaştırma işlemi ise elde edilen çıkışların harmanlanarak tek çıkış için tek sonuç değerinin elde edilmesidir.

1.10.2.1. Mamdani Bulanık Sonuç Çıkarımı

Mamdani Yöntemi en sık kullanılan bulanık sonuç çıkarım mekanizmalarından birisidir. Bulanık küme kuramını ilk kullanan uygulama olması açısından da dikkat çekici bir yöntemdir. Mamdani Yöntemi'ni şu örnekle açıklayalım. Bir otomatik fırının sıcaklığı yiyecek miktarı ve yiyecek sertliğine göre kontrol edilsin. Buna göre bulanık mantık çıkarım yöntemlerinin birinci aşaması olarak (girişlerin bulanıklaştırılması) giriş ve çıkış kümeleri Şekil 1.27'de görüldüğü gibi oluşturulmuş olsun.

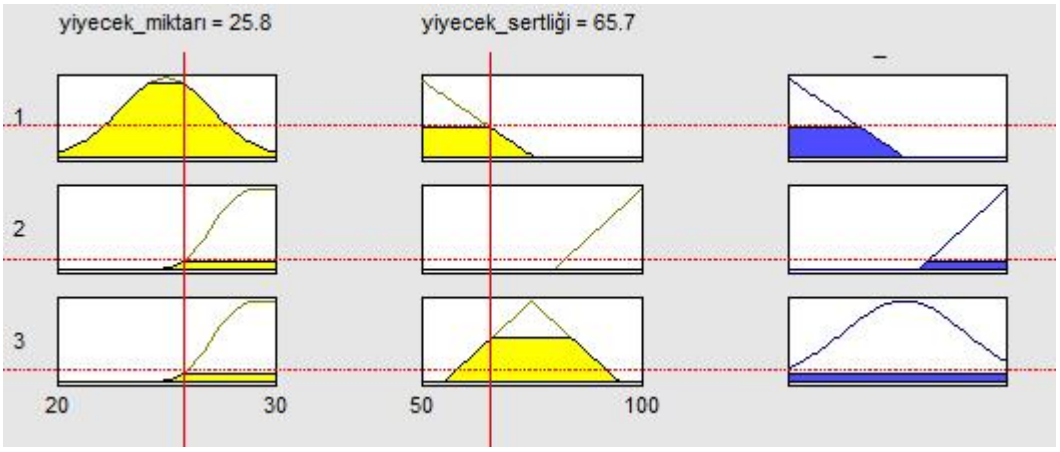


(a) (b)

(c)

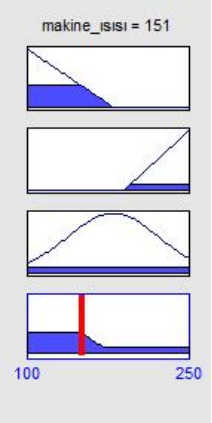
Şekil 1.27 Fırın Kontrolüne ait bulanık giriş ve çıkışlar (a) yiyecek miktarı adlı giriş (b) yiyecek sertliği adlı giriş (c) fırın sıcaklığı adlı çıkış.

Sonrada giriş ve çıkışlar arasındaki bağı kuran kurallar oluşturulsun. Bu kurallar da ‘Eğer yiyecek miktarı orta ve yiyecek sertliği yumuşak ise makine ısısı ılıktır’, ‘Eğer yiyecek miktarı çok veya yiyecek sertliği sert ise makine ısısı çok sıcaktır.’, ‘Eğer yiyecek miktarı çok ve yiyecek sertliği olağan ise makine ısısı sıcaktır.’ Önermelerinden oluşsun. Yiyecek miktarı 27.8 kg ve yiyecek sertliği 65.7 olan ani durum için ikinci adım olan bulanık işleçlerin uygulanması şu Şekil 1.28’de görüldüğü üzere gerçekleştirilir. Herhangi bir kural için kuralda yer alan girişler taranır. Girişin kuralda yer alan üyelik işlevinden elde edilen sonuçlar kuralda yer alan mantık ifadesine göre değerlendirilir ve çıkışın üyelik derecesi belirlenir. Örnekteki birinci kuralda mantık ifadesi ve olduğu için işlevlerden elde edilen üyelik derecelerinin en küçüğü sonuç çıkışın üyelik derecesi olarak belirlenmiştir. İkinci kuralda mantık ifadesi veya olduğu için giriş üyelik derecelerini en büyüğü çıkışın üyelik derecesi olarak ortaya çıkmaktadır.



Şekil 1.28 Bulanık işleçlerin uygulanması

Bir sonraki aşama ise çıkışların toplanmasıdır. Her kuraldaki çıktı değerleri toplanır. Bu toplama işlemine göre elde edilen sonuçlar son aşama olan durulaştırma işlemine tabi tutulur. Durulaştırmadan sonra bulanıklıktan kurtularak sistemin çıktısı elde edilmiş olur. Örnek sistem için durulaştırma sonucu Şekil 1.29'da görülmektedir.



Şekil 1.29 Durulaştırma işleminden sonra elde edilen sonuç

1.10.2.2. Sugeno Bulanık Sonuç Çıkarımı

Sugeno ya da diğer bir adıyla Takagi-Sugeno-Kang bulanık sonuç çıkartım mekanizması da Mamdani mekanizması gibi çok yaygın kullanılan bir yöntemdir. Mamdani'den temel farkı çıkışların bir sabit sayı ya da doğru olmasıdır.

1.11. Gri Düzey Oluşum Matrisi

Gri düzey oluşum matrisi (GDOM) örüntü tanıma alanında sıklıkla başvurulan önemli bir yöntemdir. GDOM ilk defa Haralick [66] tarafından ortaya atılmış bir yöntemdir. GDOM ikinci dereceden bileşik durum olasılık yoğunluk fonksiyonunun, P(i, j | d, θ), tahminine dayanır. Bu matris pikseller arasındaki uzaklık d ve açı θ iken gri seviyesi i'den gri düzeyi j'ye geçme olasılığını gösterir. Kare matris kullanılır ve desen özelliklerinin dönme ile değişmediğini garantilemek için genellikle θ = 0º, 45º, 90º ve 135º olacak şekilde dört yönde inceleme yapılır. [67] Fakat bu çalışmada gerek zarlı gerekse de zarsız fındık görüntülerinin karmaşık bir yapıya sahip olmamasından ve daha hızlı sonuç üretebilmek için θ = 0º için GDOM yeterli görülmüştür. Tespit edilen fındık görüntüleri 8 bit gri düzeyli görüntüden 3 bit gri düzeyli görüntüye dönüştürülerek GDOM bilgileri elde edilmiştir. Şekil 1.30'da örnek bir 3 bitlik görüntü ve ona ait GDOM görülmektedir.

3467	563	1	0	0	0	0	0
515	7602	836	10	0	0	0	0
0	686	6809	555	44	3	0	0
0	24	520	3457	349	35	10	0
0	8	18	375	4684	219	24	0
0	0	13	6	278	1765	218	0
0	0	1	14	10	227	2511	9
0	0	0	0	0	0	9	35

(a) (b)

Şekil 1.30 Örnek görüntü ve gri düzeyli oluşum matrisi (a) Resim (b) GDOM

1.12. Görüntüye Ait İstatistikî Öznitelikler

Dizinde görüntünün istatistikî özelliklerini belirlemek amacıyla pek çok formül kullanılmaktadır. Bu çalışmada fındık türünü belirleyebilmek amacıyla düzensizlik (entropi), enerji, karşıtlık (kontrast) , ilinti (korelasyon) ve tek biçimlilik (homojenlik) parametreleri kullanılmıştır. Tablo 9'de bu parametrelerin açıklamaları yer almaktadır.

Tablo 9. Görüntüye ait istatistikî öznitelikler

Öznitelik Adı	Açıklama	Formül
Düzensizlik (Entropi)	Düzensizlik görüntüye ait istatistikî değerler arasında sık başvurulan bir yöntemdir. Görüntüdeki piksellerin birbiriyle yakınlığının bir ölçüsüdür.	$-\sum \frac{c}{p} \times \frac{nül}{log2(p)}$
Enerji	Enerji gri düzey oluşum matrisinden elde edilen olasılık yoğunluk işlevindeki değerlerin karelerinin toplamıdır. 0 -1 arası değerler alır.	$\sum_{i,j} \frac{(}{p_{i,j}}$
Karşıtlık	Görüntüdeki pikseller arasındaki düzey farkını ölçmede kullanılan kullanışlı bir yöntemdir	$\sum_{i,j} \frac{(}{p_{i,j}}$ - j)
İlinti	[-1 1] Aralığında değerler alır. Bir pikselin bütün görüntü boyunca komşuluğundaki piksellerle nasıl bir ilişki içerisinde olduğunun ölçüsüdür.	$\sum_{i,j} \frac{(}{}$
Tek biçimlilik	Görüntüdeki piksellerin birbirlerine benzerliğini ölçmek amacıyla kullanılır.	$\sum_{i,j} \frac{(}{1 + \frac{p_{i,j}}{\lvert i-j \rvert}}$

2. YAPILAN ÇALIŞMALAR, BULGULAR VE İRDELEME

Bu çalışmada yapılan işlemleri hareketli bölgelerin belirlenip izlenmesi, hareketli bölgelerin istatistikî özniteliklerin çıkartılması ve bulanık mantığa dayalı karar vermeden oluşan üç ana başlık altında incelemek olasıdır. Sistemin bu yapısı Şekil 2.1'de gösterilmektedir. Sistemin birinci adımda ortaya çıkan ikili görüntüler hareketli bölgeleri belirlemek için kullanılmış daha sonra bu ikili görüntülerin sahneye ait gri düzeyli görüntü üzerindeki izdüşümleri üzerinde istatistikî özellik çıkarımı ve bulanık karar verme adımları fındıkları tanımlamak amacıyla tatbik edilmiştir.

Şekil 2.1 Sistemin Genel Yapısı.

2.1. Hareketli Bölgelerin Belirlenmesi ve İzlenmesi

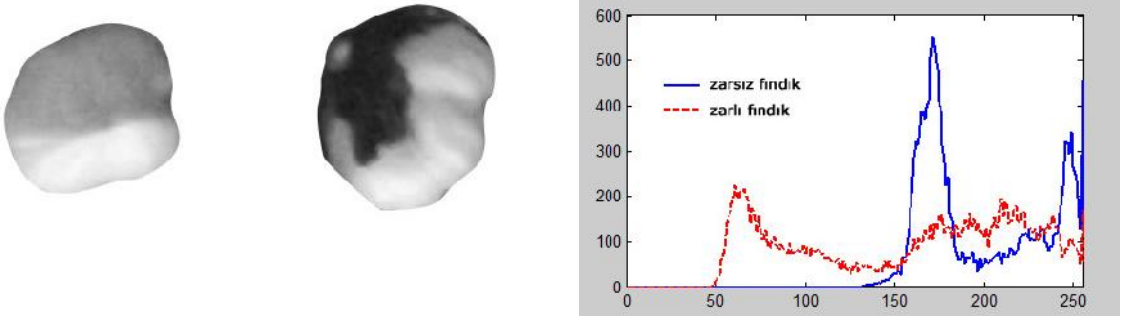
Sistemin ilk adımı bant üzerindeki fındıkların belirlenmesidir. İstatistikî öznitelik çıkarma ve bulanık sınıflandırma aşamalarında bu adımda belirlenen bölgeler kullanılacağı için çok önemli bir adımdır. Üç alt basamaktan meydana gelmektedir. Bu basamaklar ilk bir saniyedeki sahneler kullanarak arka planın modellenmesi, herhangi bir sahne için hareketli bölgelerin tespit edilmesi ve tüm video için bölgelerin izlenmesi ve konumlandırılmasıdır.

2.1.1. Arka Plan Modelleme

Geliştirilen sistemin başat niteliği bir sanayi uygulaması olmasıdır. Dolayısıyla sistem kurulurken gürültülü ve değişken yapıdaki doğal bir ortamda hareket tespiti yapan bir sistemden ziyade hızlı bir şekilde sonuca ulaşabilecek ve hareketin kolay belirlenebileceği bir düzenek geliştirilmiştir. Bir sonraki başlık olan hareketli bölge belirleme başlığında ayrıntılı olarak tartışılacağı üzere kırmızı renkli bir bant kullanılmıştır. Bandın tek renkli olmasına karşılık sürekli hareket etmesi dolayısıyla belirli bir aralıkta modellemesine gereksinim duyulmuştur. Bu amaçla bant çalıştıktan bir dakika sonra fındıklar bant üzerine yerleştirilmiş. İlk 25 sahne arka plan modellemesinde kullanılmıştır. Arka plan modellemek amacıyla en çok tekrarlılık [68] yöntemine başvurulmuştur. Bu yöntem gereğince bütün arka plan görüntüleri taranmış her hangi bir piksel için arka görüntülerinde o pikselde en çok tekrar eden değer arka plan değeri olarak belirlenmiştir. Böylece çok sayıda arka plan görüntüsü en sık değerleri tarafından temsil edilen tek bir arka plan görüntüsüne dönüştürülmüş olmaktadır.

2.1.2. Hareketli Bölge Belirleme

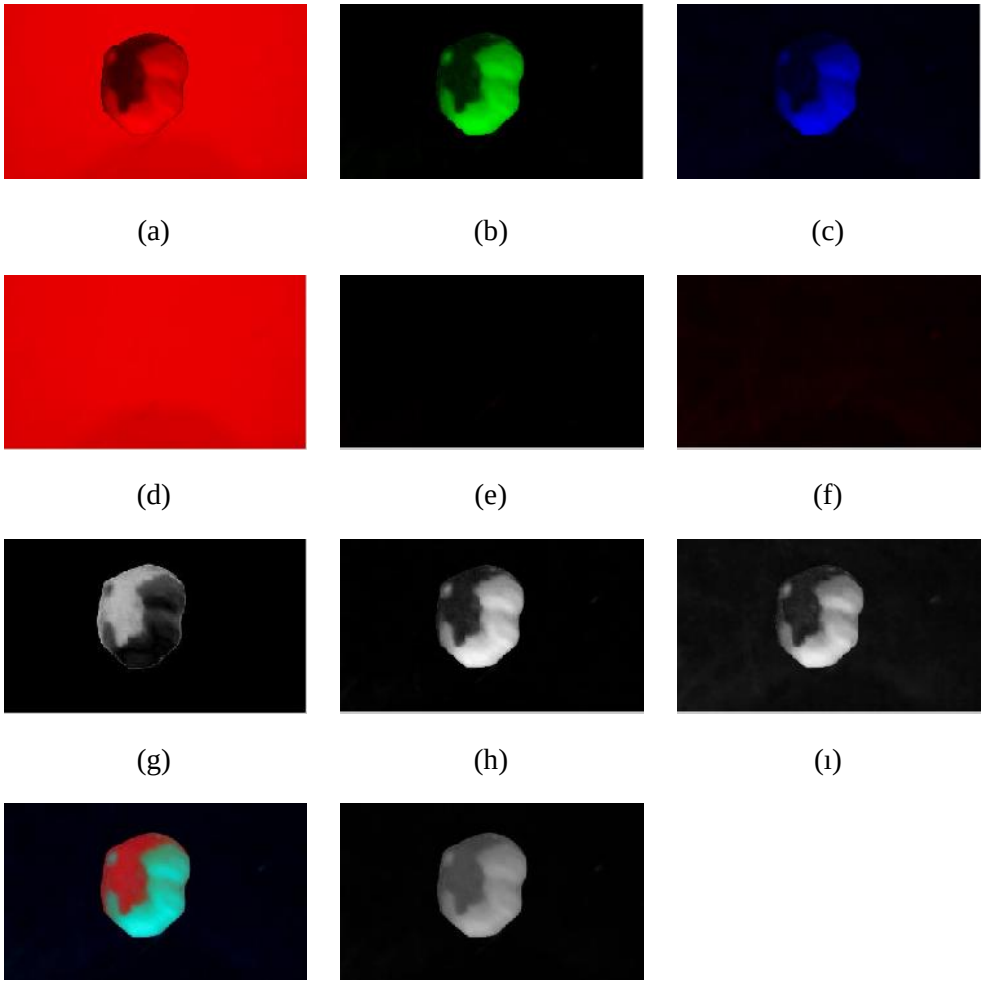
Hareketli bölgelerin tespiti temel olarak modellenen arka plan ile mevcut sahnenin farkının alınmasına dayalıdır. Bu aşamada elde edilen fark üzerinde biçimsel düzenlemelere gidilmiş ve mevcut sahne içerisinde yer alan bölgeler tespit edilmiştir. Fark alma işlemini hem zarlı hem de zarsız fındıklarda başarıyla almak çok önemlidir. Bilhassa zarlı fındıklarda zarlı bölge ile zarsız bölgenin tek bir fındık olarak etiketlenmesi hareketli bölge tespitinde aşılması gereken ciddi bir engeldir. Bu yüzden zarlı ve zarsız fındıklar ayrıntılı olarak incelenmiştir. Zarsız fındık görüntüleri incelendiğinde fındıkların beyaza yakın sarı renk taşımalarından dolay kırmızı, yeşil ve mavi kanala ait histogram değerlerinin yüksek sayılarda kümelendiği görülmüştür. Zarlı fındık görüntülerinin belirli bir oranda zarsız fındıklara benzemekle birlikte kahverengi zara sahip olmalarından dolayı alçak histogram değerleri de taşıdığı görülmektedir. Şekil 2.2’de bir adet zarlı ve bir adet zarsız fındığa ait gri düzeyli görüntü histogramları görülmektedir. .



(a) (b) ©

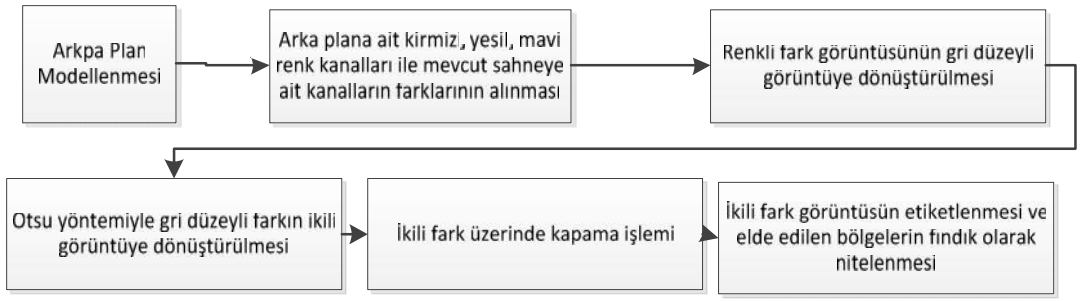
Şekil 2.2 Fındık örnekleri ve histogram değerleri (a) zarsız fındık (b) zarlı fındık (c) a ve b'deki fındık örneklerine ait gri düzey histogram değerleri

Zarlı ve zarsız fındıkların da arka plandan kolayca ayırt edilebilmeleri amacıyla kırmızı renge sahip bir bant kullanılmıştır. Kırmızı renkli bir bandın kırmızı kanalına ait histogram değerleri çok yüksek çıkmakta, yeşil ve mavi kanallarına ait histogram değerleri aşağı düzeyde seyretmektedir. Yarı zarlı bir fındık için kırmızı, yeşil ve mavi kanallara ait histogramlarda hem yüksek hem de alçak değerlere rastlanmaktadır. Dolayısıyla sadece kırmızı kanalı için düşünüldüğünde bazen fındık bazen de arka plan daha yüksek değer taşımaktadır. Diğer kanallarda ise kırmızı kanalın tam tersi bir sırada histogram farkı bulunmaktadır. Bu durumlar değerlendirilmiş ve arka plan ile sahnenin farkları kırmızı, yeşil ve mavi alınarak farkların toplamı gri düzeyli fark olarak belirlenmiştir. Şekil 2.3'de örnek bir sahne ve arka plan için değişik kanalların farkları ve nihai gri düzeyli fark görülmektedir.



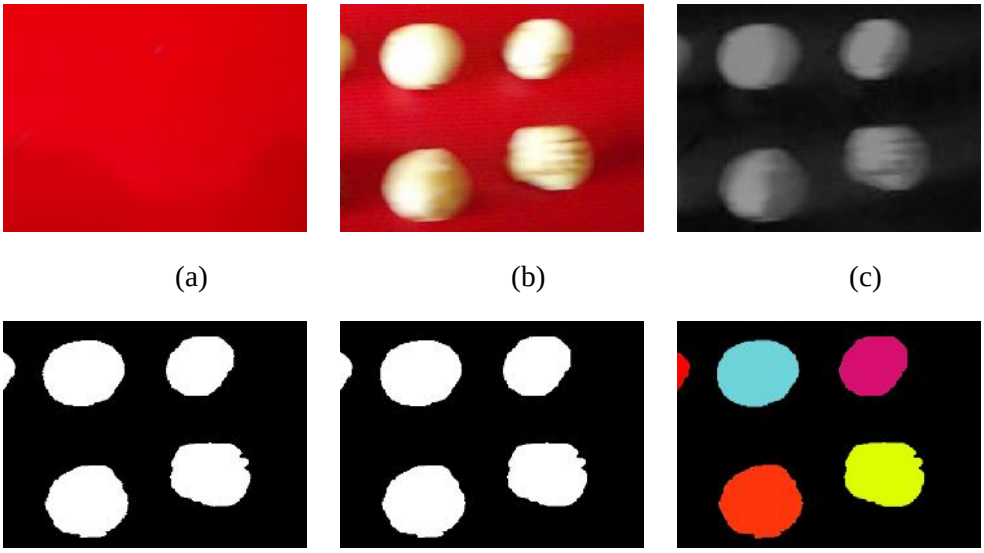
Şekil 2.3. Örnek bir sahne ve arka plan görüntüsü için değişik kanallara ait farklar ve nihai gri düzeyli fark (a) sahneye ait kırmızı kanal (b) sahneye ait yeşil kanal (c) sahneye ait mavi kanal (d) arka plana ait kırmızı kanal (e) arka plana ait yeşil kanal, (f) arka plana ait mavi kanal (g) a ile d'nin fark, (h) b ile h'nin farkı (ı) c ile f'nin farkı, (j) kırmızı, yeşil ve mavi farkların birleşmesiyle elde edilen renkli görüntü. (k) renkli farkın gri düzeye dönüştürülmesiyle elde edilen nihai fark.

Gri düzeyli fark elde edildikten sonra Otsu yöntemiyle ikili görüntüye dönüştürülmektedir. Eşikleme işlemi için Otsu yönteminin yeğlenmesinin sebebi Otsu yönteminin en küçük değişintiye göre eşik değer belirlemesidir. Dolayısıyla nihai fark görüntüsü fındığın bölgesini doğru bir şekilde içeren ikili görüntüye dönüştürülmektedir. İkili görüntü tam net olarak düzeltilmesi amacıyla son olarak biçim bilimsel kapama işleminden geçirilmektedir. Hareketli bölgelerin belirlenmesi genel itibariyle Şekil 2.4'te görülmektedir.



Şekil 2.4. Hareketli bölgelerinin belirlenmesi işleminin temel adımları.

Şekil2.5'de ise örnek bir sahne ve arka plan için hareketli bölge belirleme işlemi adım adım gösterilmiştir.



Şekil 2.5. Algoritmanın örnek görüntü üzerinde tatbik edilmesi. (a) modellenen renkli arka plan görüntüsü, (b) mevcut sahne görüntüsü (c) arka plan ve sahnenin kırmızı, yeşil, mavi kanallar için farkının alınması ve gri düzeyli görüntüye çevrilmesi (d) gri düzeyli görüntünün Otsu algoritması ile ikili görüntüye çevrilmesi (e) ikili görüntü üzerinde kapama işleminin uygulanması. (f) elde edilen ikili görüntünün etiketlenmesi ve fındıkların işaretlenmesi. (her bir bölge ayrı bir renk ile gösterilmiştir)

2.1.3. Bölgelerin Takibi

Kaynak video üzerinde tespit edilen her hareketli bölge bant üzerinde fındıktan başka bir cisim olmadığı için bir fındığa ait kabul edilmiştir. Hareketli bölgeler diğer bir deyişle fındıklar tespit edildikten sonraki önemli aşama fındıkları video süresince takip ederek zarlı fındıkların konumunu tespit etmektir.

Bu amaçla video sahnesinin herhangi bir anında tespit edilen hareketlilere bölge, hareketi sürekli olarak takip edilerek işaretlenmiş bölgelere de fındık adı verilmiştir. Her bir fındığın daha önce gittiği yolun kaydedilmesi, Kalman süzgeci ile yörünge kestirimini olası kılmıştır.

Sistemin en önemli ayağını herhangi bir bölgenin fındık ile eşlenmesidir. Bunu için bölgenin merkez noktası, (x_0, y_0) fındığın kayıtlı son merkez noktası (x_1, y_1) ve fındığın bir sonraki adımda bulunması beklenen nokta (Kalman süzgeci yardımıyla kestirilen) (x_2, y_2) kullanılarak meydana getirilmiş uzaklık formülüne (17) başvurulmaktadır. Her bir fındık kendine en yakın bölge ile eşleştirilmektedir. Bölgenin merkezi fındığın yeni merkezi olarak kabul edilmekte, yeni merkez fındığın takip ettiği çığır bilgisi olarak kaydedilmekte ve Kalman Süzgeci yardımıyla yeni yörünge kestirilmektedir.

$$x_{uzunluk} = x_0 - \sqrt{x_1 . x_2} \tag{15}$$

$$y_{uzunluk} = y_0 - \sqrt{y_1 . y_2} \tag{16}$$

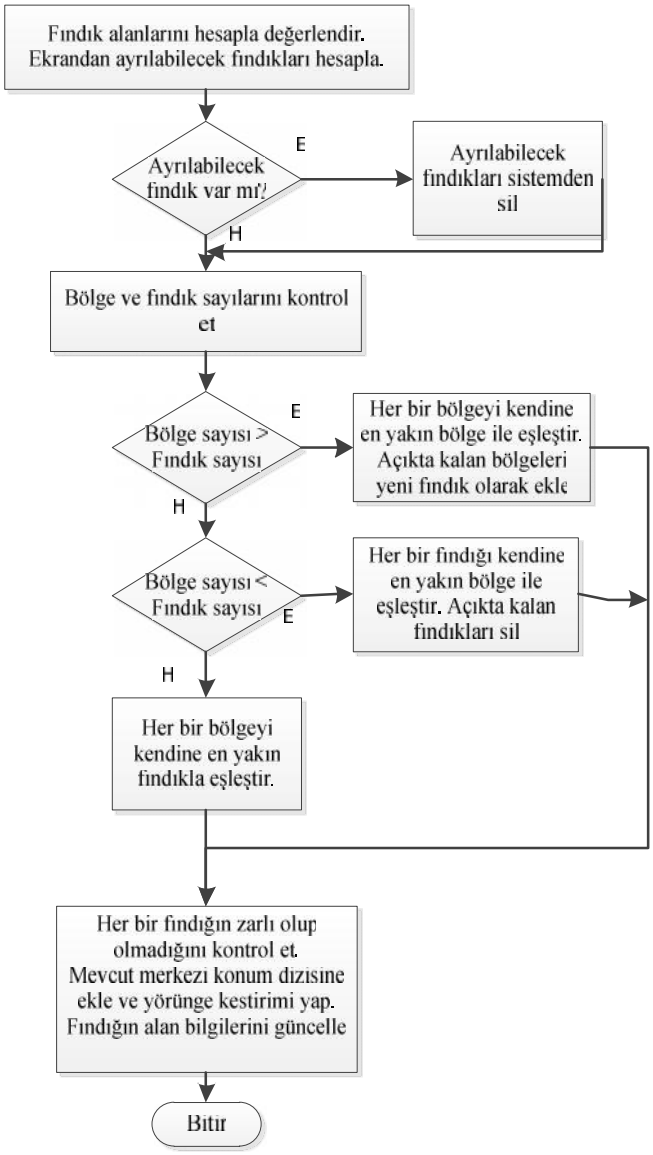
$$mesafe = \sqrt{{x_{uzunluk}}^2 + {y_{uzunluk}}^2} \tag{17}$$

Ekrandaki (gözlem alanında) fındık ve bölge sayısına göre üç farklı durum meydana gelebilmektedir. Bu durumlar bölge sayısının fındık sayısından fazla olması, fındık sayısının bölge sayısına eşit olması ve fındık sayısının bölge sayısından fazla olması durumlarıdır.

Bölge sayısının fındık sayısından fazla olması halinde ekrana yeni fındıkların girdiği varsayılmaktadır. Her bir fındık (17)'de ifade edilen uzaklığa göre kendine en yakın bölgeyle eşlenmekte ve açıkta kalan bölgeler yeni fındıklar olarak numaralandırılmakta ve kaydedilmektedir. Böylece fındık sayısı ile bölge sayısı eşit sayıya ulaşmaktadır.

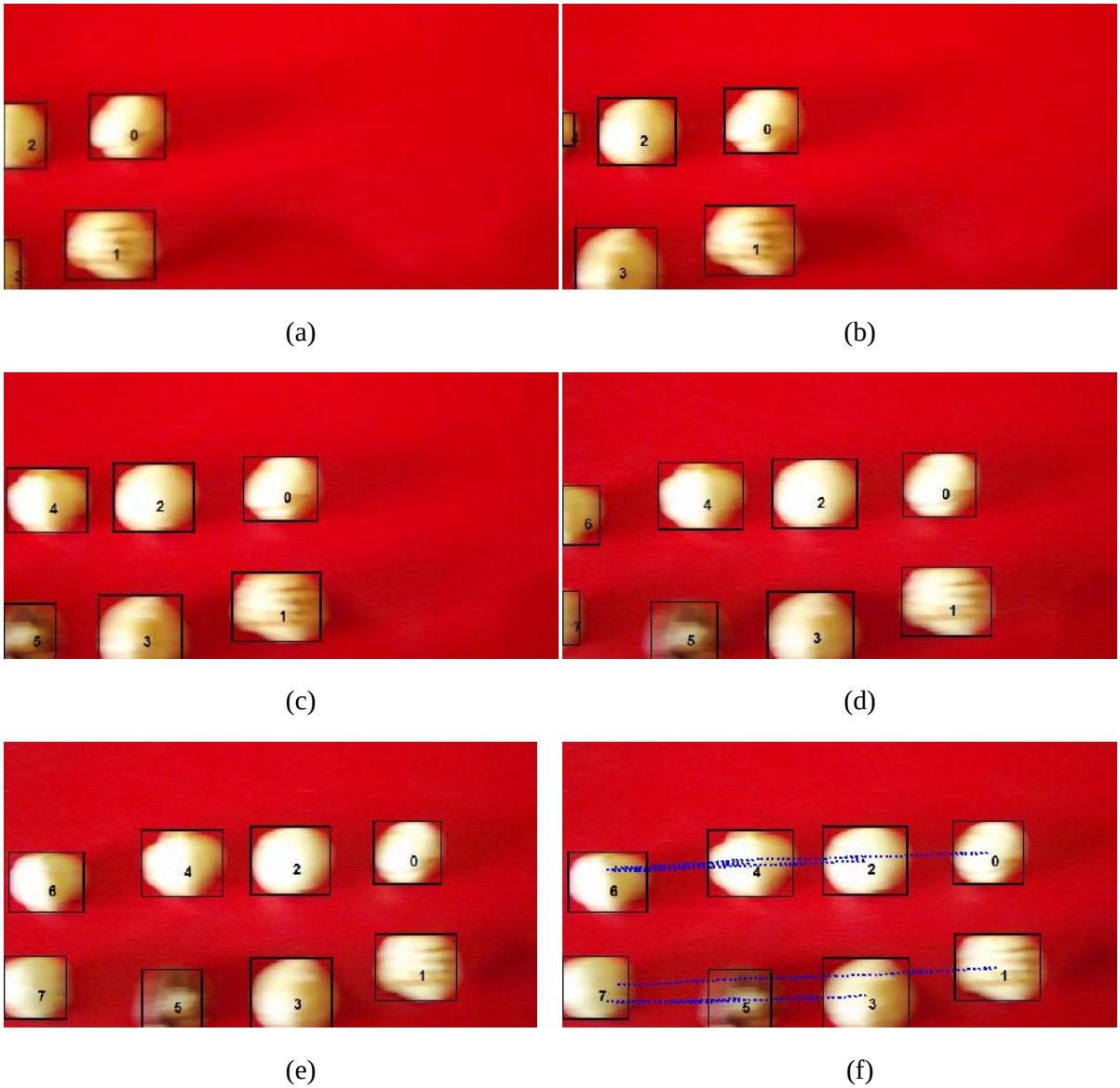
Bölge sayısının fındık sayısına eşit olması halinde uygulanan yöntem her bir bölgenin kendine en yakın fındık ile eşlenmesidir. Ancak bu durumda yanlış şekilde takip yapılmasına yol açabilecek bir sakınca söz konusudur ki bu da bazı fındıkların ekrandan çıkmasına karşın çıkan fındık miktarınca yeni fındığın aynı anda ekrana girmesidir. O zaman takipten çıkarılması (eğer zarlı ise kullanıcıya haber verilmesi) gereken fındıklar çıkarılmamış ve bazı fındıklar yanlış bölgelerle eşlenmiş olabilir. Bu istenmeyen durumun önüne geçebilmek amacıyla her bir fındığın video boyunca kapladığı alan hesap edilmiştir. Bir fındığın alanında küçülme varsa bu o fındığın bir kısmının gözlem alanından çıktığı şekilde yorumlanmıştır. Bu fındığın birkaç sahne içerisinde gözlem alanından tamamıyla ayrılması beklenmektedir. Sistem düzgün doğrusal harekete tabi olduğu için belirli bir kısmı gözlem alanında ayrılan bir fındığın tam olarak ekrandan çıkacağı zaman hesaplanabilir. Bu sayede herhangi bir anda sistemden çıkacağı hesaplanan fındıklar kayıttan düşülerek belirtilen sakınca ortadan kaldırılmaktadır.

Son durum ise kayıtlı fındık sayısının bölge sayısından fazla olması durumudur. Bu durum bazı fındıkların gözlem alanından çıkmış olmasıyla açıklanabilir. Bu halde her bir bölge kendine en yakın bölge ile eşleştirilmekte ve açıkta kalan fındıklar kayıttan düşülmektedir. Gözlem alanından ayrılan fındıkta zar tespit edildi ise kullanıcıya fındığın zarlı olduğu bilgisi verilmektedir. Hareket izleme sisteminin akış diyagramı Şekil 2.6’da gösterilmiştir.



Şekil 2.6 Hareket Takibi algoritmasının temel yapısı

Şekil 2.7 ‘de ise aynı videoya ait değişik sahnelerde gözlenen fındık seyirleri her bir fındık ayrı bir numarayla işaretlenmiş olarak gösterilmektedir.



Şekil 2.7. Kaynak videonun değişik zamanlardaki sahnelerinde elde edilen bölgelerin fındık olarak işaretlenmesi(a-e) (f) fındıkların izledikleri çığırın gösterimi

2.2. Hareketli Bölgelerin İstatistikî Özniteliklerin Belirlenmesi

Geliştirilen sistemin bir sonraki ana adımı belirlenen hareketli bölgelerin istatistikî özniteliklerini belirlemektir. Bir önceki aşamada hareketli olarak tespit edilen bölgelerin gri düzeyli sahne üzerinde iz düşümleri üzerinde istatistiki öznitelik çıkarma işlemi eş zamanlı olarak uygulanmıştır. Bu istatistikî özellikler düzensizlik, enerji, karşıtlık, ilinti, tek biçimlilik ve gri düzeyli fındık görüntüsünün optimal eşikleme algoritmasıyla eşiklenmesi sonucu oluşan siyah nokta sayısının bütün piksel sayısına oranı olan siyah bölge oranıdır.

2.3. Bulanık Mantık ile Bölgelerin Sınıflandırılması

Geliştirilen sistemde son işlem adımı ise daha önceden tespit edilen bölgelerin, her bir bölge bant üzerinde başka bir cisim olmadığı için fındık olarak adlandırıldı, bulanık mantık yöntemleriyle zarlı ya da zarsız olduğuna karar verilmesidir. Bu amaçla fındık görüntülerinin istatistiki özniteliklerine dayalı kurallar Mamdani bulanık çıkarım yöntemiyle değerlendirilmiştir.

Zarsız bir fındık görüntüsünde fındığın beyaza yakın sarı bir renk taşıdığı görülmektedir. Bu rengin de şiddetinin yüksek olmasından dolayı *yüksek* bir enerjiye sahip olması beklenir. Ayrıca bütün renk değerleri birbirine yakın olacağı için karşıtlığının *az* düzensizliğinin düşük ve tek biçimliliğin yüksek olması beklenir. Buna ek olarak iteratif eşiklemeyle elde edilen ikili görüntüdeki siyah piksel sayısının tüm fındıktaki piksel sayısına oranı olan siyah alan oranının da sıfır olması gerekir. Fakat zaman zaman fındığın tırtıklı yüzey dokusunun yol açtığı gölgelerden dolayı bu alanın zarsız bir fındık için bile sıfırdan fazla olduğu gözlenmiştir. Bu yüzden ideal bir zarsız fındık için siyah bölge oranının düşük olması gerektiği düşünülebilir. Şekil 2.8’de zarlı ve zarsız fındık örnekleri Tablo 10’de ise bu fındıklara ait istatistikî öznitelikler görülmektedir. Tabloda yer alan değerler “ideal bir fındık için düşük siyah alan, düzensizlik, karşıtlık ve yüksek enerji ve homojenliğe sahip olmalıdır” şeklindeki varsayımı doğrulamaktadır.

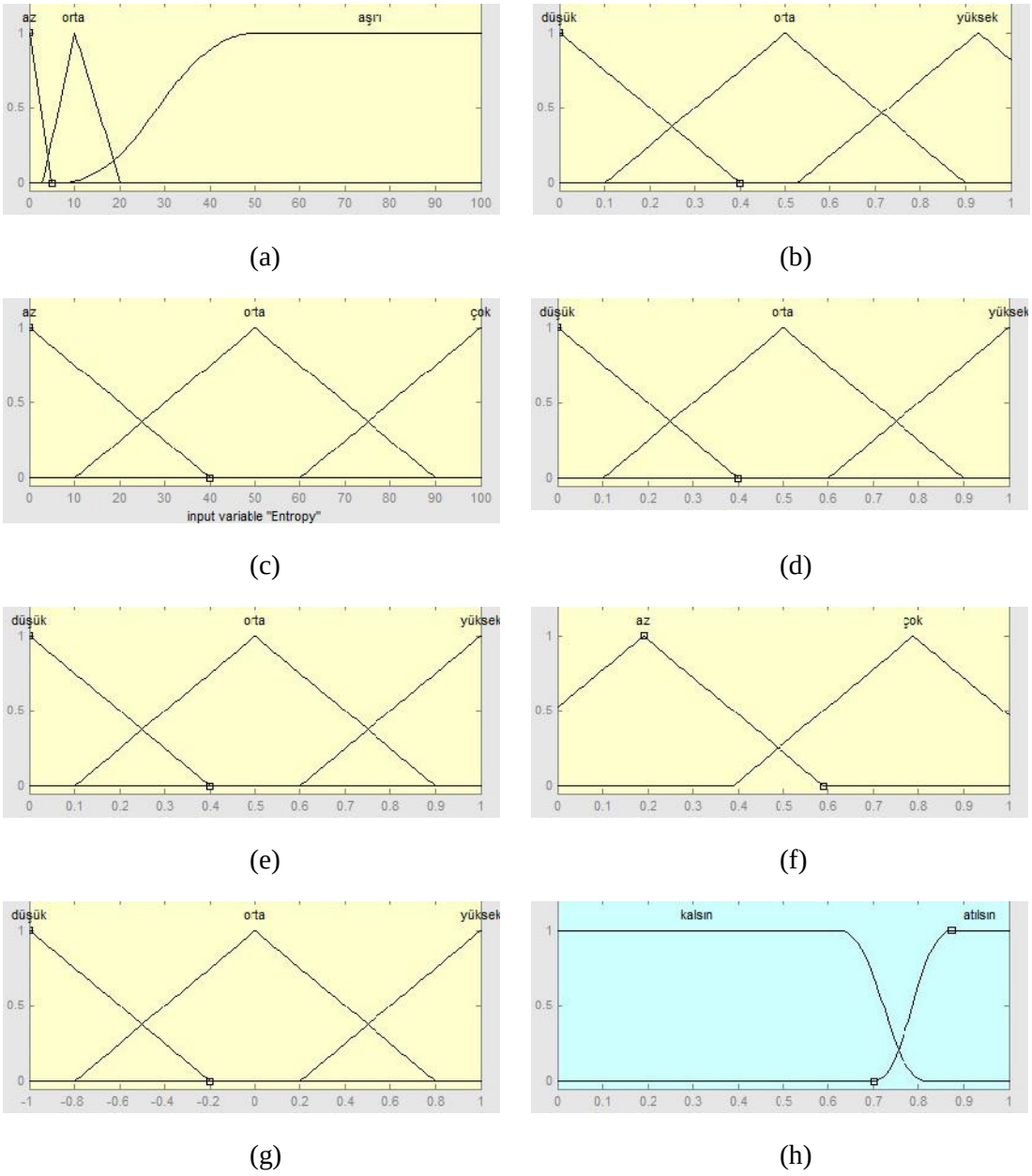
(a) (b)

Şekil 2.8. Zarlı ve zarsız fındık örnekleri (a) zarsız fındık (b) zarlı fındık.

Tablo 10. Şekil 2.8.'deki fındıklara ait istatistikî değerler.

	Zarsız fındık	*Zarlı fındık*
Karşıtlık	0.2167	0.2347
Düzensizlik	6.6434	7.46
Enerji	0.2709	0.1145
Homojenlik	0.9342	0.9049
İlinti	0.9227	0.9729
Siyah Alan Oranı	0.1	23.2007

Bu varsayım doğrultusunda istatistikî özniteliklerin alt ve üst sınırları da dikkate alınarak Şekil2.9'da görülen bulanık kümeler hazırlanmıştır.



(a) (b)

(c) (d)

(e) (f)

(g) (h)

Şekil 2.9 Bulanık sistemde giriş ve çıkış için oluşturulan bulanık kümeler. (a) siyah bölge oranı (b) enerji (c) düzensizlik (d) enerji (e) tek biçimlilik (f) karşıtlık (g) ilinti

Bulanık kümeler kullanılarak Tablo 11'de görülen on dört adet kural oluşturulmuştur.

Tablo 11. Bulanık mantık kuralları

Kural No	Kural
1	Eğer siyah alan oranı az ise fındık türü zarsızdır
2	Eğer siyah alan oranı yüksek ise fındık türü zarlıdır
3	Eğer siyah alan oranı yüksek ve karşıtlık yüksek ve enerji düşük ve tek biçimlilik düşük ve ilinti düşük ise fındık türü zarlıdır
4	Eğer siyah alan oranı orta ve karşıtlık yüksek ve enerji düşük ve tek biçimlilik düşük ve ilinti düşük İse fındık türü zarlıdır
5	Eğer Siyah alan oranı az ve karşıtlık yüksek ve düzensizlik düşük ve enerji düşük ve tek biçimlilik yüksek ve ilinti yüksek ise fındık türü zarlıdır
6	Eğer Siyah alan oranı az ve karşıtlık yüksek ve düzensizlik düşük ve enerji orta ve tek biçimlilik yüksek ve ilinti yüksek ise fındık türü zarsızdır
7	Eğer Siyah alan oranı az ve karşıtlık düşük ve düzensizlik orta ve enerji yüksek ve tek biçimlilik yüksek ve ilinti yüksek ise fındık türü zarsızdır
8	Eğer Siyah alan oranı az ve karşıtlık düşük ve düzensizlik düşük ve enerji yüksek ve tek biçimlilik yüksek ve ilinti yüksek ise fındık türü zarsızdır
9	Eğer Siyah alan oranı az ve karşıtlık düşük ve düzensizlik yüksek ve enerji yüksek ve tek biçimlilik yüksek ve ilinti yüksek ise fındık türü zarsızdır
10	Eğer Siyah alan oranı orta ve karşıtlık düşük ve düzensizlik orta ve enerji düşük ve tek biçimlilik düşük ve ilinti düşük ise fındık türü zarlıdır
11	Eğer Siyah alan oranı yüksek ve karşıtlık yüksek ve düzensizlik düşük ve enerji yüksek ve tek biçimlilik yüksek ve ilinti düşük ise fındık türü zarlıdır
12	Eğer Siyah alan oranı yüksek ve karşıtlık yüksek ve düzensizlik orta ve enerji yüksek ve tek biçimlilik yüksek ve ilinti düşük ise fındık türü zarlıdır
13	Eğer Siyah alan oranı yüksek ve karşıtlık yüksek ve düzensizlik orta ve enerji yüksek ve tek biçimlilik orta ve ilinti orta ise fındık türü zarsızdır
14	Eğer Siyah alan oranı az ve karşıtlık yüksek ve düzensizlik yüksek ve enerji orta ve tek biçimlilik yüksek ve ilinti yüksek ise fındık türü zarsızdır

Bu çalışmada kullanılan Mamdani yönteminde bulanıklaştırma işlemi için ağırlık merkezi (centroid) yöntemi kullanılmıştır. Şekil 2.10’de örnek yedi tane fındık ve tablo 12’da bu

fındıklar üzerinde bulanık sonuç çıkarım mekanizmasının ve insan verdiği kararlar karşılaştırmalı olarak yer almaktadır.

(a) (b) (c) (d) (e) (f) (g)

Şekil 2.10. Fındık Örnekleri

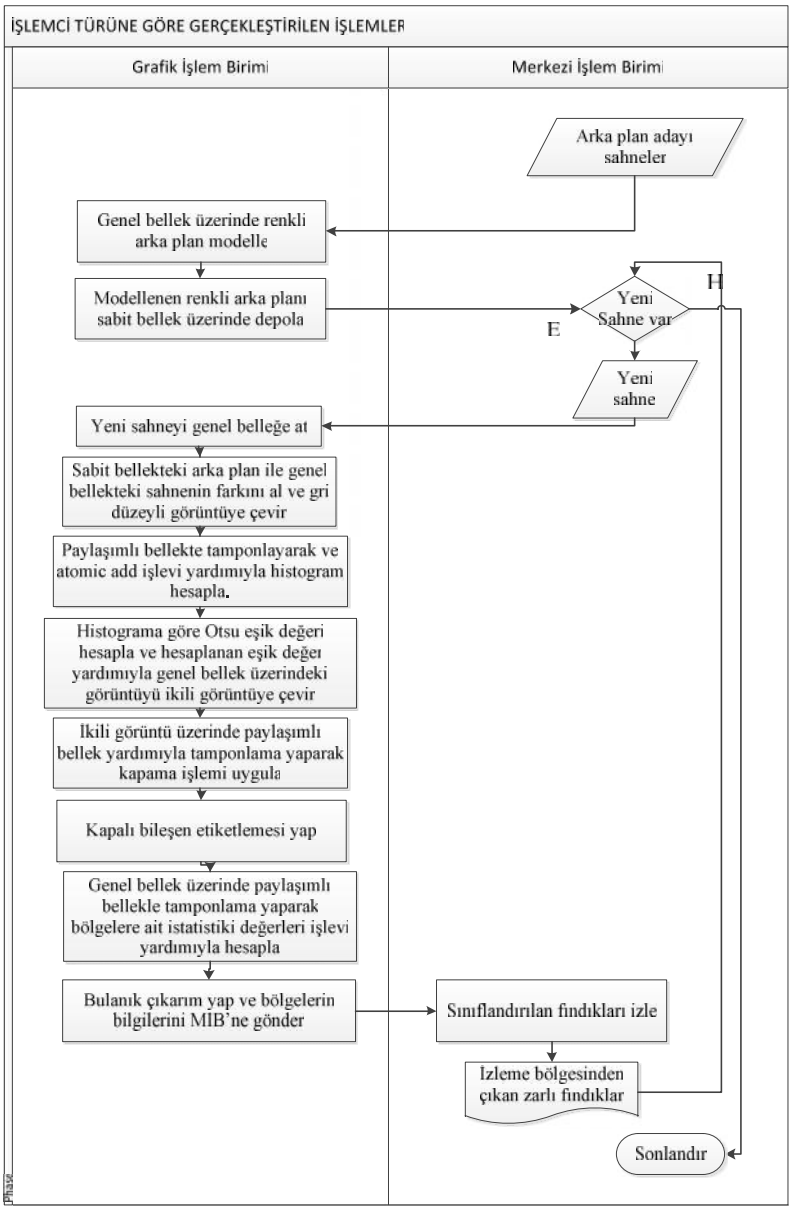
Tablo 12. Şekil 2.10'daki fındıklara ait istatistikî öznitelikler ve bulanık karar

Şekil 2.10 Numarası	Karşıtlık	Düzensizlik	Enerji	Tek Biçimlilik	İlinti	Siyah Alan Oranı	Bulanık Sonuç	Bulanık Çıkarım	İnsan Kararı
(a)	0.1266	6.0306	0.2191	0.9479	0.9588	1.2923	0.4298	Zarsız	Zarsız
(b)	0.1230	6.1112	0.2187	0.9498	0.9614	0.4974	0.4327	Zarsız	Zarsız
(c)	0.1504	6.4056	0.2014	0.9429	0.9736	11.3581	0.8200	Zarlı	Zarlı
d)	0.1792	6.7316	0.1521	0.9279	0.9722	12.9120	0.8205	Zarlı	Zarlı
(e)	0.2031	5.8498	0.2970	0.9373	0.9377	1.7318	0.4274	Zarsız	Zarsız
(f)	0.1271	6.1063	0.2513	0.9501	0.9733	4.9776	0.4000	Zarsız	Zarlı
(g)	0.1181	6.5537	0.2049	0.9432	0.9685	4.6923	0.4029	Zarsız	Zarsız

2.4. Sistemin GİB Üzerinde Koşut Olarak İcra Edilmesi

Geliştirilen sistem pek çok alt algoritmadan oluşan tümleşik bir sistemdir. Bütünü oluşturan alt algoritmalardan hemen hemen hepsi koşut programlama aracılığıyla hızlandırılabilen algoritmalardır. Bilhassa arka plan ile ön plan arası farkı bulma, ortanca süzgeciyle yumuşatma, histogram hesaplama, Otsu algoritması ile eşikleme, yayma, aşındırma gibi işlemler mükemmel derecede koşutlaştırılabilen algoritmalar arasındadır. Kaynak videoyu okuma ve hareketli bölgeleri ekranda gösterme işlemleri ise CUDA teknolojisiyle işletilemeyecek algoritmalardandır. Bölgelere ait istatistikî öznitelikleri çıkarma ve bulanık sınıflandırma (özellikle durulaştırma adımı) algoritmaları da koşut olarak icraya elverişli algoritmalardır. Fakat bir sahnede birden fazla fındık bulunabildiği

için her bir fındığın istatistikî özelliklerini bulmak ve her biri için aynı anda bulanık karar vermek aygıtın olanaklarını çok iyi tahlil edip çözüm geliştirmekle mümkün olan bir süreçtir. Sistemin çalışması sırasında MİB ve GİB'ne düşen işlem yükümlülükleri Şekil 2.11'de gösterilmektedir.



Şekil 2.11 Sistemin çalışması sırasında merkezi işlem birimi ve grafik işlem birimi arasındaki haberleşme ve her bir birimin gerçekleştirdiği işlemler.

Bütün bu işlemleri ayrıntılı bir şekilde açıklamak gerekirse;

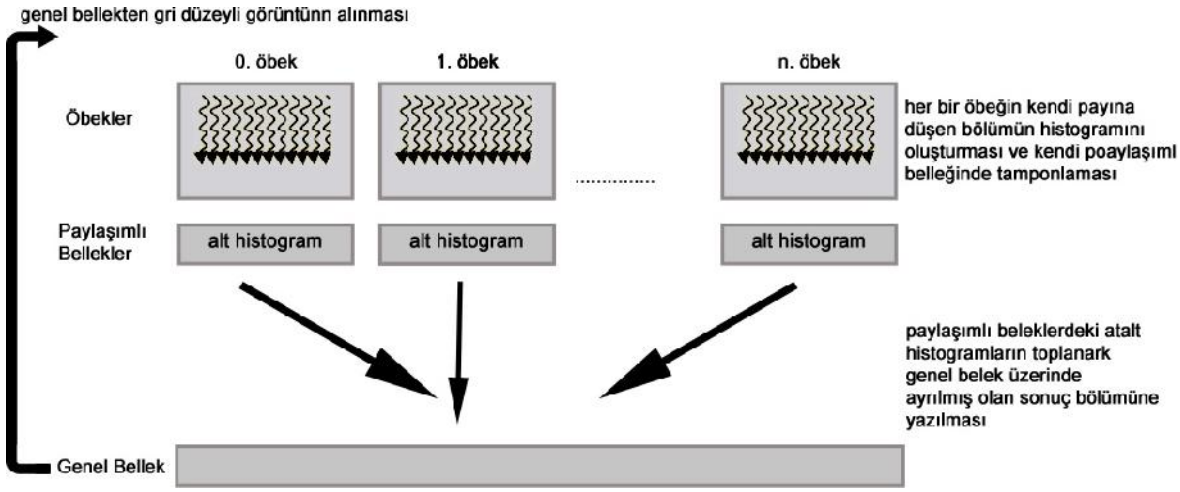
Arka plan belirlemek için belirli sayıda sahne görüntüsünün kullanılması sırasında her bir sahne MİB işlem birimi tarafından okunarak GİB'ne aktarılmış ve genel bellek üzerinde eş zamanlı bir işleme ile arka plan belirlenmiştir. Modellenen arka plan sabit belleğe aktarılmış ve her bir sahne için ön plandan çıkarılmak üzere depolanmıştır. Sabit belleğin aygıt üzerindeki çekirdekler tarafından işlenemeyeceği için sadece okuma amaçlı olarak kullanılması hedeflenmiştir.

Her bir sahne için kırmızı, yeşil ve mavi kanallar ayrıca okunarak GİB'nin genel belleğine aktarılmış ve sabit bellekteki arka plan görüntüsünden çıkarılarak fark görüntüsü elde edilmiştir. Elde edilen renkli fark görüntüsü yine aynı çekirdek işlevi içerisinde gri düzeyli görüntüye dönüştürülmüştür.

Bundan sonraki işlem adımı gri düzeyli fark görüntüsünün Otsu algoritması kullanılarak ikili görüntüye dönüştürülmesidir. Otsu Algoritması yardımıyla eşik değer belirlemek için histogram hesaplamaya gereksinim vardır. Histogram hesaplama sırasında bütün görüntü taranarak her bir piksel için histogram üzerinde piksele karşılık düşen değerin bir artırılması gerekmektedir. Bu işlemin ardışık programlama ile gerçekleştirilirse kolaylıkla doğru sonuçlandırılabilir. Fakat koşut programlama ile gerçekleştirilmesi halinde başarısızlık riski vardır. Zira farklı çekirdek işlevleri tarafından okunan aynı değere sahip farklı piksellerin histogramın aynı noktasını birer artırması gerekmektedir. Bir iş parçacığı bir histogram değerini okuduktan hemen sonra diğer bir parçacık o değeri bir arıtabilir. İlk parçacık da kendi okuduğu değeri diğerinin yaptığı işlemden haberdar olmadığı için bir artırarak histogram üzerinde aynı noktaya yazacaktır. Dolayısıyla normalde iki kez artırılması gereken değer sadece bir kez artırılacak ve yanlış bir sonuca ulaşılacaktır. Bu istenmeyen durumun önüne geçebilmek amacıyla üretici firma tarafından '*atomicAdd*' işlevi geliştirilmiştir. Bu işlev sayesinde bir parçacık belleğin bir bölgesinde işlem yapacaksa aynı bölgede işlem yapacak olan diğeri ilk yapacak olanı beklemekte ve birincisi işini gördükten sonra kendi görevini yerine getirmektedir. Bu da hız düşüşüne yol açsa da doğru sonuç elde etmek açısından zorunlu bir çözümdür.

Otsu algoritmasının icrası sırasında düşünebilecek bir diğer güçlük de verilerin tamponlanıp tamponlanmayacağıdır. Tamponlamanın hız kazandıracağı düşünülmektedir bu doğrultuda her bir öbek kendi başına düşen kısımda yerel histogram hesaplamakta bu

alt histogramı kendi paylaşımlı belleği üzerinde saklamakta diğer bir deyişle tamponlamaktadır. Her bir öbek kendi işini bitirdikten sonra alt histogramlar toplanarak genel bellek üzerindeki sonuç kısmın yazılmaktadır. Histogram hesaplama kısmına ait bu çalışma ilkesi Şekil 2.12'de sunulmuştur.



Şekil 2.12. Histogram hesaplama sırasında öbeklerin ve belleklerin kullanım şekli

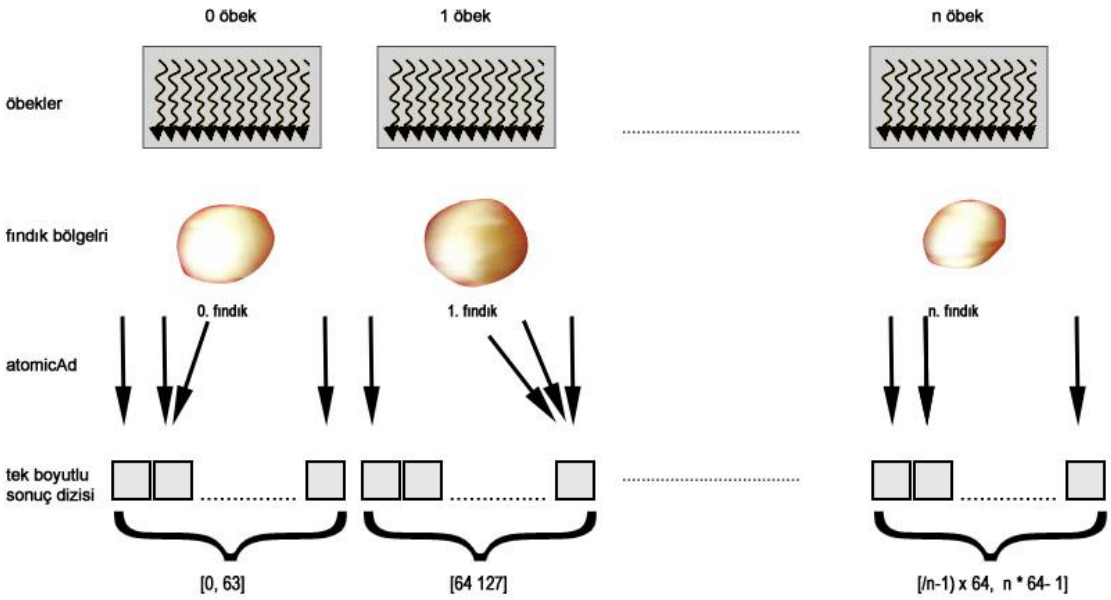
Otsu algoritması sonucu elde edilen değer ile genel bellek üzerinde yer alan fark görüntüsünü ikili görüntüye çevirebilme işlemi rahatlıkla gerçekleştirilebilmektedir. Genel bellek üzerinde artık ikili görüntüye çevrilmiş görüntü üzerinde paylaşımlı bellek yardımıyla yapılan tamponlama ile biçimsel kapama işlemi gerçekleştirilmektedir. Bütün bu işlemler veri MİB'ne gönderilmeksizin aygıt bellekleri üzerinde tatbik edilmektedir.

Bundan sonraki aşama olan kapalı bileşen etiketleme işlemi ise CUDA üzerinde icrası zahmetli olan bir işlemdir ancak Wu, Otto ve Suzuki tarafından önerilen yöntem [49] paylaşımlı bellek üzerinde tamponlama yapılarak gerçekleştirilmiştir. Ancak [48]'de de belirtildiği üzere bu yöntemde sonucunda bölgeler bir, iki, üç gibi ardışık numaralarla değil de kök pikselin tek boyutlu adresine göre etiketlenmektedir. Yine aynı kaynakta bu olumsuzluğun önüne geçilebilmesi için '*prefix sum*' algoritmasına başvurulması salık verilmiştir. Fakat bu çalışmada bant üzerindeki fındık sayısının azami miktarı bandın sığası gereği bilindiğinden etiketlenmiş ikili görüntü her bir bölge ayrı bir bölge üzerinde adres tutma işlemi gerçekleştirecek şekilde öbekler üzerine dağıtılmış, her bir bölgenin

etiket değeri iki boyutlu küçük bir dizi üzerine aktırılmıştır. Daha sonra bu dizi üzerinde tarama yapılmış her bir etiket numarası ardışık bir etiket numarasıyla değiştirilmiştir. Genel bellek üzerinde yer alan etiketlenmiş görüntü yeni ardışık numaralarla yeniden etiketlenmiştir. İlk bakışta zaman yitimine yol açan bu işlemlerin gereksiz olduğu kanısına varılabilir ancak sonraki adım olan bulanık karar verme işleminde her bir bölge kendi etiket değerine denk düşen numaralı öbekler üzerinde icra edileceği için etiket numaralarının çok yüksek olmaması şarttır. Kapalı bileşenleri tespit edebilmek ve ardışık numaralarla etiketleyebilmek amacıyla yazılan CUDA kodları Ek1'de verilmiştir.

GİB'nde bundan sonra yürütülen işlem adımları kodlama açısından önceki adımlardan daha zorlu ve dikkat isteyen adımlardır. Zira aynı görüntünün birden fazla bölgesinin aynı anda ve farkı bellek bölgelerinde işlenmesi gerekmektedir. Her bir bölgenin GDOM'nin aynı histogram hesabında olduğu gibi *atomicAdd* işlevi kullanılarak hesaplanması gerekmektedir. Hesaplamaların kesinlikte doğru ve hızlı hesaplanabilmesi için her bir bölgeyi değerlendirme işlemi ayrı birer öbek üzerinde gerçekleştirilmiştir. Burada karşılaşılan diğer bir güçlük de veri boyutudur. Her bir GDOM 8 x 8 lik iki boyutlu dizidir. Birden fazla bölge ya da fındık olması durumunda bu boyut üçe çıkmaktadır. Ancak önceki bölümlerde de belirtildiği üzere CUDA üzerinde birden fazla boyutlu veri işleme yeteneği çok kısıtlıdır ve bu yüzden büyük boyutlu veriler tek bir işaretçi dizisiyle ifade edilerek MİB belleklerine aktarılır. Bu çalışmada da üç boyutlu GDOM dizileri tek boyutlu işaretçi ile gösterilerek bellek üzerinde işlenmiştir. Duruma göre her bir öbekteki iş parçacıkları kendi üzerine düşen hesaplamayı yaptıktan sonra sonucu tek boyutlu dizi üzerine karşılık düşen bölgeye aktarmaktadır. Sistemin bu yapısı Şekil 2.13'te görülmektedir.

Olasılık yoğunluk işlevi, düzensizlik gibi hesaplamalarda karşılaşılan zorunluluk da gerçek sayılarla işlem yapmaktır. Fakat gerçek sayılarla işlem yapmak tam sayılarla işlem yapmaktan daha fazla zaman gerektirmektedir. Bu durumun önüne geçebilmek için olasılık yoğunluk bulma benzeri işlemler gerçekleştirilirken pay ve payda değerleri ayrı ayrı tam sayılar olarak hesaplanmış ve sonuç bu pay ve payda değerleri gerçek sayıya dönüştürüldükten sonra bulunmuştur.



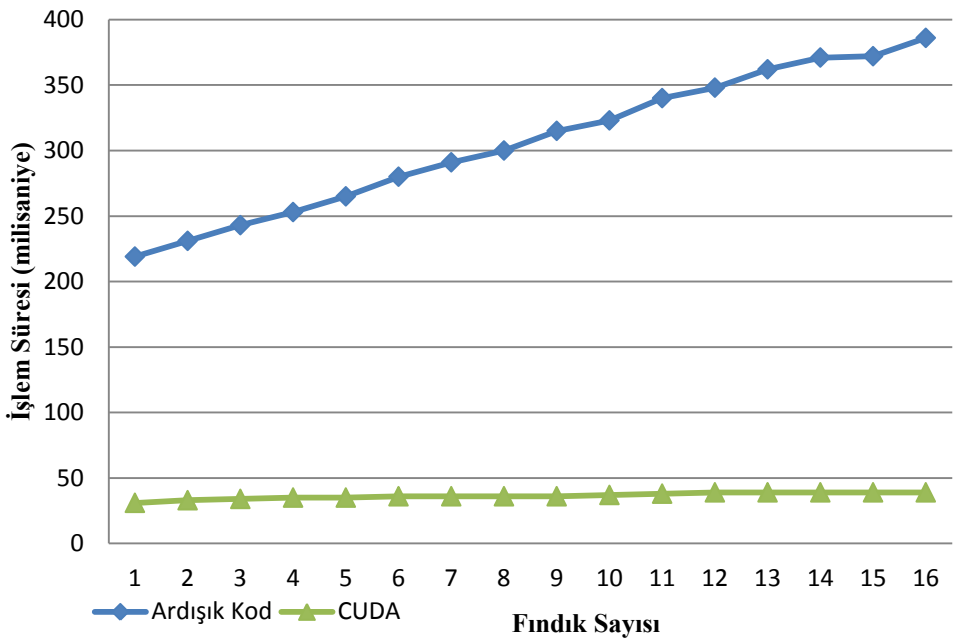
Şekil 2.13. Çok sayıda fındığa ait istatistikî özniteliklerin tek boyutlu işaretçi ile gösterilmesi ve öbeklere göre sonuç yazma bölgeleri

3. SONUÇLAR

Geliştirilen sistem 640 x 360 boyutunda videolar üzerinde hesaplama yeteneği 2.1 olan NVIDIA ekran kartı kullanılarak icra edilmiş ve arka plan belirlendikten sonraki herhangi bir sahnenin (sahne okuma ve sonucu kullanıcıya gösterme zamanı hariç) azami 39 milisaniyede işlendiği ölçülmüştür. Aynı işlemler Intel (R) Core (TM)2 Duo T9600 @2.80 GHz MİB'nde icra edildiğinde 167 milisaniyede gerçekleştirilebilmiştir.

CUDA teknolojisi kullanılarak yapılan koşut programlamada paylaşımlı bellek kullanılarak tamponlama yapıldığında sistemin hızlandığı görülmüştür. Ayrıca verilerin merkezi işlem birimine aktarılmadan grafik işlem biriminde işlenmesine devam edildiğinde taşıma zamanından kaynaklanan gecikmenin yaşanmadığı ve zaman kazancı sağlandığı görülmüştür.

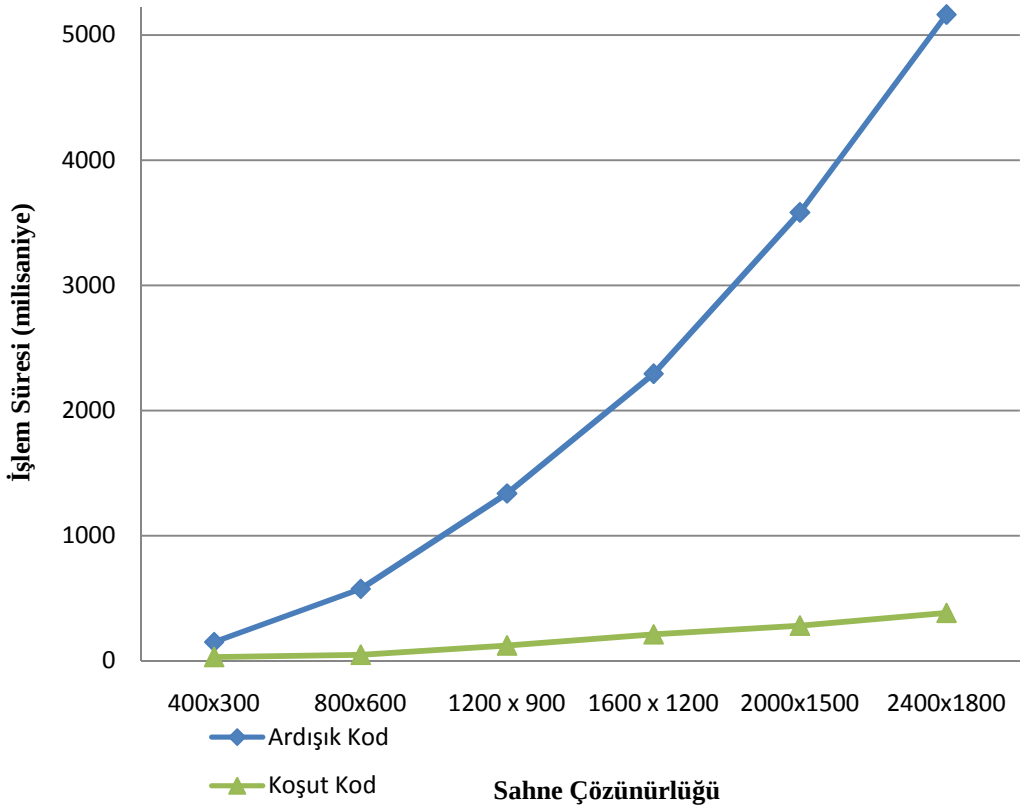
Artan fındık sayısı karşısında tek bir sahne ve arka plan üzerinde üzerinde yapılan ölçümlerde Şekil 3.1'de yer alan sonuçlar elde edilmiştir.



Şekil 3.1. Değişik sayıda fındık içeren 640x360 çözünürlüklü sahneler için ardışık ve koşut hesaplama süreleri

Görüldüğü üzere fındık sayısının artışı karşısında ardışık işletilen programda tek bir işlemciye düşen yük artacağı için hesaplama süresi de artmaktadır. Buna karşılık olarak CUDA teknolojisi kullanılarak yapılan icrada ise artan fındık sayısı işlem süresinde kayda değer bir artışa yol açmamaktadır. Bunun sebebi olarak görevlendirilebilen öbek sayısının fındık sayısını rahatlıkla karşılaması ve her bir iş parçacığının eş zamanlı olarak çalışabilmesi öne sürülebilir.

Geliştirilen algoritmayı değerlendirmek için bir diğer parametre de artan sahne çözünürlüğüdür. Yazılımın değişen çözünürlükte sahneler üzerinde icra edilmesiyle Şekil 3.2'de görülen sonuçlar elde edilmiştir.



Şekil 3.2. Artan sahne çözünürlükleri karşısında ardışık ve koşut hesaplama süreleri

Geliştirilen yazılım değişik sahne boyutlarına sahip videolar üzerinde tek bir sahne ve arka plan için icra edildiğinde ise CUDA ile koşut programla sonuçlarının ardışık

programlama değerlerinden daha iyi olduğu görülmektedir. Bu verimi elde ederken de en önemli katkıyı iş parçacığı sayısının rahatlıkla artırılabilmesinin sağladığı açıktır.

Değişik videolardan elde edilen 1027 fındık üzerinde bulanık mantık mekanizmasının verdiği kararlar insanın verdiği kararlarla karşılaştırılmıştır. İnsanın kararlarının %100 doğru kabul edildiği değerlendirmede insanın kararıyla örtüşen bulanık çıkarımlar başarılı kabul edilmiştir. 778 fındık görüntüsünde hem insan hem de bulanık çıkarım mekanizması fındıkların zarlı oldukları sonucuna ulaşmıştır, 77 fındık hakkında hem bulanık çıkarım hem de insan zarlı kararı vermiştir. 172 görüntü üzerinde ise insan ve makine farklı kararlar vermişlerdir. Toplamda %83 başarı sağlanmıştır.

Bu çalışmada önerilen yöntem bildiri olarak INISTA 2012 adlı bilgi şölenine sunulmuş [69] ve çalışmanın bildiriler kitapçığında basılması kabul edilmiştir.

4. ÖNERİLER

Görüntü işleme ve bilgisayar görmesi algoritmaları icra edilirken özellik de işlem süresinin önemli olduğu durumlarda koşut programlaya başvurulabilir. Eğer algoritmaların icra edildiği bilgisayarda NVIDIA firması tarafından üretilmiş bir ekran kartı varsa programın CUDA teknolojisi kullanılarak yazılması ciddi oranda hız kazandıracaktır.

CUDA ortamında program geliştirirken aygıtın (grafik işlem birimi) mimarisi göz önünde bulundurulmalı ve bellek çeşitlerinin özellikleri dikkate alınarak adım atılmalıdır.

Grafik işlem birimi üzerinde geliştirilen algoritmada mümkünse tamponlama yapılmalı ve verinin işlemciler arası (merkezi ve grafik) aktarımında zaman kaybolacağı unutulmamalıdır. Dolayısıyla taşıma işleminden olabildiğince kaçınılmalıdır.

Histogram hesabı ve gri düzey oluşum matrisi çıkarma gibi iş parçacıklarının ortak bir bellek bölgesine yazma gerçekleştireceği durumlarda veri doğruluğunun önemli olduğu ve ortak bölgeye yazımda tutarsızlık sorunlarının yaşanabileceği unutulmamalıdır. Bu olumsuzluğun önüne geçebilmek amacıyla atomik işlevler kullanılmalıdır.

Hareketli cismin bir sonraki adımda nerede olacağının bilinmesi hareketlinin izlenmesi sırasında yarar sağlayabilir. Yalınlığından ve gözlemlere dayalı olarak kestirimi sürekli güncellemesinden dolayı Kalman süzgeci doğrusal sitemler için kullanışlı bir teknik olarak karşımıza çıkmaktadır.

Bulanık mantık görüntüye ait istatistikî değerleri tasnif etmede faydalı olabilecek bir yöntemdir. Burada kuralları doğru şekilde ayarlamanın ve giriş çıkış kümelerini düzgün belirlemenin önemi karşımıza çıkmaktadır.

CUDA ortamında yazılım geliştirilirken üzerinde durulması gereken önemli bir nokta da ipliklerin uyumunun sağlanmasıdır. İpliklerden bazıları işlerini diğer ipliklerden önce bitirebilir. Böyle bir durumda sonucun daha erken elde edilmesi mümkün ise de bazı iplikler işlerini yapmadığı için doğru sonuç elde edilemeyecektir. Bu olumsuzluğun önüne geçebilmek için bloklama yapılarak işini önce bitiren iş parçacığının ötekileri beklemesi sağlanmalıdır.

5. KAYNAKÇA

1. Bettencourt, K. ve Somers, D., Effects of task difficulty on multiple object tracking performance, Journal of Vision, 7 (2007) 12 – 14.

2. Talu, M., Nesne Takip Yöntemlerinin Sınıflandırılması, İstanbul Ticaret Üniversitesi Fen Bilimleri Dergisi, 1 (2010) 45 - 63.

3. Demir, M. ve Ulutaş, M., Grafik ve Merkezi İşlem Birimlerinin Performanslarının Video İşleme İçin Karşılaştırılması, 4. Mühendislik ve Teknoloji Sempozyumu, Nisan 2011, Ankara, Bildiriler Kitabı 1 :436 - 441.

4. Fukunaga, K. ve Hostetler, L., The estimation of the gradient of a density function, with applications in pattern recognition, IEEE Transactions on Information Theory, 1 (1975) 32 - 40.

5. Anonim, Open Source Computer Vision Library ,Intel Corporation, USA, 2001

6. Lowe, D., Object recognition from local scale-invariant features, The Proceedings of the Seventh IEEE International Conference on, September 1999, Colombia, Bildiriler Kitabı 2 :1150 - 1157.

7. Nixon, M.S ve Aguado, A.S, Feature Extraction and Image Processing, Second Edition, Academic Press, Great Britain, 2008

8. Dhond, U., Structure from stereo-a review, Systems, Man and Cybernetics, 19 (1989) 1489 - 1510.

9. Nagel, H., On the estimation of opticalflow: Relations between different approaches and some new results, Artificial Intelligence, 3 (1987) 299 - 324.

10. Lucas, B. ve Kanade, T., An Iterative Image Registracion Technique with an Application to Stereo Vision, Proc 7th Internationalk Conference on Artifical Intelligence (IJCAI), Ağustos 1981, British Colombia, Bildiriler Kitabı I :674 - 679.

11. Yılmaz, A., Kernel-based object tracking using asymmetric kernels with adaptive scale and orientation selection, Machine Vision and Applications, 22 (2011) 255 - 268.

12. Talu, M., Türkoğlu, İ. ve Cebeci, M., Ortalama Kaydırma Algoritması Kullanarak Gerçek Zamanlı Çekirdek Tabanlı Nesne Takibi, IEEE 18. Sinyal İşleme ve İletişim Uygulamaları Kurultayı, Nisan 2010, Diyarbakır, Bildiriler Kitabı 1 :328 - 331.

13. http://en.wikipedia.org/wiki/Bhattacharyya_distance, Bhattacharyya_distance . 12 Aralık 2011

14. Özgen, N., Bilgisayar Kontrollü Hedef Takibi, Yüksek Lisans Tezi, Gazi Üniversitesi, Fen Bilimleri Enstitüsü, Ankara, 2008

15. Nixon M. ve Aguado, A., Feature Extraction ve Image Processing, Second Edition, Academic Press in an Imprint of Elsevier, United Kingdom, 2008

16. Barnard, S.T. ve Fisher, M.A., Stereo Vision, Encyclopedia of Artificial Intelligence, 1987, 1083-1090,

17. Beauchemin, S. ve Barron, J., The computation of optical flow, ACM Computing Surveys , 27 (1995) 433 - 466.

18. Horn, B. ve Schunck, B., Determining Optical Flow, Artifical Ingelligence, 17 (2003) 185 - 203.

19. Chalmers, A., Practical Parallel Processing, First Edition, International Computer Press, England, 1996.

20. Demir, M., Ulutaş, M. ve Odabaş, E., Asmuth Bloom Sır Paylaşım Tekniğinin Hızlandırılması İçin Koşut Programlama, Elektrik-Elektronik Bilgisayar Sempozyumu, Ekim 2011, Elazığ, Bildiriler Kitabı 2 :10 - 13.

21. Amdahl, G., Validity of the single processor approach to achieving large scale computing capabilities, AFIPS '67, Nisan 1967, California, Bildiriler Kitabı 1 :483 - 485.

22. http://en.wikipedia.org/wiki/Amdahl's_law Amdahl's_Law 11.02.2012

23. Gustafson, J., Montry, G. ve Benner, R., Development of Parallel Methods For a 1024 Processor Hypercube, Scientific and Statistical Computing, 9 (1988) 609 - 638.

24. http://en.wikipedia.org/wiki/Gustafson's_law Gustafson’s Law 11.02.2012

25. Wilkinson, B. ve Allen, M., Parallel Programming, First Edition, Prentice Hall, New Jersey, 1999.

26. http://top500.org 12.02.2012

27. http://www.hardwareinsight.com/nvidia-cuda/ 12.02.2012

28. Temizel, A., Halıcı, T., Logoğlu, B., Taşkaya Temizel, T., Ömrüuzun, F., Karaman, E., HWU, W-M., w., CPU Computing Gems, Volume 1, Elsevier, 2011

29. http://en.wikipedia.org/wiki/OpenCL 21.03.2012

30. http://www.khronos.org/members/promoters/, 10.01.2012

31. Tanrıverdi, İ., Bir Grafik İşlemci (GPU) İçin MPI(Message Passing Interface) Yazılım Kitaplığının Tasarımı ve Gerçekleştirimi, Yüksek Lisans Tezi, Hacettepe Üniversitesi Fen Bilimleri Ensitüsü, Ankara, 2010.

32. NVIDIA Corporation, NVIDIA CUDA, Programming Guide, Version 4.0, Santa Clara USA, 2011.

33. Asano, S., Maruyama, T. ve Yamaguchi, Y., Performance comparison of FPGA, GPU and CPU in image processing, Field Programmable Logic and Applications, Eylül 2009, Prag, Bildiriler Kitabı 1 :126 - 131.

34. Gregg, C. ve Hazelwood, K., Where is the data? Why you cannot debate CPU vs. GPU performance without the answer, Performance Analysis of Systems and Software (ISPASS), Nisan 2011, Texas, USA, Bildiriler Kitabı 1 :133 - 144.

35. ACM SIGMETRICS, Performance Evaluation Review - Special issue on the 1st international workshop on performance modeling, benchmarking and simulation of high performance computing systems, Issue 4, Volume 38, 2011,

36. Niblack, W., An introduction to digital image procesing 115-116, Prentice Hall, 1986.

37. Kapur, J.N., A new method for gray-level picture thresholding using the entropy of its histogram, Comp. Vision, Graph. Im. Process. 29 (1985) 273-825

38. White, J., Image Thresholding for Optical Character Recognition and Other Applications Requiring Character Image Extraction, IBM Journal of Research and Development, 27 (1983) 400 - 411.

39. Otsu, N., A Threshold Selection Method from Gray-Level Histogram, Systems Man and Cybernetics, 9 (1979) 62 - 66.

40. http://en.wikipedia.org/wiki/Otsu's_method Otsu’s Method 19.12.2011

41. Sonka, M., Hlavac, V. ve Boyle, R., Image Processing: Analysis and Machine Vison, Volume 1, Second Press, pg: 180, Cengage Learning.

42. Soille, P., Morphological Image Analysis: Principles and Applications, Springer-Verlag, 1999, 173-174.

43. Dillencourt, M., Samet, H. ve Tamminen, M., A general approach to connected-component labeling for arbitrary image representations, Journal of the ACM, 39 (1992) 253 - 280.

44. Suziki, K., Horiba, I. ve Sugie, N., Fast Connected-Component Labeling Based on Sequential Local Operations in the Course of Forward Raster Scan Followed by Backward Raster Scan, 15th International Conference on Pattern Recognotion, Eylül 2000, Barcelona, Bildiriler Kitabı 2 :434 - 437.

45. Cypher, R., Algorithms for image component labeling on SIMD mesh-connected computers, Computers, 39 (1990) 276 - 281.

46. Park, J., Looney, C. ve Chen, H., Fast Connected Component Labeling Algorithm Using A Divide an Conquer Technique, Matrix , 4 (2000) 4 - 7.

47. Wang, K., Chia, T. ve Chen, Z., Parallel Execution of a Connected Component Labeling Operation on a Linear Array Architecture, Journal of Informatıon Science and Engineering, 1 (2003) 353 - 370.

48. Kalentev, O., Rai, A., Kemnitz, S., and Scheneider, R., Connected Component Labelling on 2D grid using CUDA, Journal of Parallel Distrbuted Computing, in press

49. HWU, W-M., CPU Computing Gems, Volume 1, 2011, Elsevier

50. Umbaugh, S. E., Computer Vision and Image Processing, Prentice Hall Inc., London, 1998, 153- 171

51. Kalman, R., A New Approach to Linear Filtering and Prediction Problems, Journal of Basic Engineering, 82 (1960) 35 - 45.

52. Dote, Y. ve Anbo, K., Combined Parameter and State Estimation of Controlled Current Induction Motor Drive System via Stochastic Nonlinear Filtering technique, -IAS Ann. Mtg. Conference, 1979, ., Bildiriler Kitabı 1 :838 - 842.

53. Gordon, N., Salmond, D. ve Smith, A., Novel approach to nonlinear/non-Gaussian Bayesian state estimation, Radar and Signal Processing, 140 (1993) 107 - 113.

54. Can, A., Bayesçi Kestirim Teknikleri ile Hedef Takibi, Yüksek Lisans Tezi, İstanbul Üniversitesi, Fen Bilimleri Estitüsü, İstanbul, 2008

55. MMarcenaro, L., Ferrari, M. ve Marchesotti, L., Multiple Object Tracking Under Heavy Occlusions by Using Kalman Filters Based On Shape Matching, Image Processing. Proceedings. International Conference, Aralık 2002, Rochester, Bildiriler Kitabı 3 :341 - 344.

56. Weng, S., Kuo, C. ve Tu, S., Video objecttracking using adaptive Kalman Filter, Journal of Visual Communication and Image Representation, 17 (2006) 1190 - 1208.

57. Wu, C., Chung, Y. ve Chung, P., A Kalman Filtering Basaed Data Fusion For Object Tracking, Industrial Electronics and Applications (ICIEA), Haziran 2010, Taichung , Bildiriler Kitabı 1 :2291 - 2295.

58. Yılmaz, M., Multiple Target Tracking Using Multiple Cameras Yüksek Lisans Tezi, ODTÜ Fen Bilimleri Enstitüsü, Ankara, 2008

59. Wan, E. ve Van der merwe, R., The Unscented Kalman Filter For Nonlinear Estimation, Adaptive Systems for Signal Processing, Communications, and Control Symposium, 2000, Lake Louse Alta, Canada, Bildiriler Kitabı 1 :153 - 158.

60. Filozoficzny, R., Three-valued Logic, Lukasiewicz, 5 (1920) 170 - 171.

61. Blanck, M., Vagueness an Exercise in Logical Analysis, Philosophy of Science, 4 (1937) 427 - 455.

62. Zadeh, L., Fuzzy Sets, Information and Control, 8 (1965) 338 - 353.

63. Sugeno, M., An introductory survey of fuzzycontrol, Information Sciences, 1 (1985) 59 - 83.

64. Tsukamoto, Y., An Approacy to Fuzzy Reasoning Method, Advances in Fuzzy Set Theory and Applications, Ragade, K. ve Yager, R.R., 1, 1979, 137-149, North-Holland, Amsterdam

65. Mamdani, E. ve King, P., The application of fuzzy control systems to industrial processes, Automatica, 13 (1977) 235 - 242.

66. Haralick, R., Shanmugam, K. ve Dinstein, I., Textural Features for Image Classification, Man and Cybernetics, 3 (1973) 610 - 621.

67. Demirhan, A. ve Güler, İ., Özörgütlemeli Harita Ağları ve Gri Düzey Eş Oluşum Matrisleri İle Görüntü Bölütleme, Gazi Üniversitesi Mühendislik Mimarlık Dergisi, 25 (2010) 285 - 291.

68. Ekinci, M. ve Gedikli, E., Background Estimation Based People Detection and Tracking for Video Surveillance, Lecture Notes in Computer Science, 1 (2003) 421 - 429.

69. Demir, M., Ulutaş, M. ve Türe, H., Fuzzy Logic Based Identification of Partially Skinned Hazelnuts on a Conveyor Belt, International Symposium on Innovations in Intelligent Systems and Applications, Temmuz 2012, Trabzon

6. EKLER

Ek1 Kapalı Bileşen Etiketleme Algoritmasının CUDA Ortamında Kodlanması

```
#include <cuda.h>
#include <cuda_runtime.h>
#include <math.h>
#define indis(_x,_yuk,_y) (_x* _yuk + _y)
#define siraBul(a,b,c,d) ( a + b * c * d);

#define azamiFindikSayisi 20


#pragma region hesapYordamlari
__device__ int  kokBul(int *sonuc, int elementAdress)
        {
            while ((int)sonuc[elementAdress] != elementAdress)
                elementAdress = (int)sonuc[elementAdress];
            return elementAdress;
        }
__device__ void birlestir(int *sonuc, int elementAdress0, int elementAdress1)
        {
            int kok0 = kokBul(sonuc, elementAdress0);
            int kok1 = kokBul(sonuc, elementAdress1);
            if (kok0 < kok1) sonuc[kok1] = kok0;
            if (kok1 < kok0) sonuc[kok0] = kok1;
        }
__global__ void baslangic(int *kaynak, int *sonuc, int gen, int yuk,int boy)
{
        unsigned int i = indis(blockIdx.x,blockDim.x , threadIdx.x);
    unsigned int j = indis(blockIdx.y,blockDim.y , threadIdx.y);
        int sira = siraBul(i , j, blockDim.x , gridDim.x);

        if(sira >= 0 && sira < boy)
        {
                if(kaynak[sira]==0)
                        sonuc[sira]= 0;
                else
                        sonuc[sira]= sira;
                __syncthreads();
        }
        else
                return ;
}
__global__ void ula(int *kaynak, int *sonuc, int gen, int yuk, int boy)
{
        int i = indis(blockIdx.x,blockDim.x , threadIdx.x);
    int j = indis(blockIdx.y,blockDim.y , threadIdx.y);
        int sira = siraBul(i , j, blockDim.x , gridDim.x);
        if(sira>=0 && sira< boy)
          {
                if(kaynak[sira] == 0)
                        return;

        for(int x = -1; x<= 1; x++)
               for( int  y=-1; y<=1 ; y++)
                     {
                            int indisi = sira + indis(x, yuk , y);
                                if (x == 0 && y == 0)
                   continue;
                                if (kaynak[indisi] == kaynak[sira] && kaynak[sira] !=0)
                   birlestir(sonuc, indisi, sira);
                  }
```

Ek1'in Devamı

```
           __syncthreads();
        }
        else
                return ;
}
__global__ void son(int *kaynak, int *sonuc, int gen, int yuk, int boy)
{
        int i = indis(blockIdx.x,blockDim.x , threadIdx.x);
    int j = indis(blockIdx.y,blockDim.y , threadIdx.y);
        int sira = siraBul(i , j, blockDim.x , gridDim.x);
        if(sira>=0 && sira< boy)
        {
sonuc[sira]= kokBul(sonuc, sira);
                __syncthreads();
        }
        else
                return ;
}

__global__ void isaretle(int *ikili,  int boyu,    int *yerler)
{
        __shared__ int sonuc[20][20];

        for(int i=0; i<20; i++)
                sonuc[threadIdx.x][i] = 0;
           __syncthreads();

    int i = threadIdx.x + blockIdx.x * blockDim.x;
    int stride = blockDim.x * gridDim.x;
#pragma region dongu
    while (i < boyu)
        {
                if(ikili[i] !=0)
                {
                        int varmi = -1;
                        for(int k=0; k<20; k++)
                        {
                                if(sonuc[threadIdx.x][k] == ikili[i])
                                {
                                        varmi = 1;
                                        break;
                                }
                        }
                        if(varmi==-1)
                        {
                                for(int k=0; k<20; k++)
                                {
                                        if(sonuc[threadIdx.x][k] == 0)
                                        {
                                                sonuc[threadIdx.x][k] = ikili[i];
                                                break;
                                        }
                                }
                        }
                }
                i += stride;
        }
#pragma endregion dongu

        for(int i=0;i<20; i++)
        {
                for(int j=0; j<20; j++){

                                for(int k=0; k<400; k++)
                        {
```

Ek1'in Devamı

```
if(yerler[k] == 0)


{
                                        yerler[k] = sonuc[i][j];
                                        break;
                                }
                                if(yerler[k]==sonuc[i][j])
                                {
                                        break;
                                }
                        }
                }
        }
}
#pragma endregion
#pragma region etiketDuzeltmeYordamlari
__global__ void etiketle(int *ikili, int boyu, int *yerler, int *yeniYerler, int sayi)
{
        unsigned int i = indis(blockIdx.x,blockDim.x , threadIdx.x);
    unsigned int j = indis(blockIdx.y,blockDim.y , threadIdx.y);
        int sira = siraBul(i , j, blockDim.x , gridDim.x);

        if(sira>= 0 && sira<boyu)
        {
                if(ikili[sira] ==0)
                        return;
                for(int k=0;k<sayi; k++)
                {
                        if(ikili[sira]== yerler[k])
                        {
                                ikili[sira] = yeniYerler[k];
                                break;
                        }
                }
                __syncthreads();
        }
        else
                return;
}
// bolgeleriTespitEt<<<M,N>>>(ay_sonuc, gen, yuk, boy, ay_solX, ay_solY, ay_sagX, ay_sagY);
__global__ void bolgeleriTespitEt(int *ikili, int gen, int yuk,  int boy, int *solX ,int *solY,
int *sagX, int *sagY)
{
        unsigned int i = indis(blockIdx.x,blockDim.x , threadIdx.x);
    unsigned int j = indis(blockIdx.y,blockDim.y , threadIdx.y);
        unsigned int sira = siraBul(i , j, blockDim.x , gridDim.x);

        if(sira>=0 && sira< boy)
        {
                if(ikili[sira] ==0)
                        return;

                else
                {
                    int x =  (int)(sira / yuk);
                    int y = (int)(sira % yuk);

                   __syncthreads();

                   if(x> gen *yuk)
                           x = -1;

                   if(x== 1061109567)
                           x == -5;
```

Ek1'in Devamı

```
                    atomicMin(&solX[ikili[sira]-1], x);
                    atomicMin(&solY[ikili[sira]-1], y);
                    atomicMax(&sagX[ikili[sira]-1], x);
                    atomicMax(&sagY[ikili[sira]-1], y);



                }
                __syncthreads();
        }
        else
        return;
}
#pragma endregion etiketDuzeltmeYordamlari

extern "C" void __declspec(dllexport) __stdcall etiketle2(int *kaynak, int *sonuc, int gen, int
yuk,  int *solX , int *solY, int *sagX, int *sagY, int M , int N )
{


        int boy = gen * yuk;

#pragma region aygit_uzerindeki_diziler
        int *ay_kaynak; // gri düzeyli görüntü
        int *ay_sonuc;  // ikili ve işaretlenmiş görüntü
        int *ay_yerler; // bölgelerin başlangıç konumları daha sonra gerek
        int *ay_solX;
        int *ay_solY;
        int *ay_sagX;
        int *ay_sagY;

        int *ay_yeniYerler;


#pragma endregion aygit_uzerindeki_yerler

#pragma region host_uzerndeki_yerler

        int *yerler = (int*)malloc(sizeof(int) * 400);  // bölgelerin merkezi işlem birimindeki
yerleri
        int *yeniIndisler   = (int*)malloc(azamiFindikSayisi * azamiFindikSayisi);
        int *elemanSayilari = (int*)malloc(sizeof(int) * azamiFindikSayisi);

#pragma endregion host_uzerndeki_yerler
#pragma region aygitta_yer_acma
        cudaMalloc((void**)&ay_kaynak, sizeof(int) * gen * yuk);
        cudaMalloc((void**)&ay_sonuc, sizeof(int) * gen * yuk);
        cudaMalloc((void**)&ay_yerler, sizeof(int) * 400);
        cudaMalloc((void**)&ay_solX, sizeof(int) * azamiFindikSayisi);
        cudaMalloc((void**)&ay_solY, sizeof(int) * azamiFindikSayisi);
        cudaMalloc((void**)&ay_sagX, sizeof(int) * azamiFindikSayisi);
        cudaMalloc((void**)&ay_sagY, sizeof(int) * azamiFindikSayisi);
        cudaMalloc((void**)&ay_yeniYerler, sizeof(int) * 400);




#pragma endregion aygitta_yer_acma

#pragma region bellege_tasima_ve_baslangic_degerleri
        cudaMemset(ay_solX, 999999, sizeof(int) *azamiFindikSayisi);
        cudaMemset(ay_solY, 999999, sizeof(int) *azamiFindikSayisi);
        cudaMemset(ay_sagX, -1, sizeof(int) *azamiFindikSayisi);
        cudaMemset(ay_sagY, -1, sizeof(int) *azamiFindikSayisi);

        cudaMemcpy(ay_kaynak, kaynak, sizeof(int) * gen * yuk, cudaMemcpyHostToDevice);
```

Ek1'in Devamı

```
cudaMemset(ay_yerler, 0, sizeof(int) * 400);


#pragma endregion bellege_tasima_ve_baslangic_degerleri

#pragma region baslangic_etiketlemesi
        baslangic<<<M,N>>>(ay_kaynak, ay_sonuc, gen, yuk, boy);
        ula<<<M,N>>>(ay_kaynak, ay_sonuc, gen, yuk, boy);


son<<<M,N>>>(ay_kaynak, ay_sonuc, gen, yuk, boy);
        isaretle<<<azamiFindikSayisi,azamiFindikSayisi>>>(ay_sonuc, gen * yuk, ay_yerler);
#pragma endregion baslangic_etiketlemesi

#pragma region komumlari_duzelt

        cudaMemcpy(yerler, ay_yerler, sizeof(int) * 400, cudaMemcpyDeviceToHost);

        int bolgeSayisi =0;

        for(int i=0; i<azamiFindikSayisi * azamiFindikSayisi; i++)
        {
if(yerler[i]> 0)
                {
                        yeniIndisler[bolgeSayisi]= bolgeSayisi+1;
                        bolgeSayisi++;
                }
        }

        #pragma endregion komumlari_duzelt

        cudaMemcpy(ay_yeniYerler, yeniIndisler, sizeof(int) * 400, cudaMemcpyHostToDevice);
        etiketle<<<M,N>>>(ay_sonuc,boy, ay_yerler,  ay_yeniYerler, bolgeSayisi);
        cudaMemcpy(sonuc, ay_sonuc, sizeof(int) * gen * yuk, cudaMemcpyDeviceToHost);

        // burada her bir bölge tespit edildi
        // bölgedeki noktalara ait en sol ve en sağ değerler
        // diziler içerisinde aktarıldı
        bolgeleriTespitEt<<<M,N>>>(ay_sonuc, gen, yuk, boy, ay_solX, ay_solY, ay_sagX, ay_sagY);

        cudaMemcpy(solX, ay_solX, sizeof(int) * azamiFindikSayisi, cudaMemcpyDeviceToHost);
    cudaMemcpy(solY, ay_solY, sizeof(int) * azamiFindikSayisi, cudaMemcpyDeviceToHost);
    cudaMemcpy(sagX, ay_sagX, sizeof(int) * azamiFindikSayisi, cudaMemcpyDeviceToHost);
    cudaMemcpy(sagY, ay_sagY, sizeof(int) * azamiFindikSayisi, cudaMemcpyDeviceToHost);

        solX[bolgeSayisi]= -1;

        int enBuyukAlan = 0;
        for(int i=0; i<bolgeSayisi; i++)
        {
                int alan = abs( (solX[i] - sagX[i]) *  (solY[i] - sagY[i]));
                if(alan<0)
                        alan *= -1;

                if(alan > enBuyukAlan)
                        enBuyukAlan = alan;
        }

#pragma region iadeIslemleri

        free(yeniIndisler);
        free(elemanSayilari);
        free(yerler);

        cudaFree(ay_yeniYerler);
        cudaFree(ay_kaynak);
```

Ek1'in Devamı

```
        cudaFree(ay_sonuc);
        cudaFree(ay_yerler);
        cudaFree(ay_solX);


        cudaFree(ay_solY);
        cudaFree(ay_sagX);
        cudaFree(ay_sagY);



#pragma endregion iadeIslemleri
}
```

ÖZGEÇMİŞ

1985 yılında Antalya'da doğdu. İlkokulu Nimet Güvener İlköğretim Okulu'nda orta okul ve liseyi Burdur Anadolu Lisesi'nde okudu. 2003 yılında girdiği Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü'nden 2008 yılında mezun oldu. 2009 yılında Karadeniz Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda yüksek lisans programına başladı. Halen aynı üniversitenin bilgi işlem dairesinde bilgisayar mühendisi olarak çalışmaktadır. Yabancı dil olarak İngilizce bilmektedir.