
GAZİ ÜNİVERSİTESİ
Gazi
1926

İRİS KARŞILAŞTIRMA ALGORİTMASININ OPENCL DİLİ KULLANILARAK HIZLANDIRILMASI

Buğra ŞİMŞEK

YÜKSEK LİSANS TEZİ
ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KASIM 2016

Buğra ŞİMŞEK tarafından hazırlanan "İRİS KARŞILAŞTIRMA ALGORİTMASININ OPENCL DİLİ KULLANILARAK HIZLANDIRILMASI" adlı tez çalışması aşağıdaki jüri tarafından OY BİRLİĞİ ile Gazi Üniversitesi Elektrik Elektronik Mühendisliği Anabilim Dalında YÜKSEK LİSANS TEZİ olarak kabul edilmiştir.

Danışman: Doç. Dr. Nursel AKÇAM
Elektrik Elektronik Mühendisliği, Gazi Üniversitesi
Bu tezin, kapsam ve kalite olarak Yüksek Lisans Tezi olduğunu onaylıyorum ……

Başkan: Prof. Dr. Erkan AFACAN
Elektrik Elektronik Mühendisliği, Gazi Üniversitesi
Bu tezin, kapsam ve kalite olarak Yüksek Lisans Tezi olduğunu onaylıyorum ……...

Üye: Yrd. Doç. Dr. Selma ÖZAYDIN
Elektronik ve Haberleşme Mühendisliği, Çankaya Üniversitesi
Bu tezin, kapsam ve kalite olarak Yüksek Lisans Tezi olduğunu onaylıyorum ……...

Tez Savunma Tarihi: 09/11/2016

Jüri tarafından kabul edilen bu tezin Yüksek Lisans Tezi olması için gerekli şartları yerine getirdiğini onaylıyorum.

……

Prof. Dr. Hadi GÖKÇEN
Fen Bilimleri Enstitüsü Müdürü

ETİK BEYAN

Gazi Üniversitesi Fen Bilimleri Enstitüsü Tez Yazım Kurallarına uygun olarak hazırladığım bu tez çalışmasında;

- Tez içinde sunduğum verileri, bilgileri ve dokümanları akademik ve etik kurallar çerçevesinde elde ettiğimi,
- Tüm bilgi, belge, değerlendirme ve sonuçları bilimsel etik ve ahlak kurallarına uygun olarak sunduğumu,
- Tez çalışmasında yararlandığım eserlerin tümüne uygun atıfta bulunarak kaynak gösterdiğimi,
- Kullanılan verilerde herhangi bir değişiklik yapmadığımı,
- Bu tezde sunduğum çalışmanın özgün olduğunu,

bildirir, aksi bir durumda aleyhime doğabilecek tüm hak kayıplarını kabullendiğimi beyan ederim.

Buğra ŞİMŞEK

09.11.2016

İRİS KARŞILAŞTIRMA ALGORİTMASININ OPENCL DİLİ KULLANILARAK HIZLANDIRILMASI

(Yüksek Lisans Tezi)

Buğra ŞİMŞEK

GAZİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

Kasım 2016

ÖZET

Son yıllarda biyometrik sistemler pek çok farklı alanda yaygın olarak kullanılmaya başlanmıştır. Bu sistemler kişilerin tespitini ve kimlik doğrulamasını parmak izi, retina, yüz şekli, el damarları gibi eşsiz özellikler kullanarak sağlarlar. İris bazlı biyometrik sistemler kullanımı yaygın olan ve diğer biyometrik sistemlerle karşılaştırıldığında yanlış eşleştirme oranı düşük olan sistemlerdir. İris tanıma sistemleri, insan gözüne zarar vermedikleri ve kişinin ölümünden sonra iris yapısı hızlı bir şekilde değiştiği için gerçek dünya uygulamalarında yaygın olarak kullanılmaktadırlar. Bu tez çalışmasında, gelişen donanım ve yazılım teknolojilerinden faydalanılarak iris karşılaştırma algoritması hızlandırılmış ve heterojen sistemlerde kullanılabilir hale getirilmiştir. İris karşılaştırma algoritması eş zamanlı işlem ve çok çekirdekli donanım altyapısı kullanılarak paralelleştirilmiştir. Uygulama aşamasında heterojen, mobil ve paralel hesaplama altyapısı sağlayan C temelli Açık Hesaplama Dili (Open Computing Language-OpenCL) kullanılmıştır. OpenCL dilinin sağladığı tüm avantajlar kullanılarak Hamming Mesafesi algoritması paralelleştirilmiş ve heterojen sistemlere uyumluluğu sağlanmıştır.

Bilim Kodu	:	90508
Anahtar Kelimeler	:	Paralel hesaplama, heterojen hesaplama, iris karşılaştırma, OpenCL, paralel programlama, heterojen programlama
Sayfa Adedi	:	66
Danışman	:	Doç. Dr. Nursel AKÇAM

ACCELERATING IRIS COMPARISON ALGORITHM WITH OPENCL

(M. Sc. Thesis)

Buğra ŞİMŞEK

GAZİ UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

November 2016

ABSTRACT

In recent years, the biometric systems have acquired a widespread usage in many different areas. These systems ensure the identification and recognition of persons by using unique features such as fingerprint, retina, iris, face shape, and hand vessels. Iris based biometric systems are frequently used systems and have lower false matching rates compared to other biometric systems. Since iris recognition systems are not harmful to human eye and since the structure of human iris changes quickly after the death of the humans, iris recognition systems are used largely in real world applications. In this study, iris comparison algorithm is accelerated by using new hardware and software technologies and it is ensured that it is compatible with the heterogeneous systems. Iris comparison algorithm is parallelized via multi core hardware substructure with synchronous threads. For the implementation, C based OpenCL language is used which maintains a heterogeneous, mobile and parallel computing substructure. Hamming distance algorithm is parallelized and it is ensured that it is compatible with heterogeneous systems by using all the benefits of OpenCL.

Science Code : 90508

Key Words : Parallel computing, heterogeneous computing, iris matching, OpenCL, parallel programming, heterogeneous programming

Page Number : 66

Supervisor : Assoc. Prof. Dr. Nursel AKÇAM

TEŞEKKÜR

Öncelikle hayatın her basamağında desteğini her fırsatta gösteren aileme, tez çalışmasını yaptığım dönemde çalışmalarım konusunda beni motive eden eşim Esra ŞİMŞEK'e teşekkürlerimi sunmayı bir borç bilirim.

Gerek lisans, gerekse yüksek lisans eğitimimde her konuda kendisinden destek aldığım danışman hocam Sn. Doç. Dr. Nursel AKÇAM' a saygı ve teşekkürlerimi sunuyorum.

Yüksek lisans eğitimime olan desteklerinden dolayı işverenim ASELSAN A.Ş. ye sonsuz teşekkürlerimi sunuyorum.

"Yurtiçi Yüksek Lisans Burs Programı" ile beni destekleyen TÜBİTAK A.Ş. ye bilimsel araştırmalarımızı teşvikinden ötürü teşekkür ederim.

İÇİNDEKİLER

Sayfa

Sayfa

ÇİZELGELERİN LİSTESİ

Çizelge **Sayfa**

ŞEKİLLERİN LİSTESİ

Şekil **Sayfa**

RESİMLERİN LİSTESİ

Resim **Sayfa**

Resim **Sayfa**

SİMGELER VE KISALTMALAR

Bu çalışmada kullanılmış simgeler ve kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

Kısaltmalar	Açıklamalar
CCD	Charge-coupled device (yükle birleştirilmiş cihaz)
CPU	Central processing unit (merkezi işlem birimi)
CUDA	Compute unified device architecture
DSP	Digital signal processor (sayısal sinyal işlemcisi)
FPGA	Field programmable gate array
GPU	Graphics processing unit (grafik işleme ünitesi)
HM	Hamming Mesafesi
OPENCL	Open computing language (açık hesaplama dili)
PCIe	Peripheral component interconnect express
RAM	Random access memory (rastgele erişim belleği)
SN	Saniye

1. GİRİŞ

İris göz bebeğini çevrelemiş olan renkli tabakaya verilen isimdir. Gözün iris bölgesi kendine özgü özellikler taşır. Şekilsel yapısı incelendiğinde irisin karmaşık bir görüntüsel model oluşturduğu ve bu modelin kişiden kişiye büyük dereceli farklılıklar gösterdiği bilinir [1]. Bu yapısal farklılıkların bilinmesi yepyeni bir alanın doğmasına öncülük etmiştir. İnsan irisinden kişinin tespit edilmesi fikri ortaya çıkmış ve bu alanda bazı çalışmalar yapılmaya başlanmıştır. Otomatikleştirilmiş ilk iris tanıma sistemi Aran Safir ve Leonard Flom tarafından 1987 yılında geliştirilmiştir. Bu geliştirilmiş olan sisteme "İris Tanıma Teknolojisi" adı verilerek patenti alınmıştır [2]. Cambridge Üniversitesi araştırmacılarından John Daugman, Aran Safir ve Leonard Flom'un patentini aldıkları "İris Tanıma Teknolojisi"nde belirtilen sistem esaslarına uygun bir algoritma tasarımı geliştirmiştir. Bu tasarımı bir patentle [3] belgeleyen Daugman iris tanıma konusunda öncü isimlerden biri olmuştur. Patentin süresinin dolması akabinde bu alanda birçok çalışma meydana getirilmiştir. İris tanımadaki doğruluk oranının artırılması, performansın geliştirilmesi, iris tanıma sisteminde güç tüketiminin azaltılması üzerine birçok çalışmalar yapılmıştır. Son yıllarda ise farklı donanımlar kullanılarak performans artışı sağlamaya yönelik çalışmalar yapılmaya başlanmıştır. İris tanıma iris resminden ayırt edici verilerin elde edilmesi işlemi ve bu elde edilen verilerin bir veritabanı üzerindeki verilerle karşılaştırılarak eşleştirme sağlanması olmak üzere iki bölümden oluşturur. Bu teze konu olan çalışma irislerin yalnızca hızlı bir şekilde karşılaştırılarak, karşılaştırma algoritmasına performans artışı sağlamak ile kalmayıp aynı zamanda bu karşılaştırmayı C temelli OpenCL (Open Computing Language) dilinde yaparak, OpenCL dilini destekleyen tüm cihazlarda (Merkezi işlem birimleri, grafik işlem birimleri, alanda programlanabilir kapı dizilimleri ve sayısal sinyal işlemci kartları gibi) ortak olarak çalışabilme altyapısı da sağlamaktadır. Literatürde iris karşılaştırma algoritmasının farklı cihazlarda yapılan uygulamaları mevcuttur ancak şimdiye kadar yapılan çalışmalarda heterojen bir sistemde ilgili algoritma çalıştırılmamıştır. Bu bizim çalışmamızı diğer çalışmalardan farklı kılan özelliklerden bir tanesidir. Yani CPU (Central Processing Unit), GPU (Graphical Processing Unit), FPGA (Field Programmable Gate Array) ve DSP (Digital Signal Processor) kartlarından oluşan bir sistemde kod istenen cihaz üzerinde koşturulabilecek bir imkâna kavuşturulmuştur. Paralelleştirilen kod ile performans artışı sağlanmış ve algoritmanın gerçeklenmesi hızlandırılmıştır.

Bu çalışmada ilk bölüm olan iris resminden özelliklerin çıkarımı kısmında Labor Masek' in geliştirmiş olduğu ve kaynak kodlarını akademik dünyaya ücretsiz olarak sunduğu iris tanıma sistemi kullanılmıştır [4]. Bu açık kaynak kodu, tek bir iris resminden irisi ifade eden ve şablon olarak ifade edilen bit matrisini ayrıca resimde gürültü olarak algılanan bölgeleri ifade eden bir maske bit matrisi oluşturmaktadır. Çalışmamızda yazılmış olan bu kod bir C# uygulaması ile otomatik hale getirilmiş ve tüm iris veritabanı bu C# uygulaması ile otomatik olarak iris şablonu ve maskesi üretecek şekilde geliştirilmiştir. Elde edilen bu şablon ve maske verileri yeni bir veri tabanında tutulmuştur. Böylece karşılaştırma yapılacak önceden tanımlı kişilerin irislerinin verisinin olduğu bir veritabanı elde edilmiştir. Öte yandan bir de bilinmeyen bir kişiye ait irisin sistemdeki diğer irislerle karşılaştırılması amacıyla tek kişilik bir iris ve maske verisi elde etmek için aynı kod kullanılmıştır. Bu aşamayı takiben, yazılmış olan bir C++ uygulaması ile veritabanında bulunan irislerle ile bu bilinmeyen iris karşılaştırılarak performans değerleri yine yazılım tarafından belirlenmiştir. Yazılmış olan bu C++ uygulaması seri (ya da sıralı) olarak işlem yapmaktadır. Bu uygulama herhangi bir paralelleştirme olmadan algoritmanın performansının belirlenmesinde kullanılmıştır. Akabinde paralelleştirilmenin yapılması için tekrar bir C++ temelli OpenCL uygulaması yazılarak performanstaki artış ve farklı donanımlar üzerinde çalıştırma denenmiştir. Paralelleştirme amaçlı olarak yazılan bu heterojen hesaplama uygulaması sağlayıcı kodu CPU üzerinde koşarak, OpenCL dilinde yazılmış olan cihaz kodlarının hedef cihazlar üzerinde çalıştırılmasının organizasyonunu sağlayan, veritabanından iris verilerinin alınıp düzenlenerek hedef cihazlara yönlendirilmesini sağlayan koddur. Yapılan bu çalışma son dönem teknolojilerinden yararlanması ve bu yeni donanım ve yazılım altyapısını kullanarak iris karşılaştırmada ve bit matrislerinin genel olarak karşılaştırılmasında da kullanılan "Hamming Mesafesi" algoritmasının bu teknolojiye uyarlanması ve tüm destek veren cihazlarda kullanılabilmesi açısından önemlidir. OpenCL henüz 2008 yılında ortaya çıkmış ve uluslararası bir konsorsiyum olan Khronos Grup tarafından ortaya konulmuştur. İlerleyen sayfalarda bu konuda daha detaylı bilgi verilecektir. Tezimizin konusu karşılaştırma olduğu için bu özellik çıkarımı aşamasında gerçek zamanlı bir sistem kullanılmamıştır. Ayrıca bit verilerinin elde edileceği iris resimlerinden oluşan bir veritabanı kullanılması amacıyla bu konuda Portekiz'deki Beria Üniversitesinde çalışma yapan SOCIA Laboratuarı ile iletişime geçilmiş, oluşturdukları bir sisteme kaydolunarak iris verileri elde edilmiştir [5].

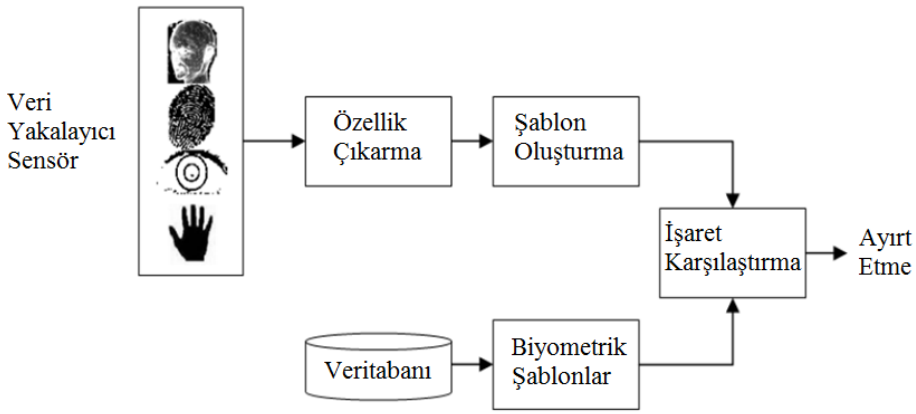
2. BİYOMETRİK SİSTEMLER

Günümüzde güvenliğe verilen önemin artmasıyla güvenlik sistemlerinde bazı değişiklikler meydana gelmiştir. Yaygın olarak kullanılmakta olan şifreli ya da kartlı güvenlik sistemleri yavaş yavaş yerini biyometrik sistemlere bırakmaktadır. Şifreli ve kartlı sistemlerde, şifrenin çalınması, kartın kaybolması gibi nedenlerle güvenliğin sağlanamaması gibi durumlar söz konusu olabilmektedir. Kimi zaman kullanıcılar tarafından kendi şifreleri unutulabilmekte ve bu gibi durumlarda da mağduriyetler yaşanabilmektedir. Bu nedenle güvenlik sistemleri biyometrik olarak insanların ayırt edici özelliklerinden faydalanma noktasında bir gelişme göstermiştir Bu gelişme insanın unutamayacağı ya da çaldıramayacağı kendi parçası olan bir anahtar güvenliğinin sağlanmasının önünü açmıştır. Biyometri fiziksel ve davranışsal biyometri olmak üzere iki ana kategoriye ayrılır. Fiziksel biyometri insanların parmak izi, yüz yapısı, DNA, avuç içi izi, el geometrisi, retina, iris yapısı ve ses gibi ayırt edici fiziki özelliklerini konu edinir [6]. Davranışsal biyometri ise daha çok ses, kişinin yürüyüşü ve yazı yazma ritmi gibi bazı psikolojik ayırt edicileri konu edinir. Şüphesiz bu biyometrik ayırt edici özellikler rastgele bir şekilde bulunmamıştır. Belirli ortak özellikleri sağlayan organ ve yapılar biyometrik ayırt edici olarak kullanılabilmektedir. Bu özellikler genel olarak yedi maddede toplanabilir [7]:

- Genellik: Her insanın sahip olduğu bir organ, yapı ya da özellik olmalıdır.
- Teklik: İnsandan insana değişen bir özellik olmalıdır.
- Kalıcılık: Zamanla değişmeyen, yaşlanma gibi etkilerden etkilenmeyen bir özellik olmalıdır.
- Elde Edilebilirlik: Ölçülebilen bir özellik olmalıdır.
- Performans: Performans açısından değerlendirilebilir olmalıdır. Başka örnekleriyle karşılaştırılmasıyla hız, güvenilirlik ve doğruluk gibi kriterlerde sonuçlar elde edilebilmelidir.
- Kabul Edilebilirlik: İnsanların kabul edebileceği, insanlara zarar vermeden ölçülebilen bir özellik olmalıdır.
- Önleme: Kolay taklit edilemeyen bir özellik olmalıdır.

Bu özellikleri sağlayan organ, yapı ya da diğer karakteristikler biyometrik ayırt edici olarak kullanılabilir. Biyometrik bir sistemin blok şeması Şekil 2.1 de gösterilmiştir.

Şekilde ilk blok veriyi alan, tüm biyometrik sistemi dış dünya ile bağlayan sensördür.



Şekil 2.1. Biyometrik sistem blok şeması

Sensörden veri alındıktan sonra öncelikle gürültüyü silmeye yönelik bazı ön işlemler gerçekleştirilir. Daha sonra normalleştirme ve özellik çıkarımı aşamalarından geçilerek bir şablon oluşturulmuş olur. Tüm geçerli şablonlar bir veri tabanında tutulur. Bilinmeyen ve kime ait olduğu belirlenecek olan veri de aynı şekilde sensörden alınır ve özellik çıkarımı yapılır. Daha sonra ise bir diğer faz olan karşılaştırma fazına geçilmiş olur. Karşılaştırma aşamasında ise kullanılan algoritmaya göre iki şablonun birbirine olan uzaklıkları bulunur. Bu uzaklıklardan en düşük olanına göre veyahut belirlenmiş bir değerin altında kalan değere göre eşleştirme yapılır. İki şablon arasındaki uzaklık ne kadar az ise bu iki şablon birbirine o kadar benzerdir denilerek eşleştirme gerçekleştirilir. En çok kullanılan biyometrik ayırt ediciler aşağıdaki gibidir:

DNA

DNA deoksiribo Nükleik Asit in kısaltılmış adıdır. DNA iki sarmaldan oluşan ve üzerinde birçok farklı özellik barındıran bir biyometrik ayırt edicidir. DNA nın biyometrik ayırt edici olarak kullanılmasında bazı sorunların olduğu da bilinir. Bunlardan en önemlisi ikiz bireylerin aynı DNA ya sahip olmalarıdır. Aynı DNA ya sahip bireylerde güvenlik ve adli vakalarda oldukça ciddi eşleştirme problemleri gözlenmektedir [8]. Ayrıca DNA nın yapısının hassas olması ve çabuk bozulabilmesi ve ayrıca günlük gerçek zamanlı uygulamalarda kolayca test edilememesi DNA nın kullanımındaki bazı dezavantajlardır.

Yüz

Yüz tanıma sistemleri, oldukça yaygın kullanılan bir diğer biyometrik sistemlerdir. Bu sistemlerde üç farklı özellik çıkarım metodu kullanılmaktadır. Jenerik metotta yüz resmindeki çizgiler, kıvrımlar ve kenarlar yakalanmaya çalışılır. Şablon temelli olan metotta ise yüz üzerindeki organlardan hareketle yüzün tanınması gerçekleştirilir. Örneğin burun ve gözler hesaplanarak yüzün yapısal özellikleri çıkarılır. Yapısal özellik çıkarım metodunda ise yüzün yapısal özellikleri ve eliptik yapısı yakalanmaya çalışılır [9]. Bu sistemlerde yaşanan en büyük sorunlar yüz resminde yüzün açısının uygun olmamasından kaynaklanan sorunlardır. Resim 2.1'de bir maratonda koşucuların yüz yakalama sistemi ile görüntülendiği sistemin ekran çıktısı gösterilmektedir.

Resim 2.1. Yüz tespit sistemi ekran çıktısı

Yürüme şekli

Yürüme şekli çok yaygın olmasa da kullanılan bir çeşit biyometrik ayırt edicidir. 90 lı yılların başında yürümenin kişiden kişiye farklılık gösterdiği bulunmuştur [10]. Bacağa takılan cihazlar yardımıyla uzun ve kısa adımların modellemesi yapılır. Her adımda bacağın nasıl bir çevrim yaptığı kaydedilir. Ayrıca bacakların birbiri ile ve tüm omurga ekseni ile yaptığı açılar yürüyüş tanımada kullanılır. Daha sonra video üzerinden yürüyüş şekli modellenen kişinin tanınması gerçekleştirilmeye çalışılır. Resim 2.2'de yürüme halinde olan kişiler üzerinde yürüyüş tanıma sisteminin uygulaması gösterilmektedir.

Resim 2.2. Yürüyüş tanıma sistemi

Parmak izi

Parmak izi yüzyıllardır kullanılan en bilindik biyometrik ayırt edicidir. Parmak izini oluşturan izler bir okuyucu sensör kullanılarak kolaylıkla elde edilir. Bir kerede tüm parmakların izlerinin alınmasını sağlayan sensörler verinin alınmasını oldukça kolaylaştırır (Resim 2.3). Bunun ardından parmak izi çizgilerinden ve parmak bölgesindeki yükseltilerden faydalanılarak kişiye özgü özellikler çıkarılır.

Resim 2.3. Tüm parmakların sensör yardımı ile okunması

Parmak izi günümüzde tabletler, cep telefonları, güvenlik sistemleri ve hatta yüksek güvenlikli sistemlere sahip ATM'lerde kullanılmaktadır (Resim 2.4). Bu sistemlerin tek

dezavantajı parmağın vücut ile bağlantısı koptuğunda ya da kişi öldüğünde hala sistem tarafından parmağın kabul edilmesidir.

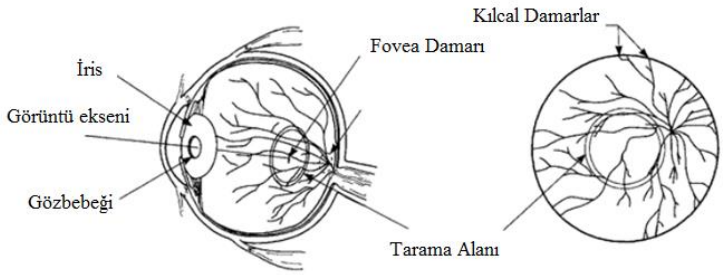
Resim 2.4. ATM parmak izi tanıma sistemi

Tuş darbesi

Tuş darbesinden kişinin tanınması ilgi çeken tanıma sistemlerindendir. Bu tanıma sisteminde tuşa basma süresi ve tuştan tuşa geçiş süresi klavye ya da tuş takımı kullanılarak elde edilir. Bu süreler kişiden kişiye değişir. Belli bir süre kişinin tuş takımı kullanarak bir şeyler yazması istenir ve kişiye özgü bir şablon oluşturulur. Daha sonra bu şablonlar kullanılarak bilinmeyen kişinin veritabanında olup olmadığı ya da kim olduğu bulunabilir [8].

Retina taraması

Gözün retina bölgesini beslemekte olan kan damarları eşsiz bir yapıdadır. Retina taraması retina bölgesini besleyen bu kan damarlarının modellenmesinden faydalanılarak kişinin tanınmasını sağlar [11]. Şekil 2.2'de retinayı besleyen damar yapısı gösterilmektedir.



Şekil 2.2. Retinayı besleyen damarlar

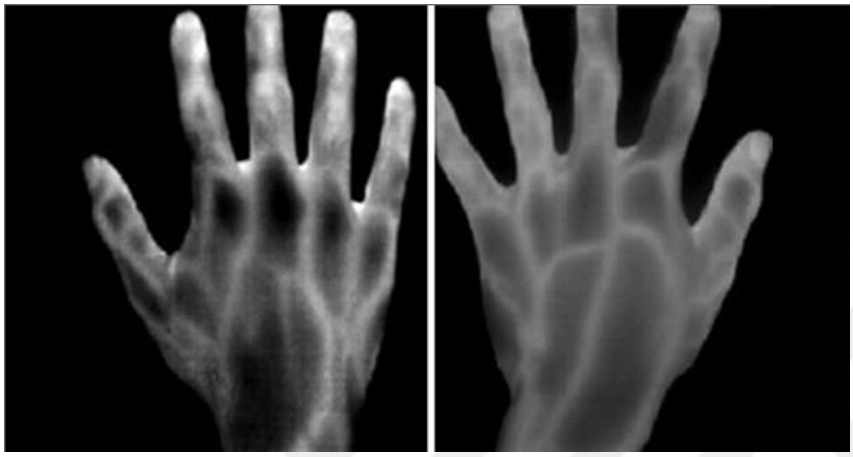
Retina taramasında kişinin gözü düşük enerjili kızıl ötesi ışınlar ile taranır. Buradan elde edilen veriler, şablonlar halinde veri tabanında tutulur. Daha sonra kimliği tespit edilmek istenen kişinin retina şablonu elde edilerek veri tabanındaki şablonlar ile karşılaştırma yapılır. Retina taraması kullanılan tanıma sistemlerinde kesin bir doğrulukla %100 doğru eşleştirme yapılmaktadır. Fakat bu sistemde kullanılan kızılötesi ışınlar kişilerin göz sağlığını olumsuz etkiler. Bu ışınlar katarakt ve astigmat gibi hastalıklara sebep olur.

Damar tanıma

Damar tanıma sistemleri daha yaygın olarak avuç içi damarlarından kişilerin tespit edilmesinde kullanılır. Kanda yer alan hemoglobin pigmentleri kızıl ötesi ışınları yansıtma özelliğine sahiptir. Resim 2.5'de gösterildiği gibi tasarlanmış bir cihaz ile kişinin eline kızıl ötesi ışınlar gönderilir.

Resim 2.5. El damarı tarama sistemi

Şarj Çifti Cihazı (CCD) temelli bir kamera ile hemoglobinden yansıyan kızıl ötesi ışınlar görüntülenir. Böylece kişinin damar haritası çıkarılmış olur. Resim 2.6'da bir bireye ait el damar haritası gösterilmiştir. Daha sonra bu damar bölgeleri, işlenerek dijital veri haline getirilir. Tüm diğer biyometrik sistemlerde olduğu gibi bilinmeyen kişiye ait veriler ile veri tabanındaki veriler karşılaştırılır ve kişinin tespit edilmesi sağlanır.



Resim 2.6. El damar haritası

Ses tanıma

Ses tanıma sistemlerinde, kişilerin sesindeki eşsiz özellikler elde edilerek bu özellikler üzerinden sesin tanınması sağlanır. Burada söylenilen sözün anlamına göre sisteme farklı kararların verdirilmesi sağlanabilir. Ses tanıma sistemleri "konuşmacı bağımlı" ve "konuşmacı bağımsız" sistemler olarak ikiye ayrılır. Eğer her konuşmacıya belirli bir metin okutularak bir eğitim verisi elde ediliyorsa "konuşmacı bağımlı", eğitim verisi alınmıyorsa "konuşmacı bağımsız" sistem olarak tanımlama yapılır.

Konuşmacı tanıma sistemleri ise; alınmış olan ses verileri üzerinden sesin hangi bireye ait olduğunun bulunduğu sistemlerdir. Bu sistemler "konuşmacı bağımlı" sistemler olup kişilerin daha önceden veri tabanında ses verilerinin bulunması gerekmektedir.

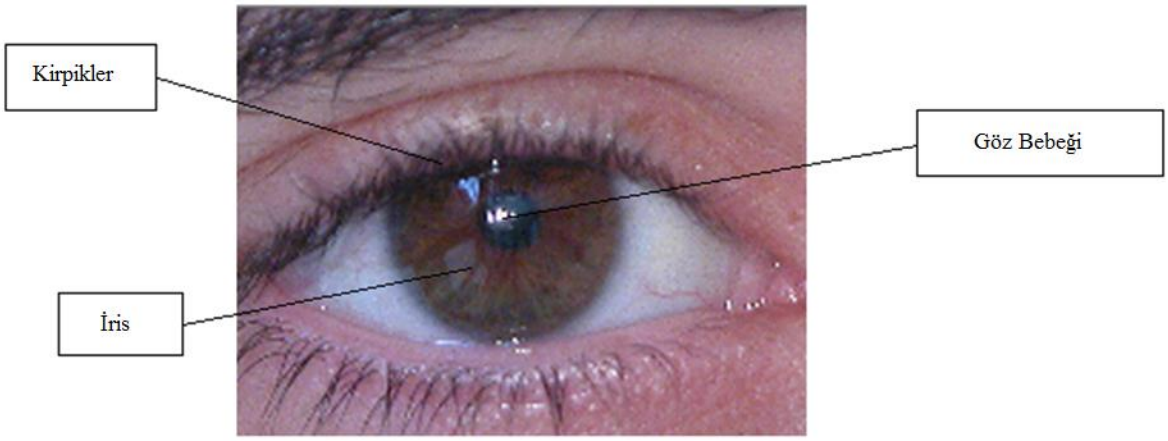
İris

İris yapısında yay, uzun şerit, halka, taç ve zikzak gibi karmaşık şekilleri bulundurur. İçinde yer alan bu şekiller sayesinde iris kişiden kişiye değişen eşsiz bir yapı kazanır. Öyle ki aynı yumurta ikizlerinde dahi irisler birbirinden farklılık gösterir. Retinadan farklı olarak iris resminin çekilmesi göze herhangi bir zarar vermez. Bu nedenle kişilerin çekinmeden kullanabilecekleri bir biyometrik sistemdir. İristen kişi tanıma sistemleri iris resminin kalitesinin yüksek olduğu uygulamalarda oldukça yüksek başarı oranına sahiptir. Resimde gürültülerin olması, gözün kısık çıkması, parlamaların meydana gelmesi veya resmin uzaktan çekilmesi gibi durumlar bu başarıyı olumsuz yönde etkiler. Fakat yine de iris yapısının karmaşıklığı sebebiyle resmin kalitesinin düşük olması çoğu zaman yanlış eşleştirme yapılmasına neden olmaz. İris tanıma sisteminin kullanılması için gri tonda resim kaydedebilen bir kameranın olması yeterlidir. Resim 2.7'de iris tanıma sistemi genel hatlarıyla gösterilmiştir. Hem maliyetinin düşük olması, hem de yüksek başarı oranı ile eşleştirme yapması sebebiyle iris tanıma en popüler biyometrik sistemlerden biridir.

Resim 2.7. İris kamerası

3. İRİS TANIMA SİSTEMİ

İris göze rengini veren, ince ve yuvarlak yapıdaki, göze gelen ışığa göre göz bebeğinin boyutunu belirlemekle görevli olan göz kısmıdır. Bu kısım göze gelen ışık yoğunluğu fazlaysa göz bebeğini daraltır. Eğer ışık miktarı azsa ya da ortam tamamen karanlıksa iris kasları göz bebeğini büyütür. Biyolojik açıdan gözün morfolojisi Resim 3.1'de gösterilmiştir.



Resim 3.1. Gözün morfolojisi

İris tanıma sistemleri diğer sistemlerle karşılaştırıldığında birçok avantaja sahiptir. Bu avantajlar iris tanımanın diğer biyometrik tanıma sistemlerinden daha çok kullanılmasını sağlamaktadır. İrisin insan vücudunda içsel bir organ olması, kornea tarafından korunması, hasarlara karşı dayanıklı bir yapısının olması, embriyonik dönemde oluşup kişinin hayatını kaybetmesiyle de saniyeler içinde yapısının bozulması, iris kamerasının retina tarayıcılar gibi gözün yapısına zarar vermemesi, herhangi bir şekilde irise fiziksel olarak dokunmadan iris tanımanın yapılabilmesi gibi özellikler iris tanımanın tercih edilme nedenlerindendir. Ek olarak iris tanımada kullanılan kameralar diğer biyometrik sistemlerle karşılaştırıldığında maliyet açısından da avantajlıdır. İris tanıma sistemleri ülkemizde çok yaygın olarak kullanılmamakla birlikte dünyada oldukça yaygın bir kullanım alanına sahiptir. Özellikle Amerika Birleşik Devletleri (ABD) ve Birleşik Arap Emirlikleri'nde sınır kontrol birimlerinde, yine ABD de mobil polis birimlerince ve havaalanlarında vize işlemlerinde, Amerikan ordusunun Irak ve Afganistan'daki faaliyetlerinde yoğun olarak iris tanıma sistemlerinden faydalanılmıştır. Bunlara ek olarak İngiltere'de Londra Havaalanında pasaport kontrolleri eGATE olarak bilinen biyometrik güvenlik sistemleri

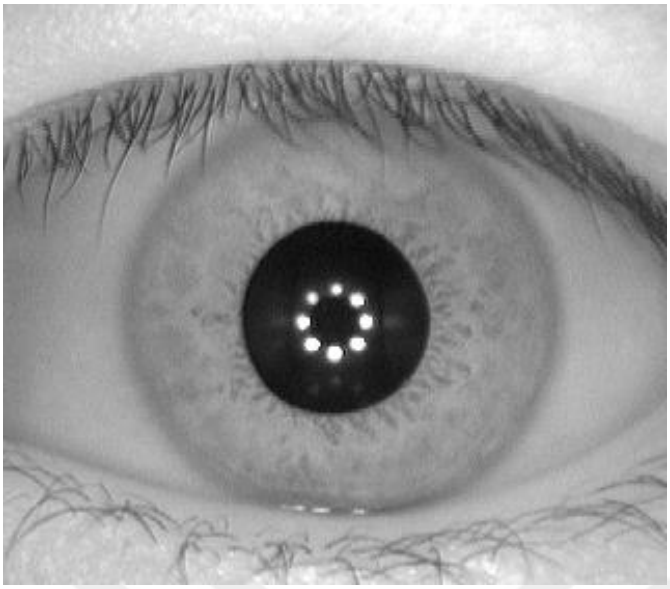
aracılığıyla yapılmaktadır. Kişilerin pasaport verileri ile biyometrik verileri kısa sürede karşılaştırılıp güvenli ve hızlı bir şekilde pasaportun kişiye ait olup olmadığı ya da kişinin İngiltere' ye giriş çıkışında herhangi bir güvenlik engeli olup olmadığı tespit edilmektedir. İngiltere de kullanılmakta olan eGATE sistemi Resim 3.2'de gösterilmiştir.

Resim 3.2. eGATE biyometrik pasaport kontrolü

İris tanıma iki farklı bölümde incelenebilmektedir. Bunlar iris özelliklerinin çıkarılması aşaması ve özellikleri çıkarılmış olan iris şablonlarının karşılaştırılmasıdır.

3.1. İris Özelliklerinin Çıkarılması

İris resminin işlenerek karşılaştırma aşamasında kullanılacak olan özelliklerin iris resminden çıkarılması aşamasını ifade eder. İris özelliklerinin iris resminden elde edilmesi öncesinde iris resminin bazı ön işlemlerden geçirilmesi gerekir. Bu ön işlemler iris bölgesinin belirlenmesi, göz kapağı ve kirpiklerin belirlenmesi, irisin normalleştirilmesidir [12]. Bu aşamalardan geçen iris resmi özellik çıkarımına hazır bir hale getirilmiş olur. Veri tabanında bulunan iris resimleri daha kolay işleme tabi tutulması için gri renkte tercih edilir. Resim 3.3'de veri tabanındaki örnek bir iris resmi gösterilmiştir.



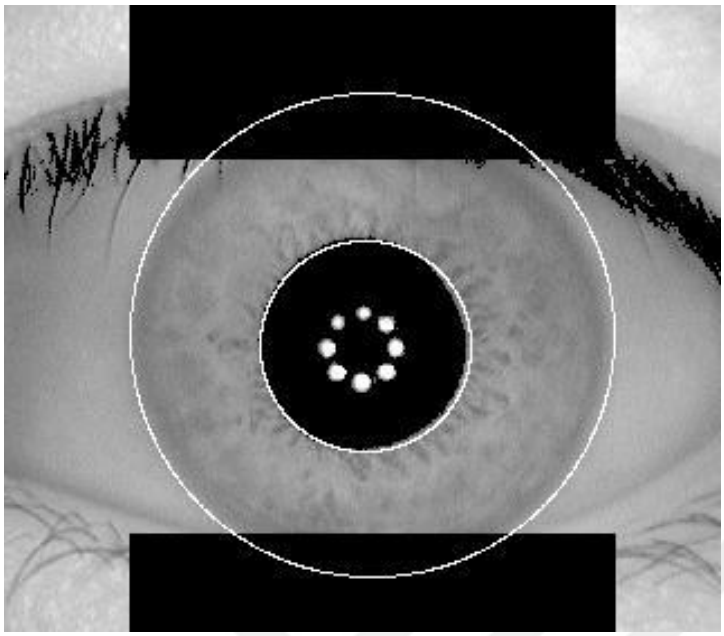
Resim 3.3. Veri tabanındaki iris resmi örneği

İris bölgesinin belirlenmesi

Veri tabanında yer alan resimler bütün olarak gözü içeren resimler oldukları için, gözün tamamından iris bölgesinin izole edilmesi gerekir [12]. İris bölgesinin belirlenmesi aşamasında irisin iç ve dış sınırları tespit edilir. İris bölgesinin tespiti, doğru bir eşleştirme yapması için oldukça önemlidir. Çünkü belirlenen bölge üzerinden diğer bütün işlemler yapılır.

İris bölgesinin tespitinde bizim kullanmış olduğumuz açık kaynak kodu Dairesel Hough Transformasyonunu kullanır. Bu yöntemde irisin iç ve dış kısmı birer çember olarak modellenir [13]. Gri tondaki resme bir kenar algılayıcı uygulanarak tüm kenarlar belirlenir. Kenar belirleme uygulaması türev alınarak ve bir baraj değeri uygulanarak bulunur. Daha sonra çember denklemi kullanılarak gözbebeği merkezi, gözbebeği yarıçapı ve iris yarıçapı bulunur. Bu aşamada Gauss filtresi uygulanarak resim donuklaştırılır. Gauss filtresi resim üzerindeki gürültülerin kaldırılmasını sağlar. Belirlenen kenarlar sınır kabul edilerek tarama işlemi gerçekleştirilir. Göz kapağı ve kirpiklerin resimden çıkarılması için genel uygulama çember olarak modellenmiş olan sınırların parabole çevrilmesidir. Libor Masek ise bu işlem için açık kaynak kodlu iris tanıma sisteminde Doğrusal Hough Transformasyonu kullanmıştır. Doğrusal Hough Transformasyonu ile belirlenen kenar bölgesinin en alt kısmından itibaren resimden gürültü olarak nitelendirilebilecek göz kapağı ve kirpikler gibi bölgeler resimden çıkarılır. Resim 3.4'de

kirpiklerin ortadan kaldırılarak iris bölgesinin belirlendiği örnek resim görülmektedir.

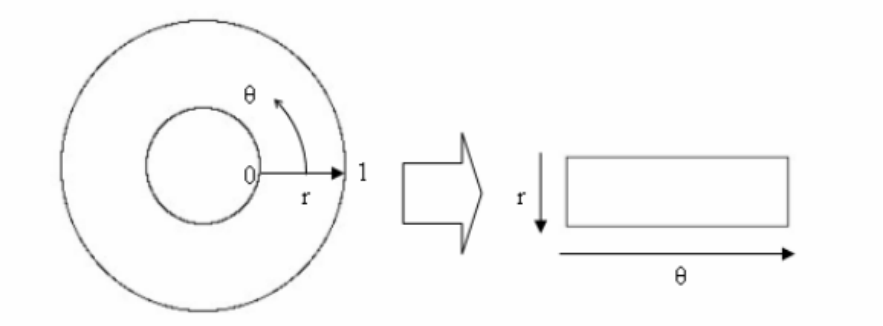


Resim 3.4. İris bölgesinin belirlenmesi

İris normalleştirilmesi

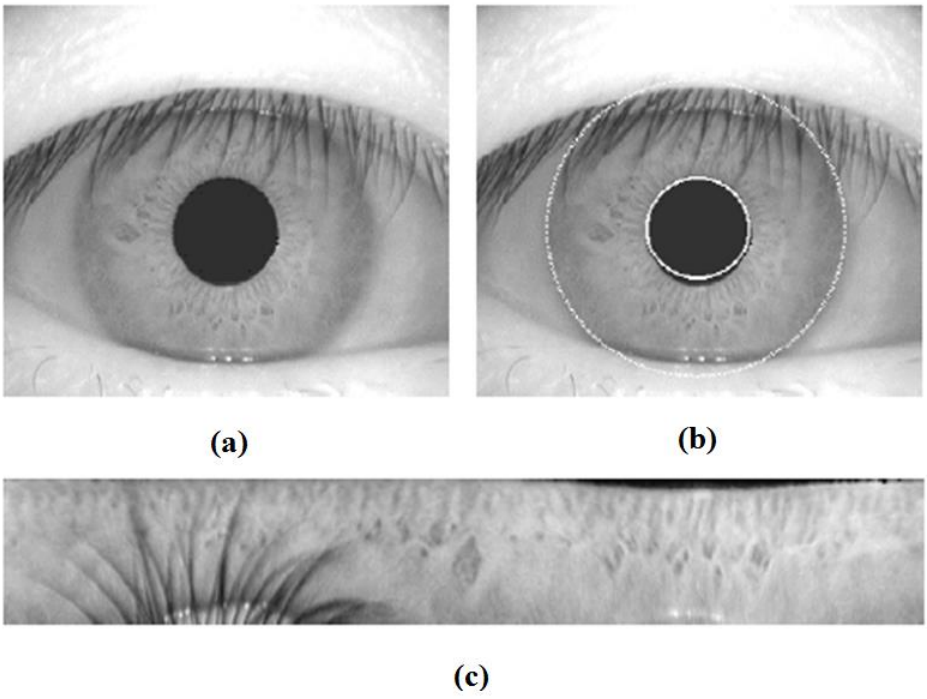
İris normalleştirilmesi aşaması elde edilen iris bölgesinin alınarak daha kolay işlenecek bir hale getirilmesini ifade eder. Normalleştirme işleminden sonra yapılacak işlemler, normalleştirilmiş resim üzerinde daha kolay yapılır. Bizim kullanmış olduğumuz kodda iris normalleştirilmesi işlemi "Rubber Sheet" metodu kullanılarak yapılmıştır. "Rubber Sheet" metodunda irisin her pikseli polar koordinatlardan kartezyen koordinatlara çevrilir [14]. Burada belirlenen yarıçaplarda tarama yapmak mümkündür. Seçilen yarıçapın irisin dış bölgesine taşmayacak bir değer olmasına dikkat etmek gerekir. İrisin dış yarıçapını geçmeme durumu sağlandıktan sonra daha küçük yarıçaplarda da iris tanıma yapılabilir. Örneğin irisin yarı bilgisi alınarak dahi irisin tanıması gerçeklenebilir. Burada önemli olan yeterince veri alınmasıdır. İrisin karşılaştırılmasında kullanılacak yeterli veri alındığı takdirde istenen yarıçapta işlem yapılabilir. Bu çalışmada 2 farklı yarıçap için iris normalleştirilmesi yapılmış ve karşılaştırma aşamasında herhangi bir sorun olmadan irisin ait olduğu kişi bulunmuştur. Rubber Sheet modelinde belirlenen yarıçap ve taranacak olan açı, irisin bit matrisinin boyutlarını belirlemektedir. Polar koordinatlarda bulunan bu açı ve yarıçap değeri, Kartezyen koordinatlarda x ve y değerlerine dönüşür [4]. Bu x ve y değerleri Şekil 3.1'de ayrıntılı şekilde gösterildiği gibi özellik çıkarımı aşamasında bit

matrisinin boyutlarını oluşturur.



Şekil 3.1. Rubber Sheet metodu

Rubber sheet metodu uygulandığı zaman iris sanki bir dikdörtgenmiş gibi tekrardan şekillendirilmiş olur. İris bölgesinin resmin içinden çıkarılıp dikdörtgen formda normalleştirilmiş ifadesi Resim 3.5'de gösterilmiştir [15]. Bu metodun kullanılmasında bir dezavantaj da mevcuttur. Bu metotla normalleştirilmiş resimden elde edilecek olan iris şablon ve maske verileri başlangıç noktası uyumsuzluğu olarak bilinen sapmalardan korumak amaçlı olarak sağa ve sola kaydırılmalıdır. Karşılaştırma aşamasında yazmış olduğumuz kodda bu konuya dikkat edilmiştir.



Resim 3.5. (a) Orijinal iris resmi, (b) Bölgelendirilmiş iris resmi, (c) Normalleştirilmiş iris resmi

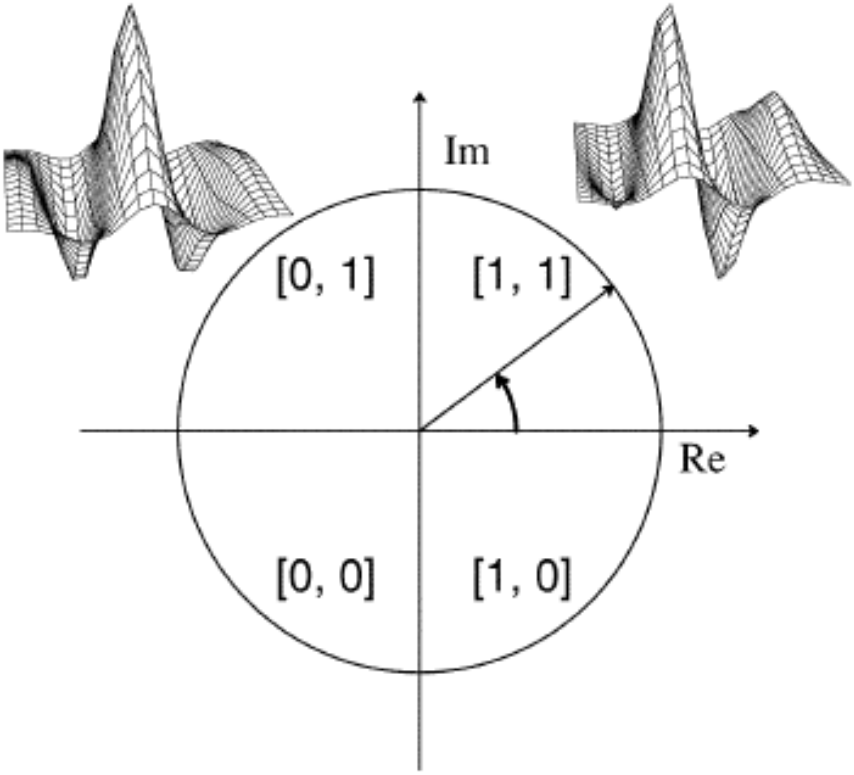
İris normalleştirilmesi aşmasından sonra özellik çıkarımı için gereken ön işlemler bitirilmiş olur. Normalleştirilmiş resim üzerinden özellik çıkarımı yapılır. Ön işlemlerdeki başarı, özellik çıkarımındaki başarıyı etkiler. Özellik çıkarımı normalleştirilmiş ve dikdörtgen hale getirilmiş resmin bit matrislerine çevrilmesi olarak tanımlanabilir. Özellik çıkarımı işlemi gerçekleştirilirken, Libor Masek ' in açık kaynak kodlu "İris Tanıma Sistemi" 1D Gabor Filtrelerinden faydalanılır [4]. John Daugman ise 2D Gabor Dalgacığı metodunu kullanmaktadır [16]. Aslında iki yöntem arasında sadece boyut farkı mevcuttur. Libor Masek algoritması sinyali tek boyutlu sinyale çevirerek 1D Gabor filtreyi kullanmıştır. 1D Gabor Filtre formülü Eş 3.1 de gösterilmiştir.

$$G(f)= e^{-\left(\log\left(\frac{f}{f_0}\right)\right)^2/2(\log(\sigma/f_0))^2} \tag{3.1}$$

Gabor dalgacığı kullanılarak normalleştirilmiş olan iris resmi demodule edilerek faz bilgisi elde edilir. Faz bilgisi elde edilirken Eş. 3.2 den faydalanılır.

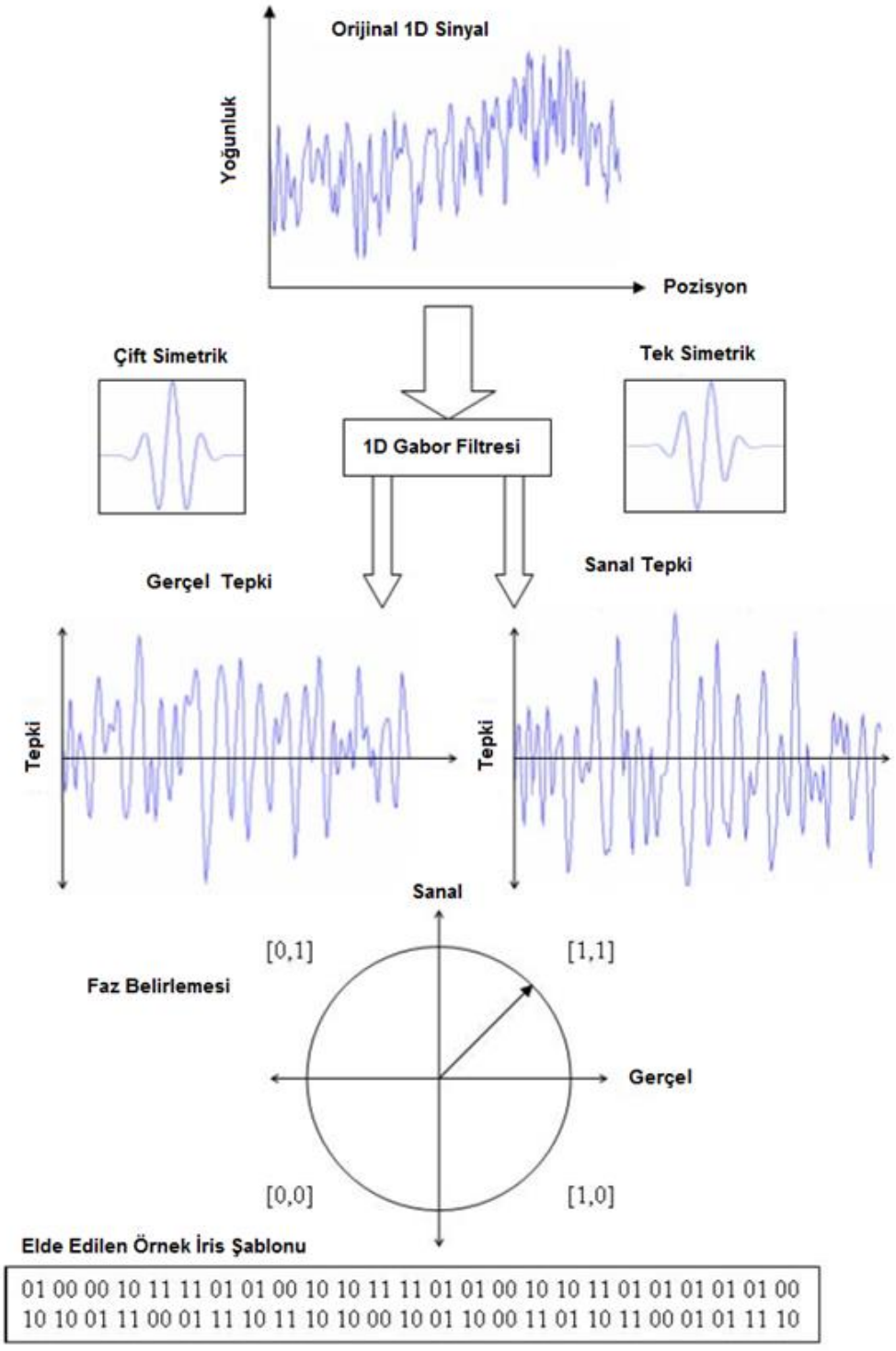
$$h(Re,Im)=sgn(Re,Im) \int_{\rho} \int_{\emptyset} I(\rho,\emptyset)\, e^{-i\omega(\theta 0-\emptyset)}.e^{-\frac{(r0-\rho)^2}{\alpha^2}}. e^{-(\theta 0-\emptyset)^2/\beta^2} \rho\, d\rho\, d\emptyset \tag{3.2}$$

Eş. 3.1 de I(ρ, θ) ifadesi iris resmin polar koordinatlardaki piksel değerini vermektedir. α ve β değerleri dalgacık boyutunu, ω ise dalgacık frekansını ifade etmektedir. r_0 ve θ_0 değerleri ise h değerinin hesaplanması için kullanılan, irisin her bölgesinin polar koordinatlardaki ifadeleridir. $sgn(Re, Im)$ fonksiyonu ile ise alınan integrallerin sonucu olarak gelen karmaşık ifadenin gerçek ve sanal kısımlarının işareti bulunmaktadır. Alınan bu işarete göre faz bilgisi bulunmuş olur. Faz bilgisi her piksel için karmaşık düzlem üzerindeki 4 çeyrek bölgeden birini ifade eder. Pikselin faz bilgisi hangi bölgeye düşüyorsa o bölgeyi ifade eden 2 bitlik karşılığa göre niceleme işlemi gerçekleştirilir. Böylece her pikselin 2 bitlik bir karşılığı olmakla beraber, piksel verilerinin hepsinden bir bit matrisi elde edilir. Faz verisi bulunmasında Şekil 3.2 deki karmaşık düzlem ve bu düzlemdeki çeyreklere karşılık gelen bit ifadeleri kullanılır [16].



Şekil 3.2. Kompleks düzlem ve her çeyreğin karşılığı olan bit verisi

Karşılaştırılacak olan iris şablonunu oluşturmak için kullanılan açık kodlu "İris tanıma Sistemi" nde uygulama olarak 1 boyutlu Gabor dalgacığı kullanılmıştır. Burada 2 boyutlu olan normalleştirilmiş iris pikselleri tek boyutlu sinyale çevrilmiş, ardından da 1 boyutlu Gabor filtresinden geçirilmiştir. 1 boyutlu Gabor filtresinin sonucu olarak gerçek ve sanal tepki sinyalleri elde edilir. Gerçek ve sanal tepki sinyalleri aynı zamanda çift ve tek simetrik sinyal parçaları olarak da bilinir. Bu sinyallerin işaretleri incelenerek karmaşık düzlem üzerinde hangi bölgeye karşılık geldiği bulunur. Karmaşık düzlem üzerindeki bölgedeki 2 bitlik veri her piksel için karşılık olarak seçilir ve bitlerden oluşan bir iris şablonu elde edilir. Buna ek olarak elde edilen tepki sinyallerinin hem gerçek hem de sanal kısmının 0 değerini aldığı durum ise bize gürültülü bir veri olduğunu gösterir. Bu gürültü verisi resimdeki parlamaları ve bozulmaları içerir. Hem gerçek hem de sanal değerin 0 olduğu (0,0) noktası karmaşık düzlem üzerinde hiçbir bölgeye ait değildir. Gerçek ve sanal tepkinin 0 olduğu bu noktaların pozisyonları ise maske adı verilen ayrıca bir bit matrisinde tutulur. Bu bit matrisinde sonucun 0 olduğu bölgelere 1 değeri verilerek saklanır. Saklanan bu maske bilgileri karşılaştırma aşamasında dikkate alınmaz, hesaplamada gürültülü veriler kullanılmaz [4]. Özellik çıkarımı işlemi Şekil 3.3 de gösterilmiştir.



Şekil 3.3. Özellik çıkarımı işlemi

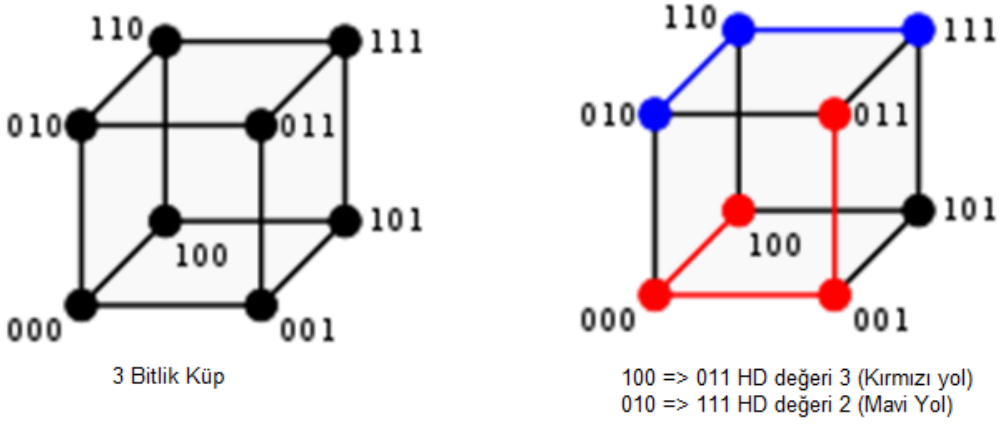
Özellik çıkarımı aşaması sonrası bir diğer önemli adım olan iris karşılaştırma işlemi yapılır.

3.2. İrislerin Karşılaştırılması

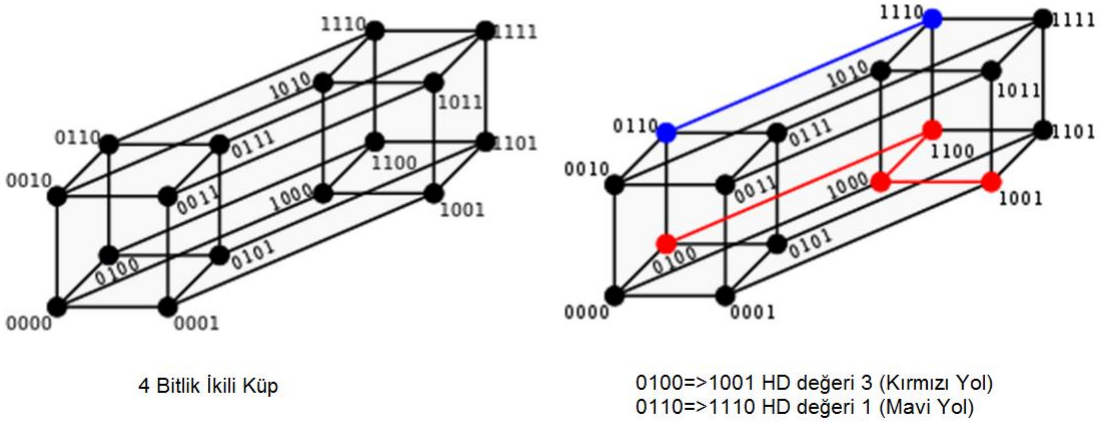
Hamming Mesafesi Algoritması

İris karşılaştırma işleminde en yaygın olarak kullanılan algoritma “Hamming Mesafesi” algoritmasıdır. “Hamming Mesafesi” algoritması bir matematikçi olan, telekomünikasyon ve bilgisayar bilimleri alanında çalışan Richard Wesley Hamming tarafından bulunmuştur. Bu algoritma ilk olarak telekomünikasyon sinyallerinde meydana gelen bozulmaları tespit ve bu bozulmaların düzeltilmesi amacıyla kullanılmıştır [17]. Richard Wesley Hamming in çalışmaları 1950 yılında yayınlanmış ve bu tarihten sonra birçok alanda kullanılmaya başlanmıştır. “Hamming Mesafesi” algoritmasında karşılaştırılacak olan iki matrisin birbirinden farklılığı bulunur. İris karşılaştırmada özellikleri çıkarılan iris şablonları bir veritabanında saklanır. Dışarıdan alınmış olan irisin şablon verisi ile veri tabanındaki irisler teker teker karşılaştırılır. “Hamming Mesafesi” algoritmasında nasıl işlem yapıldığı basit şekilde aşağıda gösterilmiştir.

“BAŞAR” ve “BEŞER” sözcüklerinin “Hamming Mesafesi” değeri 2 dir. Bu iki sözcük arasındaki farklılıklar A ve E harfleri arasındadır. Diğer harfler ise her iki sözcükte aynıdır. Aynı şekilde 1011101 ve 1001001 bit dizileri arasındaki “Hamming Mesafesi” değeri de 2 dir. “Hamming Mesafesi” algoritması iki nokta arasındaki en kısa yolun bulunmasında da kullanılır. 3 bitlik ve 4 bitlik ikili küplerde iki nokta arasında en kısa mesafenin bulunması örnekleri sırasıyla Şekil 3.4 ve Şekil 3.5 de gösterilmiştir.



Şekil 3.4. 3 bitlik ikili küp üzerinde Hamming Mesafesi bulma örneği



Şekil 3.5. 4 bitlik ikili küp üzerinde Hamming Mesafesi bulma örneği

İris tanıma sistemlerinde "Hamming Mesafesi" algoritması biraz daha geliştirilmiş şekilde kullanılır. Burada sadece bitler halinde karşılaştırma yapılmayıp aynı zamanda gürültü bilgilerini de içeren maskeler de kullanılarak hesaplamada yanlış sonuca erişmemize neden olacak gürültülü kısım çıkarılır. "Hamming Mesafesi" algoritması iris karşılaştırma işleminde Eş. 3.3'deki gibi ifade edilir [18].

$$HM = \frac{1}{N - \sum_{k=1}^{N} Xnk \text{ OR } Ynk} \sum_{j=1}^{N} Xj \text{ XOR } Yj \text{ (AND) } Xnj' \text{ (AND) } Ynj' \quad (3.3)$$

Bu eşitlikte iki iris şablonu Xj ve Yj şeklinde ifade edilmiştir. Bu şablonlar özellik çıkarımı aşamasında elde edilen bit matrisleridir. Bu iki şablon mantıksal XOR operasyonuna tabi tutulur. Mantıksal XOR operasyonunun ne şekilde çalıştığı Çizelge 3.1 de gösterilmiştir.

Çizelge 3.1. XOR operasyonu çalışma mantığı

Giriş1	Giriş2	Çıkış
0	0	0
0	1	1
1	0	1
1	1	0

Çizelgede gösterildiği üzere giriş ve çıkışın aynı olduğu durumlarda çıkış değeri 0 olup

geri kalan durumlarda çıkış 1'dir. "Hamming Mesafesi" algoritmasında giriş ve çıkışın farklı olduğu bitler bulunarak iki irisin birbirinden farklılığı tespit edilir. Karşılaştırılmakta olan her iki irisin gürültü maskeleri ise AND operatörü ile işleme sokulmaktadır. Özellik çıkarımı işlemi sırasında resmin gürültülü kısımlarının maske matrislerinde tutulduğu ve bu maskelerin gürültü verisini elimine etmek amacıyla kullanıldığı ifade edilmişti. Karşılaştırılmakta olan her iki irisin maske verileri bitler halinde birbirleri ile AND işlemine tabi tutulur. Burada maske verilerinden herhangi birisinde dahi gürültü yoksa yani değeri "0" ise sonucun da "0" olması sağlanır. Daha sonra bu iki maskenin AND işlemine tabi tutulması sonucu ortaya çıkan sonucun mantıksal tersi ile daha önce XOR işlemine tabi tutulmuş şablonların çıkış değeri AND işlemine tabi tutulur. Böylece ilgili bit her iki iris için gürültü verisi içeriyorsa o bitin işleme tabi tutulmasının önüne geçilmiş olur. Eğer bu bit her iki iris için de herhangi bir şekilde gürültü verisi içermiyorsa o zaman XOR işleminin sonucu önemli hale gelir. Çıkışın 1 olduğu durumların sayısı, yani her iki iris bitinin birbirinden farklı olduğu durumların sayısı bulunur.

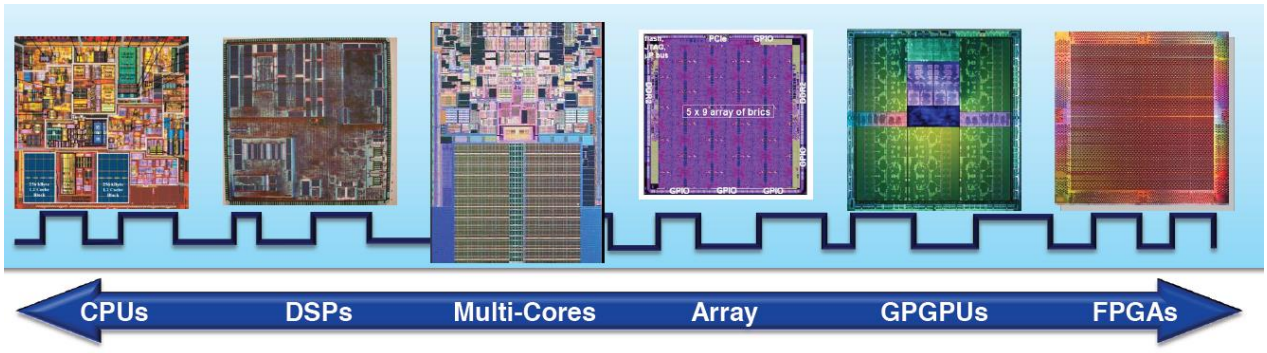
Eş. 3.3 de ifade edildiği gibi payda da yer alan gürültü maskelerinden elde edilen verilerin dışında kalan verilerin toplamı alınarak gürültü içermeyen bitlerin sayısı bulunur. İki iris arasındaki farklı bitlerin sayısı, gürültü içermeyen bit sayısına bölünerek iki irisin birbirinden farklılığının ifadesi olan "Hamming Mesafesi" değeri bulunmuş olur. En basit şekliyle Hamming Mesafesi ifadesi Eş. 3.4 deki gibidir.

$$\mathrm{HM} = \frac{\text{Gürültüden Etkilenmemiş Farklı Bitlerin Sayısı}}{\text{Gürültüden Etkilenmemiş Toplam Bit Sayısı}} \tag{3.4}$$

Eş. 3.4 de HM değerinin "1" ile "0" arasında olacağı görülür. Tüm bitlerin birbirinden farklı olduğu durumda HM değeri 1 olur. Teorik olarak iki irisin birbiriyle eşleştirildiği durumda ise HM değeri "0" olur. Fakat gerçek dünya uygulamalarında HM değerinin 0 olmasını sağlamak çok da mümkün görünmemektedir. Resimde gürültü olarak algılanmayan bazı bozuklukların varlığı HM değerinin "0" çıkmasına engel olur. Bu nedenle 0'a en yakın olan yani, sonuçlar arasında en küçük olan HM değerini sağlayan irisler arasında eşleştirme yapılır.

4. OPENCL (Açık Hesaplama Dili)

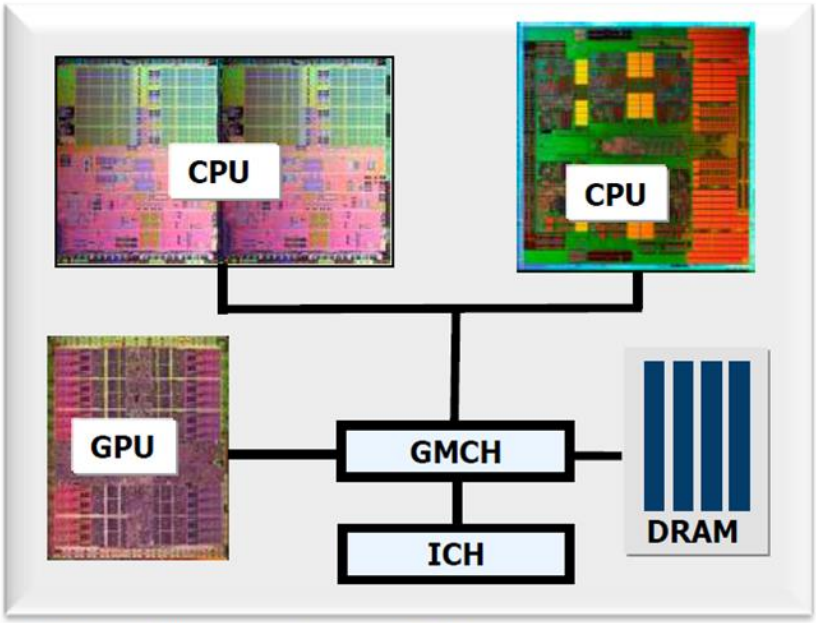
Heterojen platform CPU, GPU, FPGA ve DSP kartları gibi donanımsal ve işlevsel olarak birbirinden farklı olan cihazların iki ya da daha fazlasının bir arada bulunduğu sistemlere verilen isimdir. OpenCL heterojen platformlar üzerinde ortak bir kodun koşmasını sağlayan C tabanlı bir dildir. OpenCL kullanarak heterojen bir platformda istenilen cihazlar üzerinde aynı kod koşturulabilir. Heterojen platform sunucu ve hedef cihaz ya da cihazların birbirine bir PCIe (Peripheral Component Interconnect Express) ara yüzü ile bağlanmasını ifade edebileceği gibi son zamanlarda oldukça yaygın kullanılan "chip içinde sistem" adıyla bilinen tek entegre devre içinde CPU, FPGA ya da GPU nun bir arada bulunmasından da oluşabilir (Şekil 4.1). OpenCL, Khronos grup olarak bilinen bir konsorsiyum tarafından ortaya konulmuştur. Apple, Intel, Qualcomm, Advanced Micro Device (AMD), Nvidia, Altera, Samsung, ARM holdings ve Vivante bu konsorsiyum'un üyeleridir. Khronos konsorsiyumu grafik, medya ve paralel programlama standartlarını belirlemektedir [19]. OpenCL destekleyen donanımsal platformlar oldukça geniş donanım ailelerini kapsamaktadır. Bu platformlara her geçen gün yeni ürünler eklenmektedir. Bu ürünlerin ortak özellikleri çoklu çekirdek yapısına sahip olmalarıdır. Çekirdekler üstünde her bir donanımın kendi çalışma frekansına göre işlemler gerçekleştirilir. Çekirdek sayısı arttıkça bir donanım üzerinde eş zamanlı olarak yapılabilecek işlemlerin sayısı da artar. Günümüzde çoklu çekirdek yapısının yaygın olarak kullanılmaya başlaması ve gittikçe daha da yayılmasının altında yatan sebep de aynı anda çoklu işlem yapabilme kapasitesinin arttırılmak istenmesidir. Son dönemde yüksek hızlı işlem gücü gerektiren uygulamalarda çoklu çekirdek yapısını barındıran donanımlar kullanılmakta, işlem gücünün bu yeni teknoloji cihazlarla arttırılması sağlanmaktadır.



Şekil 4.1. OpenCL destekleyen donanım platformları

4.1. Heterojen Platform Mimarisi

Heterojen platform bir ya da daha fazla CPU, GPU, DSP işlemcisi, FPGA ve hızlandırıcı bazı kartlardan oluşabilir. Heterojen sistemler, işlemlerin yalnızca CPU üzerinde yapıldığı sistemlerden birçok açıdan farklılık gösterir. Aslında bu farklılıklar heterojen sistemlerin neden ortaya çıktığının da cevabıdır. Bir sistemin gerçekleştireceği tüm işlemleri CPU üzerinde yapmak CPU ya fazla yük binmesine neden olur. CPU'ların temel çalışma frekansları her ne kadar diğer hızlandırıcı kartlardan fazla da olsa her işlemin CPU tarafından yapılması işlemlerin belli bir sıraya sokulmasını gerektirir ve dolayısıyla işlemleri yavaşlatır. İstenilen performansın sağlanması için sisteme fazladan CPU eklenirse hem sistemin maliyetini arttırır hem de güç harcaması oldukça yükselir. Bu sorunların çözümü içim daha az güç harcayan ve daha ucuza mal olabilecek kartlar heterojen sisteme dâhil edilerek sistemin maliyeti azaltılabilir. Aynı zamanda GPU, FPGA ve diğer hızlandırıcı kartlar daha az güç tükettiği için de sisteme avantaj sağlar. İşlem performansı açısından da bu cihazlar kendilerine has mimarileri sayesinde paralel hesaplama yapabilecek kabiliyetlere sahiptir. Dolayısıyla heterojen bir sistem kullanmak hem güç tüketimini azaltmak, hem performansı arttırmak hem de sistemi daha ucuza mal etmek bakımından avantajlıdır. Bir heterojen sistem örneği Şekil 4.2 den görülebilir.

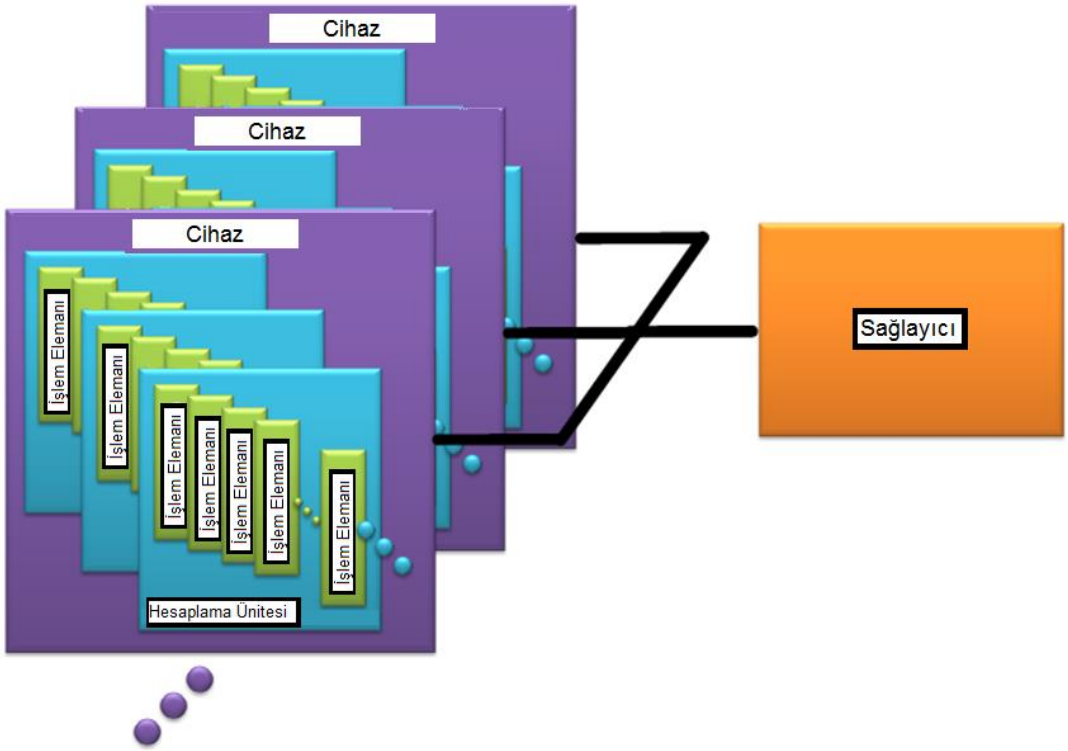


Şekil 4.2. Bir heterojen platform örneği

Şekil 4.2'de gösterilen örnek heterojen sistemde bir veri yolu üzerinden iki CPU birbirine bağlanmıştır. Her iki CPU'nun kontrol ettiği ve yürüttüğü işlemler mevcuttur. Grafik hafızası kontrol birimi ise GPU nun RAM bellekten veri okumasını ya da yazmasını sağlar. Giriş/çıkış kontrol ünitesi ise dışarıdan bilgi almaya ya da dışarıya bilgi çıktısı üretmeye kanal sağlar. Benzer şekilde birçok heterojen sistem mimarisi kurmak mümkündür. Mevcut olan ihtiyacı karşılayacak şekilde heterojen sistemlerin tasarımı yapılır. Günümüzde heterojen sistem mimarisi oldukça yaygın olarak kullanılmaktadır. Cep telefonları heterojen sistemlere örnek olarak gösterilebilir. Cep telefonlarında kullanılan CPU lar hızlı veri işlemlerinin gerçekleştirilmesini, cep telefonu işletim sistemi üzerindeki çalışacak uygulamaların koşturulmasını sağlarken, sisteme gömülü olarak entegre olan bir DSP üzerinden de dışarıdan alınan ses verisi sayısallaştırılarak bit verileri haline dönüştürülür. Daha sonra bu bit verileri tekrar kulağın duyabileceği ses sinyallerine çevrilir. Böylece DSP, CPU'dan bağımsız olarak kendi görevini yerine getirir. Aynı zamanda bunu düşük güç tüketimiyle yaptığı için batarya ömrünün de uzamasını sağlamış olur [19].

4.2. OpenCL Platform Modeli

OpenCL heterojen mimaride aynı program kodunun herhangi bir değişikliğe gerek kalmadan tüm cihazlar üzerinde işlem yapmasını sağlar. OpenCL platformunda bir cihaz sağlayıcı (host), diğer cihazlar ise hesaplama birimi olarak kullanılır. Host adı verilen cihaz (genellikle bir CPU) diğer hesaplama birimi olarak kullanılan birimlere görev dağıtımı yaparak ilgili birimler üzerinde bellek bölgesinin ayarlanması, ayarlanmış bu bellek bölgelerine gerekli verilerin yazılması, hesaplama birimi olarak kullanılan birimlerde yapılan hesaplamaların sonuçlarının okunması gibi görevlere sahiptir. Tabii olarak sağlayıcı görevinde kullanılan bir CPU, bu işlemlerin yanında OpenCL konseptinden bağımsız şekilde sistemin ihtiyaçlarını karşılamaya yönelik başka işlemler de yapabilmektedir. OpenCL platform modeli Şekil 4.3'de gösterilmiştir [20].



Şekil 4.3. OpenCL platform modeli

Bir sağlayıcı sistem mimarisi içinde tüm hesaplama birimlerine bağlıdır. Sağlayıcı cihaz tarafından bu hesaplama birimlerinde yer alan hesaplama ünitelerine işlem elemanlarının atamaları gerçekleştirilir. Bu atamalar sonucunda her birim kendine atanan işlemi kendi içinde yaparak sonuçları tekrardan sağlayıcı cihaza dönerler. Sonuçların okunması ve tekrardan değerlendirilmesi işlemi ise sağlayıcı cihaz tarafından gerçekleştirilir. OpenCL platform modelinde hesaplama birimlerine paylaştırılmış olan işlem elemanları genellikle bir algoritmanın gerçeklenmesi şeklindedir. Genelde paylaştırma aynı işlemin tüm hesaplama birimlerinde farklı veriler üstünde yapılması şeklindedir. Yapılabilecek iki çeşit paralelleştirme mevcuttur.

Veri paralel model

Veri paralel modelde, paralelleştirilmesi istenen bir algoritma OpenCL ile yazılır. Oluşturulan bu kod çekirdek (kernel) olarak ifade edilir. Çekirdekler her hesaplama birimi üzerinde koşabilen OpenCL fonksiyonlarıdır. Bu kodlar sağlayıcı cihaz üzerinde çalışmayan, ama sağlayıcı cihaz tarafından hesaplama birimlerine gönderilen ve hesaplama birimleri üzerinde çalışan fonksiyonlardır. Çekirdekler hesaplama cihazlarına gönderilip

ilgili cihaz için organize edilirler. Her çekirdek, hesaplama cihazlarında bir işlem elemanı olarak işlem yapar. Veri paralel model, çalışan bu kodların farklı veriler üzerinde eş zamanlı olarak çalıştırılarak tüm verilerin senkron olarak işleme tabi tutulmasını ifade eder.

İşlem paralel model

İşlem paralel modelde ise bir algoritma çeşitli parçalara ayrılır. Her parça hesaplama cihazları üzerinde koşturulur.

4.3. OpenCL Uygulama Modeli

OpenCL uygulama modelinin 5 başlık altında incelenmesi gerekir.

İş Nesnesi: Hesaplama cihazında çalışacak olan en temel işlem elemanıdır. İş nesnelerinin kodlarına çekirdek adı verilmektedir.

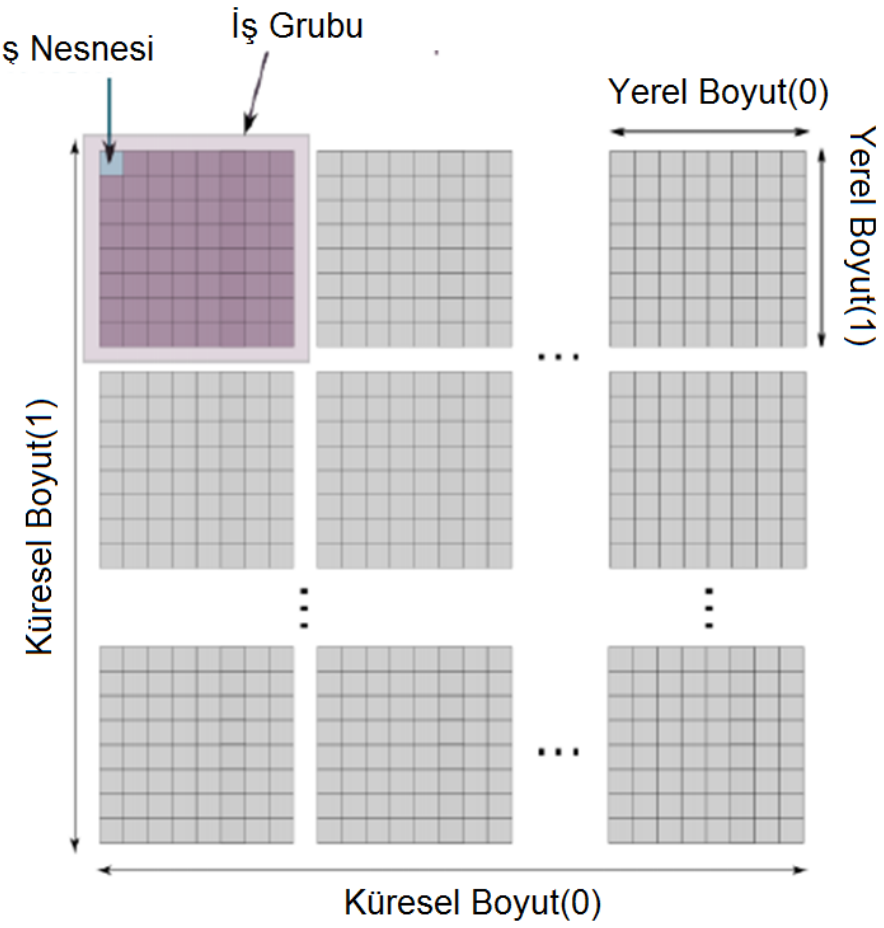
İş Grubu: İş grubu iş nesnelerinden oluşan bir kümedir. Bu küme içinde yer alan iş nesneleri aynı zamanda işlem yapar. Fakat farklı iş grupları aynı cihaz üzerinde dahi olsa farklı zamanda işleme tabi tutulabilirler.

Program: Bir ya da daha fazla çekirdeği içinde barındırır. Bir ya da daha fazla çekirdekten oluşan, çekirdek koleksiyonu olarak da tanımlanabilir.

Kaynak: İş nesnelerini çalıştıran çevrenin bütünüdür. Bu bütünün içinde çekirdeğin çalışacağı cihaz bilgisi, cihazın hafızaları ve cihaz üzerinde koşacak olan kod bulunur.

Bu tanımlar OpenCL standartlarınca belirlenmiş ve bir terminoloji oluşturulmuştur [21]. İş nesnesinden kaynağa doğru büyüyen bir model söz konusudur. Sonuç olarak kaynak hem iş nesnelerini hem de cihaz ve cihaza yönelik bilgileri içeren bir topluluk oluşturur. Şekil 4.4’ de iş nesnelerinin iş grupları içindeki yerleri gösterilmiştir. Burada her iş nesnesinin ait olduğu bir iş grubu mevcuttur. İş grubu sayısı ise donanımın özelliklerine göre değişir. Donanım ne kadar çok sayıda iş nesnesi ve iş grubuna altyapı sağlıyorsa, o kadar paralelleştirmeye imkân sağlıyor demektir. Çünkü iş nesneleri eş zamanlı olarak

çalışabilecek olan fonksiyonların bir örneği olarak donanım üzerinde gerçekleştirilmektedir.



Şekil 4.4. OpenCL uygulama modeli

Yine Şekil 4.4'den görülebileceği gibi her iş grubunun boyutu yerel boyut, tüm iş nesnelerinden oluşan tüm modelin boyutu küresel boyut olarak tanımlanır [22]. OpenCL Uygulama Modeli içinde değerlendirilebilecek bir nokta da CPU ile GPU arasındaki iletişim ve programın gerçeklenmesi adımlarıdır. Bu adımlar genel hatlarıyla aşağıdaki gibi ifade edilebilir [15].

1. İlk olarak sağlayıcı (CPU) Şekil 4.4 deki matris yapısına uygun şekilde iş nesnelerinin ve iş gruplarının oluşturulmasını sağlar. Bu yazılımsal olarak küresel ve yerel boyutların belirlenmesiyle yapılır. Ama aslında sağlayıcı yazılımında yapılan bu ayarlama hesaplama cihazı üzerinde bir bellek bölgesini donanımsal olarak da ifade eder. Yani yazılıma uygun şekilde hesaplama cihazının bellek bölgesinde donanımsal bir bölge

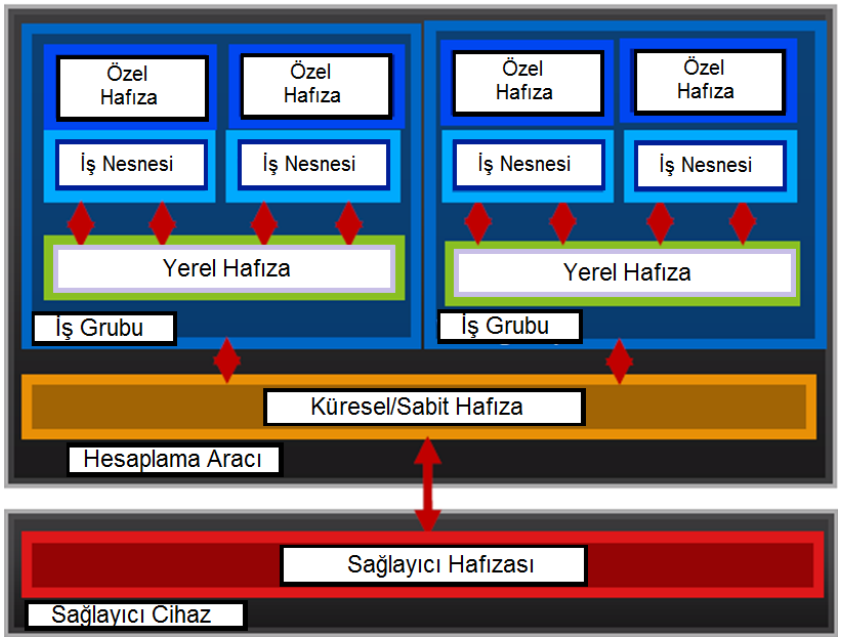
ayarlanır. İlk aşamada problemin kaç adet iş nesnesi ile çözüldüğünün tespit edilmesi gerekir.

2. Daha sonra kaç adet iş grubu kullanılacağı belirlenir. İş grubu belirlenirken dikkat edilmesi gereken birbirleriyle etkileşimde olan iş nesnesi olup olmadığıdır. Çünkü bir iş grubu içinde aynı yerel hafızanın kullanılması mümkündür. Daha sonra bu iş grupları, OpenCL ile yazılmış kod, ve cihaz bilgileri de bulunduran kaynak oluşturulur.
3. Sağlayıcı (CPU) iş grupları, cihaz bilgisi ve cihaz üzerinde koşturulacak kodu içeren kaynağı hesaplama cihazının küresel hafızasına yerleştirir.

OpenCL uygulamalarında dikkat edilmesi gereken bir diğer konu ise hesaplama cihazının özelliklerinin iyi bilinmesidir. Donanımsal olarak hesaplama cihazının kullanıcıya sunduğu altyapı belirlidir. Kullanılacak olan donanımda ön plana çıkan unsur, bir iş grubunda yer alabilecek maksimum iş nesnesi sayısıdır. Bu ne kadar çoksa o kadar paralel işlem yapabilme kabiliyeti artar. Sağlayıcı tarafından (CPU) kendisine bağlı olan cihazların özellikleri okunabilir.

4.4. OpenCL Hafıza Modeli

OpenCL hafıza modelinde her hesaplama biriminin içinde özelleştirilmiş hafıza bölgeleri bulunur. Bu bellek bölgeleri Şekil 4.5 de gösterilmiştir [19].



Şekil 4.5. OpenCL hafıza modeli

Özel Hafıza (Private Memory): Özel hafıza sadece iş nesnesi tarafından erişilebilen hafızadır. Her iş nesnesi bir özel hafıza elemanına erişebilmektedir. Her özel hafıza elemanı bir iş nesnesine özel olarak adanmış şekildedir. Yalnızca ilgili iş elemanı hafıza bölgesine yazıp okuma yapabilmektedir.

Yerel Hafıza (Local Memory): Yerel hafıza aynı iş grubunda yer alan iş nesneleri tarafından erişilebilen bir hafızadır. İş nesneleri bu hafıza türünü ortak olarak kullanmaktadır. İş nesnesinin bu hafızaya ulaşması özel hafızaya ulaşmasından daha uzun zaman almaktadır.

Küresel Hafıza (Global Memory): Küresel hafıza tüm iş gruplarındaki iş nesneleri tarafından erişilebilir bir hafızadır. Küresel hafıza bir anlamda tüm hesaplama cihazının ortak hafızası olarak düşünülebilir. Küresel hafıza hem yazma hem de okuma işlemlerinde kullanılabilir.

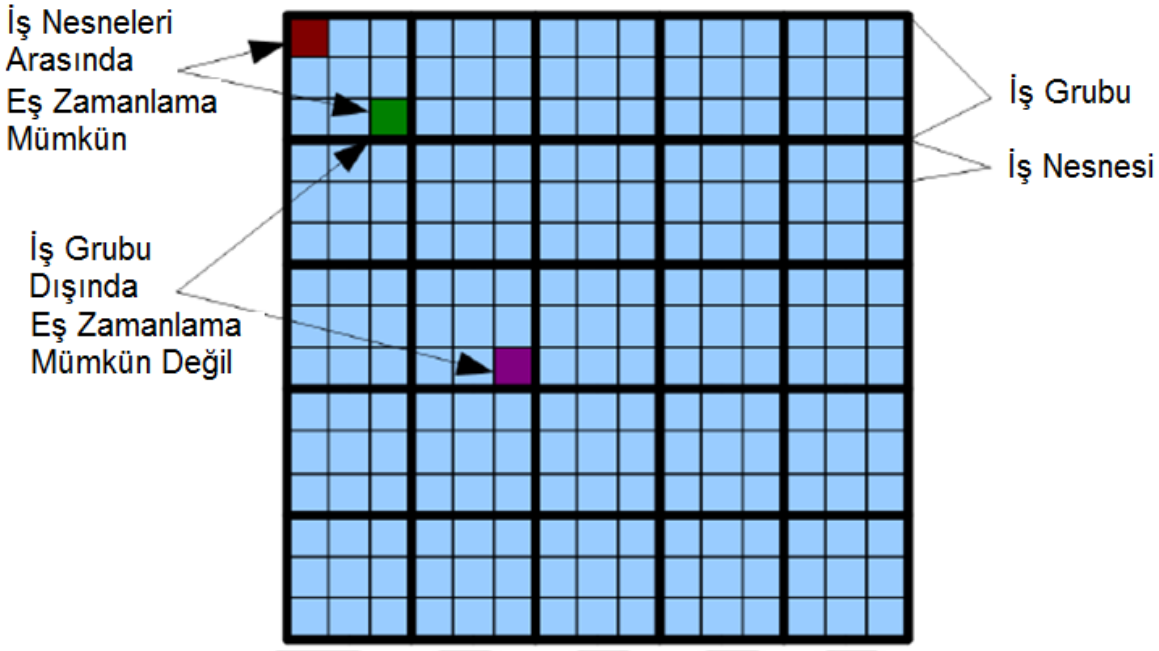
Sabit Hafıza (Constant Memory): Tüm iş gruplarındaki iş nesneleri için erişilebilir bir hafızadır. Fakat bu hafızada küresel hafızadan farklı olarak sadece okuma yapılabilir.

Sağlayıcı Hafızası (Host Memory): Bu hafıza sağlayıcı cihazın hafızasıdır. Genel olarak CPU kullanıldığı için CPU'nun kullandığı hafıza olarak düşünülebilir.

Hesaplama cihazları programlanırken veri akışı CPU dan başlayarak, küresel hafıza, yerel hafıza ve son olarak da özel hafızaya doğru gerçekleşir. Hesaplama cihazı CPU tarafından gönderilen bu verileri giriş olarak kullanarak ürettiği sonucu aynı yol üzerinden geri iletir. Bu sefer izlenen yol özel hafıza, yerel hafıza daha sonra hesaplama cihazından çıkmadan önce küresel hafızadır. Bu aşamadan da sonra veri CPU'nun hafızasına iletilir. CPU üzerinde hesaplama birimlerinden gelen sonuçların değerlendirilmesi ya da işlemin akışına göre sonuçların ilgili şekilde kullanılması gerçekleştirilir. Bu kısım uygulamanın ne olduğuyla alakalı olarak değişir.

OpenCL kodu ile belirlenmiş olan işlemler iş nesneleri tarafından donanım üzerinde gerçekleştirilirken eş zamanlama ile ilgili bir kural mevcuttur. Bu kurala göre aynı iş grubunda olan iş nesneleri aynı zamanda görevlerini gerçekleştirirler. Farklı iş grubunda yer alan iş nesneleri ise birbirine hiçbir şekilde senkron olamazlar. Şekil 4.6'da bu durum

görsel olarak anlatılmıştır.



Şekil 4.6. İş nesneleri arasında senkronizasyon

5. UYGULAMA

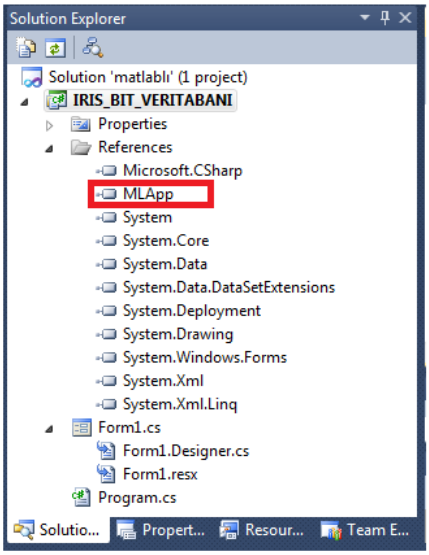
Bu tezin konusu olan uygulama üç basamaklı olarak belirlenmiştir. İlk basamak olarak bir C# yazılımı yazılarak iris bit veri tabanı oluşturulmuştur. Daha sonra Hamming Mesafesi algoritması C++ dilinde yazılmış ve Xeon işlemci üzerinde bu yazılımın ne kadar zamanda işlem yaptığı gözlenmiştir. Daha sonra C tabanlı OpenCL dili kullanılarak Hamming Mesafesi algoritması paralelleştirilmiş ve aynı zamanda heterojen sistemlere uygun hale getirilmiştir.

5.1. İris Bit Veri Tabanının Oluşturulması

İris bit veri tabanının oluşması için öncelikle iris resim veri tabanının bulunması lazımdır. Bu çalışmada iris resim veri tabanı olarak UBIRIS V1 veri tabanı kullanılmıştır. Bu veri tabanının kullanılma nedeni Libor Masek'in "açık kodlu iris tanıma sistemi" nin gri tonda resimleri giriş olarak kabul etmesidir. Bu aşamada renkli resimlerden oluşan bir veri tabanı seçilerek gri tona çevirme işlemi de gerçekleştirilebilirdi fakat UBIRIS V1 veri tabanı gri tonda resimler içerdiği için bu aşamada kolaylık sağlamaktadır. Ayrıca bu veri tabanında iris resimleri üzerindeki gürültüler minimize edilmiştir. Bu durum ise bize yanlış karşılaştırma oranını en aza indirmek gibi bir avantaj sağlamıştır. Bu teze konu olan çalışmada doğru karşılaştırma oranı en yüksek veri tabanının seçilmesinin en uygun olduğu düşünülmüştür. Çünkü çalışmanın amacı yanlış karşılaştırmaları önlemek ya da azaltmak değil, karşılaştırma algoritmasının farklı donanımlar üzerinde (çoklu çekirdek altyapısından faydalanarak) hızlandırılmasını sağlamaktır. Daha önce Bölüm 3.1 de anlatılan 1D Gabor filtreleri tek boyutlu diziler için kullanılan filtrelerdir. Bu filtrelerin kullanılması için resim öğesinin tek boyutlu olması gerekmektedir. Gri tondaki resimlerin her satırı tek boyutlu diziler olup, 1D Gabor filtreler bu resimlere uygulanabilmektedir. 1D Garbor filtreden geçirilen iris resimlerinden gerçek ve sanal tepki sinyalleri elde edilir. Gerçek ve sanal tepki sinyallerinin işaretlerinden faz belirlemesi yapılarak iris resminin bit matrisi şeklinde ifadesi olan şablon ve gürültü verisinin tutulduğu maske bit matrisi elde edilir.

İris bit veri tabanının oluşturulmasında Libor Masek'in yazmış olduğu ve akademik kullanıma ücretsiz olarak açtığı "açık kodlu iris tanıma sistemi" C# ara yüzü yazılarak

kullanılmıştır [4]. Bir C# programına ihtiyaç duyulmasının nedeni, iris resimlerinden oluşmuş olan iris resim veri tabanında yer alan tüm resimlerin aynı işlemlerden geçmesini otomatik olarak sağlamaktır. Yani iris resimlerinden, iris bit veri tabanının otomatik olarak oluşturulmasını sağlamak için sırasıyla her iris resmine aynı sinyal işleme işlemlerinin uygulanması (iris bölgesinin belirlenmesi, iris normalleştirmesi gibi) yazılmış olan bu C# yazılımıyla otomatikleştirilmiştir. C# programlama dili, MATLAB fonksiyonlarını çalıştırmayı sağlayan bir altyapıya sahiptir. Bu işlem iki şekilde gerçekleştirilebilir. Birinci olarak, C# kodları arasında MABLAB kodları yazarak, C# programlama dili aracılığıyla bu kodlar çalıştırılabilir. İkinci yöntem ise MATLAB da yazılmış olan ".m" uzantılı dosyaların bulunduğu klasör, çalışma klasörü olarak programa gösterilerek bu dosyaların C# programı tarafından kullanılmasını sağlamaktır. Libor Masek in paylaşmış olduğu açık kaynak kodlarını ikinci olarak anlatılan şekilde kullanılarak iris şablonu ve gürültü maskesi verisinden oluşan iris bit veri tabanı oluşturulmuştur. MATLAB da oluşturulmuş olan ".m" uzantılı MATLAB dosyalarının C# programlama dili aracılığıyla çalıştırılması amacıyla öncelikli olarak MATLAB referansının C# projesine eklenmesi gerekmektedir. "Interop.MLApp" isimli bu referans C# projesinin referanslarına Resim 5.1 de gösterildiği üzere eklenmiş olmalıdır. İlgili referans, projeye eklendikten sonra bir MATLAB objesinin oluşturulması gerekmektedir. Bu aşamadan sonra ".m" uzantılı dosyaların bulunduğu klasörün çalışma klasörü olarak gösterilmesi gerekmektedir. Çalışma klasörünün gösterilmesi "execute" isimli fonksiyon kullanılarak yapılmaktadır.



Resim 5.1. Projeye eklenen MATLAB referansı

Yukarıda anlatılmış olan MATLAB objesinin oluşturulması ve çalışılacak klasörün tanıtılması aşağıdaki kod parçasıyla gerçekleştirilmektedir.

```
MLApp.MLApp matlab = new MLApp.MLApp();
matlab.Execute(@"cd C:\Users\bugra\Desktop\tez\iriscode\iriscode\Segmentation");
```

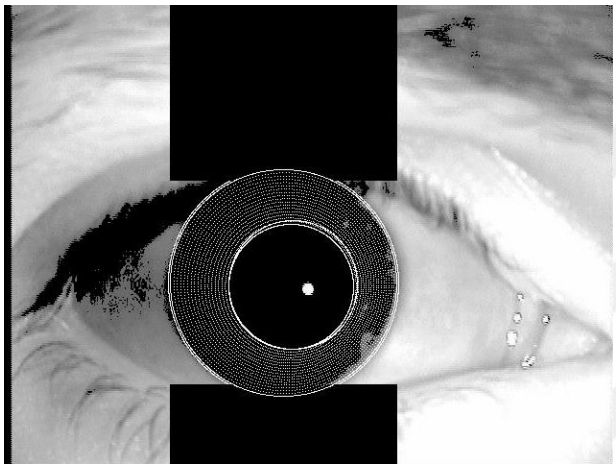
Bu aşamanın ardından açık kaynak kodlu iris tanıma sisteminin şablon ve maske oluşturulmasında kullanılan "createiristemplate.m" dosyası C# ara yüzünden çağrılarak her bir iris resmi için aşağıdaki şekilde gösterildiği gibi çalıştırılır.

```
foreach (FileInfo file in Files)
{
        try
        {

            matlab.Feval("createiristemplate", 2, out result, file.FullName);
            object[] res = result as object[];
            temp.Add(res[0]);
            mask.Add(res[1]);
            result = null;

        }
        catch
        {
                str += file.FullName+Environment.NewLine;

        }
}
```

Bu kod parçasında "foreach" komutuyla iris resim veri tabanındaki tüm resimlerin tek tek sırayla seçilmesi yapılmaktadır. Oluşturulmuş olan MATLAB nesnesinden "matlab.Feval" fonksiyonu yardımıyla, seçilmiş olan çalışma klasörünün içinde yer alan herhangi bir ".m" uzantılı dosya çağrılabilmektedir. Yukarıda gösterilen "matlab.Feval" fonksiyonunun yer aldığı kod satırında feval fonksiyonunun argümanları olarak; "createiristemplate.m" dosyasının kullanılacağı, bu dosya kullanılınca "result" objesine 2 adet çıktının alınacağı belirtilir ve son olarak da seçilen iris resminin tüm dosya yolu argüman gönderilir. Daha sonra result obje dizisine dönen sonuçlar ayrı ayrı "sablon" ve "maske" listelerine eklenir. Kodda yer alan "foreach" döngüsü tamamlandıktan sonra iris resim veri tabanındaki tüm resimlerin birer tane şablon birer tane de gürültü maskeleri iki ayrı listede tutulmuş olur. Bu listeler daha sonra her irise ait şablon ve gürültü verileri sıralı olacak şekilde bir metin

dosyasına kaydedilmiştir. Böylece bir iris bit veri tabanı oluşturulmuş olur.

Kullanılmış olan açık kaynak kodlu iris tanıma sisteminde yer alan “createiristemplate.m” dosyasında John Daugman’ ın “Rubber Sheet” metoduna uygun şekilde iris şablonu için veri boyutu seçilmektedir. Bu çalışmada iris boyutu olarak 8x256 ve 16x480 bitlik iki boyut seçilmiştir. Bu boyutların ayarlanmasında açık kaynak kodlu iris tanıma sistemi koduna müdahale edilmiştir. Yukarıda anlatılmış olan kodun işleyişiyle iris resmi Resim 5.2 de gösterildiği şekilde taranmaktadır.

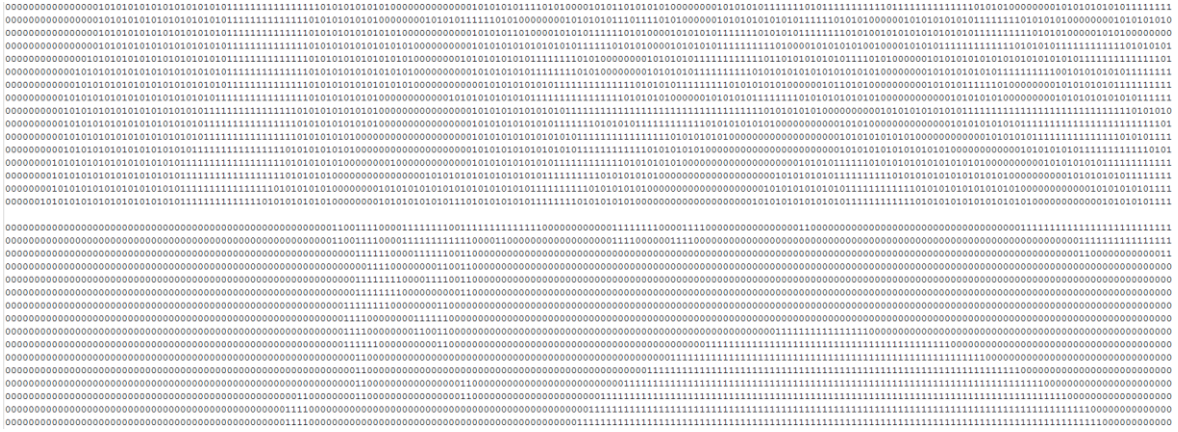


Resim 5.2. İşlem görmüş iris resmi

Taranan bu bölge daha sonra normalleştirilerek (polar koordinatlardan Kartezyen koordinatlara geçiş yoluyla) düz şerit şeklinde bir iris resmi elde edilir. Resim 5.3 de normalleştirilmiş iris resminin şerit şeklindeki görüntüsü gösterilmiştir.

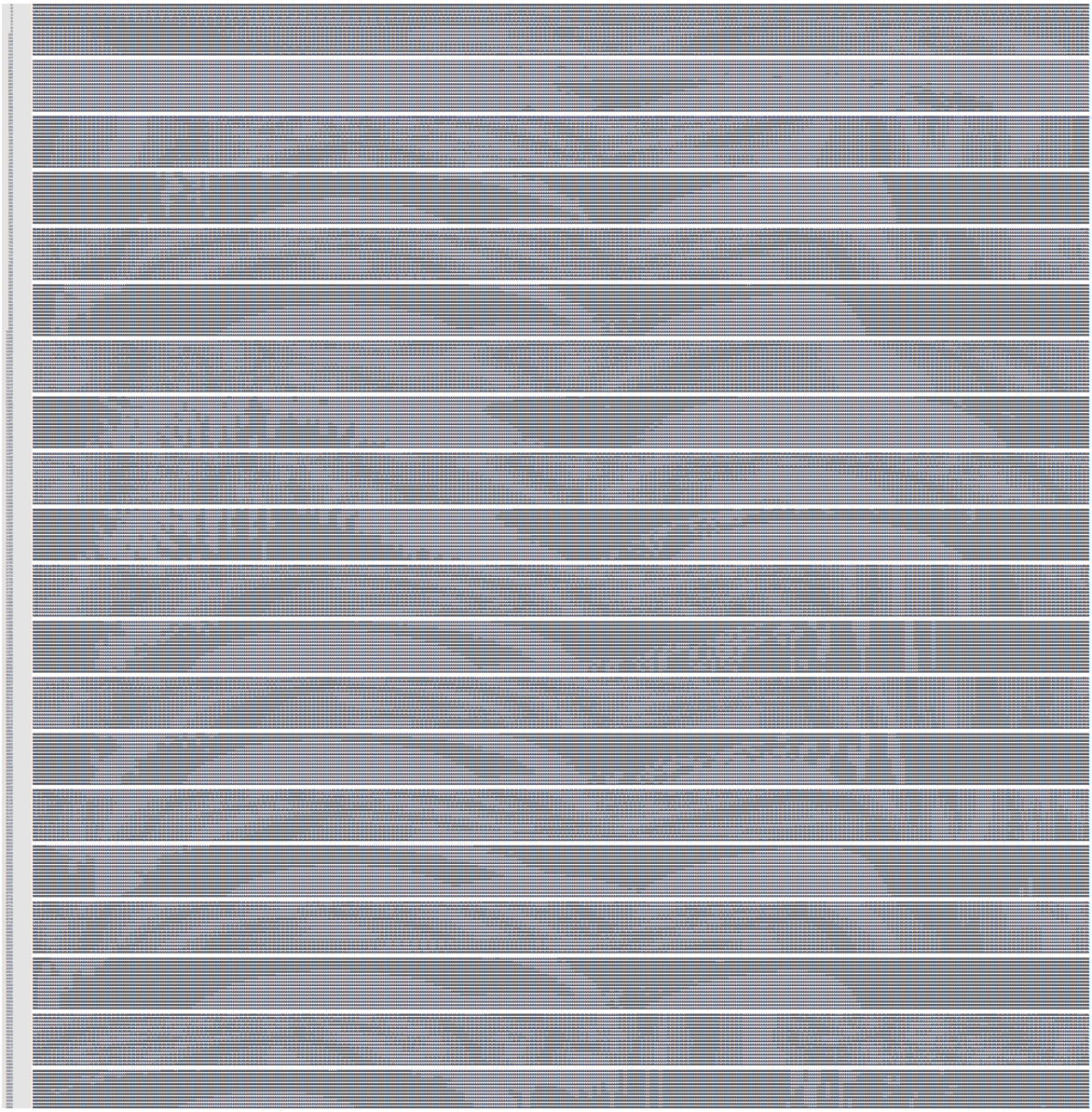
Resim 5.3. Kod kullanılarak normalleştirilmiş iris resmi

Yukarda değinildiği üzere “result” değişkeni ile elde edilen bir iris için şablon ve maske verileri için bir kesit Resim 5.4 de gösterilmiştir. Bit verilerinin tamamı enine uzun olup görüntüsel olarak anlaşılması güçleştiğinden dolayı belirli bir kesit alınarak Resim 5.4 de gösterilmiştir.



Resim 5.4. Bir iris için şablon ve maske verilerinden bir kesit

Tüm veri tabanı bu şekilde oluşturulmuş 1329 adet iristen meydana gelmektedir. İris bit veri tabanının bir ekranda görüntülenebilen kısmının görüntüsü Resim 5.5 deki gibi ortaya çıkmıştır. Veri tabanının ekran görüntüsünde her irisin kendine özgü şablon ve maskelere sahip olduğu görünmektedir. Veri tabanındaki şablon verileri incelendiğinde Resim 5.5 de bir örneği gösterilmiş olan normalleştirilmiş iris resimlerine benzerlik gösterdiği görülmektedir. Aslında bu beklenmekte olan bir sonuçtur, çünkü iris resminin bit matrisiyle ifadesi iris şablonunu oluşturmaktadır. Veri tabanının Resim 5.5 deki ekran görüntüsü incelendiğinde ortaya çıkan bir diğer sonuç ise görsel olarak iris şablon verilerindeki dokunun birbirinden oldukça farklı olduğudur. Herhangi bir karşılaştırma algoritması kullanmadan çıplak gözle bile iris şablonlarının dokusundaki farklılığın gözlenebilir olması, iris tanımanın oldukça güvenilir bir biyometrik tanıma sistemi olduğunu açıkça göstermektedir.



Resim 5.5. İris bit veri tabanından bir ekran görüntüsü

5.2. Hamming Mesafesi Algoritmasının Sıralı Kod Şeklinde Yazılması

Sıralı kod, komutların yazılış sırasına göre akan ve bir giriş verisinin işlemi bitmeden diğer giriş için işlem yapılmayan kod şeklinde tanımlanabilir. Hamming Mesafesi algoritmasının temel işlevi iki irisin birbiri ile karşılaştırılması ve birbirlerinden farklılıklarını ifade eden bir değerin bulunmasıdır. Bu teze konu olan çalışmada sıralı kod ifadesi her irisin tek tek birbiri ile karşılaştırmasını ifade eder. Bir iris diğeri ile karşılaştırılmadan, başka bir irisin karşılaştırılmasına geçilmez. Karşılaştırma işlemi kime ait olduğu bilinmeyen iris ve bu irisin sağa ve sola 8 kere kaydırılmasını içeren toplamda 17 iris verisinin, veri tabanındaki tüm irislerle teker teker karşılaştırılması ve bu karşılaştırmadan elde edilen Hamming

Mesafesi değerlerinin bulunması ile sona erer. Daha sonra bulunan bu değerler arasından en küçüğü ile bilinmeyen iris arasında eşleştirme yapılır. Böylece irisin veri tabanında kayıtlı olan kişilerden kime ait olduğu bulunmuş olur. Yapılan çalışmada UBIRIS V1 veri tabanından belirlenen bir iris resmi, Libor Masek'in akademik alanda çalışma yapan araştırmacılarla ücretsiz olarak paylaşmış olduğu açık kaynak kodu kullanılarak bit matrislerinin üretilmesi sağlanmıştır (şablon ve maske verileri). Bilinmeyen bir kişiye ait olarak karşılaştırmada kullanılacak olan bu iris resmi ayrı bir veri tabanında tutulmuştur. Hamming Mesafesi algoritması aşağıda belirtilen algoritmaya uygun şekilde kodlanmıştır.

Seri kod genel algoritma

- Bilinmeyen İris Verisini Oku.
- Tüm İris veri tabanını Oku.
- Veri tabanındaki İris ile Bilinmeyen İrisin HM Değerini Bul.
- Bunu veri tabanındaki Her İris için Tekrarla.
- Her Sağa ve Sola Kaydırılmış Bilinmeyen İrisler için 3. Ve 4. İşlemleri Tekrarla
- En düşük HM değerini bul ve irisi belirle.

HM değerinin bulunması

- Bilinmeyen İris ilgili biti ile veri tabanındaki İrisin ilgili bitini XOR işleminden geçir.
- Sonucu "Şablon" değişkenine at.
- Bilinmeyen İris Maskesinin ilgili biti ile veri tabanındaki İris Maskesinin ilgili bitinin terslerini AND işleminden geçir.
- Sonucu "Maske" değişkenine at.
- Eğer "Maske" değeri tersi '1' ise "Maske biti sayısı" nı 1 arttır.
- "Maske" değişkeni ile "Şablon" değişkenini AND işleminden geçir.
- Sonucu Tekrar "Şablon" Değişkenine At.
- Eğer "Şablon" Değeri '1' ise "Farklı Bit Sayısı" nı 1 arttır.
- İlk 8 işlemi iris şablonunu oluşturan tüm bitler için ayrı ayrı gerçekleştir.
- İris matrisindeki toplam bit sayısından (Satır*Sütun) , "Maske Biti Sayısı" çıkarılarak işleme tabi tutulan "Toplam Bit Sayısı" bulunur.
- HM değerini bulmak için "Farklı Bit Sayısı", "Toplam Bit Sayısı" na bölünür.

HM değerinin bulunması aşamasında bilinmeyen iris ve veri tabanından alınan karşılaştırılacak irisin gürültülü verilerinin hesaba katılmaması da sağlanmaktadır. Her iki iris için ilgili gürültü maskesi tersi değeri 3. adımda gösterildiği gibi AND işlemine tabi tutulur. Eğer maske değerleri iki iristen birinde dahi '1' ise yani ilgili bitte de gürültü varsa AND işleminin sonucu da '1' olacaktır. Daha sonra 6. adımda anlatıldığı gibi bu sonuç işleme alınır. Yani sonuç olarak '0' elde edilir. '0' değeri de yine 6. adımda anlatıldığı gibi

iris şablonuyla AND işlemine tabi tutulur. Burada sonuç ‘0’ olacaktır. Bunun anlamı eğer her iki iristen birinde işleme tabi tutulan pikselde yer alan bir gürültü verisi mevcutsa şablondaki verinin bir önemi olmadığı, yani işleme tabi tutulmayacağıdır. Eğer iki iriste de, işleme tabi tutulan piksel verisi gürültü içermiyorsa, bu sefer iris şablon verileriyle yapılacak olan işlemin önemi olur. 1. adımda anlatıldığı üzere her iki irisin şablon bit matrislerinin aynı elemanları XOR işlemine tabi tutulmaktadır. XOR işlemi daha önce de bahsedildiği gibi giriş olan her iki bit verisinin aynı olması durumunda ‘0’ çıkışı üretmektedir. Eğer her iki iris şablonunun aynı pozisyondaki bitleri aynıysa XOR işleminin çıkışı ‘0’ olacaktır bunun aksi durumda ise yani sonucun ‘1’ olduğu durumda “Farklı Bit Sayısı” isimli değişkenin değeri bir arttırılır. Çünkü bu durumda iki irisin ilgili bitlerinin birbirinden farklı oldukları tespit edilmiş olur. Her iki iris için ilgili bitte gürültü olduğu durumda, bu bit için işlem yapılmayacağından bahsedilmişti. İşlem yapılmayan bitlerin sayısı “Maske Biti Sayısı” isimli bir değişkende tutulur. 10. adımda bahsedildiği gibi iris şablonunda bulunan tüm bitlerin sayısından işlem yapılmayan bitlerin sayısı çıkarılarak, işleme tabi tutulan bitlerin sayısı bulunur. Bu sayı “Toplam Bit Sayısı” isimli bir değişkene atanır. Sonuç olarak Hamming Mesafesi değerini bulmak için Farklı Bit Sayısı”, “Toplam Bit Sayısı” na bölünür. Bu işlemden anlaşılacağı üzere HM değeri aslında iki bit verisinin birbirinden farklılığını ifade eden bir uzaklık değeridir. O yüzden benzerlik kıyaslaması yapılırken HM değerinin en düşük olanı seçilerek eşleştirme gerçekleştirilmelidir. En düşük HM değerini kullanmak birbirine en benzer iki irisin bulunmasını sağlayacaktır. Teorik olarak bu değerin 0 olması gerekmektedir. Fakat pratikte iris resimlerinde yer alan gürültüler nedeniyle bu değer sağlanamamaktadır. Bu teze konu olan uygulamada gürültünün neden olduğu yanlış eşleştirmelerin önüne geçmek amacıyla zaten veri tabanında yer almakta olan bir iris görüntüsü tekrardan Libor Masek’in paylaşıma açmış olduğu “açık kaynak kodlu iris tanıma” sistemine giriş olarak verilmiş ve çıkan şablon ve maske verisi ayrı bir veri tabanında bilinmeyen iris olarak tutulmuştur.

Hamming Mesafesi algoritmasının sıralı kodunun performansını değerlendirmek amaçlı olarak sadece “HM Değerinin Bulunması” algoritmasında anlatılan kısım ele alınmıştır. İris verilerinin veri tabanından çekilmesi performans değerlendirmesine tabi tutulmamıştır. Zaten bu kısım paralelleştirilmiş olan kodda aynı şekilde kullanılmaktadır. Ayrıca veri tabanındaki iris verilerinin alınması işlemi irislerin tek tek karşılaştırılması işlemiyle karşılaştırıldığında oldukça kısa (yüzde birine yakın) bir zaman tutmaktadır. Bu nedenle hem sıralı kodda hem de paralel kodda sadece iris verilerinin karşılaştırılmasının yapıldığı

kısımların tuttuğu süreler performans değerlendirilmesinde kullanılmıştır. Bu süreler tezin ilerleyen kısımlarında detaylı olarak incelenmiştir.

5.3. Hamming Mesafesi Algoritmasının Paralel Kod Şeklinde Yazılması

Hamming Mesafesi algoritması paralelleştirilirken kullanılabilecek programlama dili alternatifleri gözden geçirilmiştir. Oldukça yaygın olarak kullanılan ve algoritmaların, üzerinde çalıştığı donanımların cihaz özelliklerinden faydalanarak hızlandırılmasını sağlayan CUDA ve OpenCL dilleri değerlendirilmiştir. CUDA dili C temelli olup, yalnızca Nvidia GPU ları üzerinde çalışan bununla birlikte bu GPU'larda algoritmaların paralelleştirerek daha hızlı çalışmasını sağlayan bir dildir. OpenCL ise yine CUDA ya benzer şekilde C tabanlı olmakla beraber herhangi bir üretici firma ayırt etmeksizin tüm GPU'larda, destek veren CPU, FPGA ve DSP kartlarında çalışabilen ve bu kartlardan oluşan sistemlerde istenilen cihazlar üzerinde sırasıyla da çalışabilen bir özellik taşır. Bu nedenle çalışmamızda kodun heterojen sistemlere uygunluğunun sağlanması amacıyla paralelleştirme işleminin OpenCL kullanılarak yapılmasına karar verilmiştir.

Heterojen sistemlerde tüm sistemi yöneten bir sağlayıcı cihaz ve bu cihazın kontrol ettiği birçok cihaz olabilir. Genelde bu sağlayıcı cihaz bir CPU'dur ve kendisine bağlı olan cihazlara (GPU, FPGA ve DSP kartları gibi) OpenCL dili kullanılarak yazılmış olan kod parçalarını gönderir ve bu cihazlardan işlem sonuçlarını okur. Bu nedenle OpenCL kullanılarak bir algoritmanın paralelleştirilmesi, hem sağlayıcı da hem de sağlayıcıya bağlı olarak komutları gerçekleştiren cihazlarda koşacak olan kodların yazılmasıyla mümkün olur. CPU da koşacak olan kod (host code / sağlayıcı kodu) C++ programlama dili kullanılarak yazılır. Fakat projeye OpenCL'i kullanmaya destek sağlayan yazılım geliştirme kiti eklenir. Böylece OpenCL'e özgü olan fonksiyonların ilgili kütüphanelerden çağrılarak kullanılması sağlanır. Cihazlarda çalışacak olan kod ise OpenCL ile yazılan koddur. Bu kod ise OpenCL in kendine özgü bazı fonksiyonlarını içerir, bununla beraber C tabanlı olduğu için C programlama dilindeki birçok fonksiyon da kullanılabilir.

Sağlayıcı kodu (Host Code)

C++ programlama dilinde yazılan, CPU'nun kendisine çeşitli arayüzlerle bağlı bulunan cihazları kontrol etmesini, cihazlar arasında görev dağılımı yapmasını sağlayan koddur.

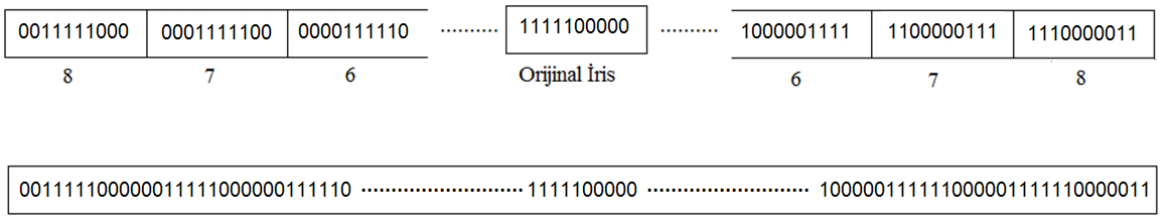
Bunun yanında yüksek hızlı hesaplama gerektirmeyen işlemler de CPU üzerinde gerçekleştirilmektedir. Bu teze konu olan uygulamada sağlayıcı kod, iris bit veri tabanından iris şablon ve maske verilerinin okunması, bu verilerin gönderileceği cihazlara uygun şekilde hazırlanması, OpenCL kodunun çalışacağı cihazlarda işlem yapılacak veriler için hafıza bölgelerinin ayarlanması ve sağlayıcı cihaza (CPU) bağlı olan cihazlardan gelen sonuçlara göre eşleşen irislerin belirlenmesi işlemlerini gerçekleştirmektedir. Sağlayıcı kodun gerçekleştirdiği işlemler aşağıda ayrıntılı olarak gösterilmiştir.

Sağlayıcı kod algoritması

- Bilinmeyen iris verisi okunur.
- Bilinmeyen iris verisinin matris şeklindeki şablon ve maske verilerini tek boyutlu dizilere çevrilir.
- Tüm iris veri tabanı okunur.
- Tüm iris veri tabanındaki irislerin matris şeklindeki şablon ve maske verileri tek boyutlu dizilere çevrilir.
- İris veri tabanından tüm irislere ait şablon verileri art arda eklenerek iris şablonları için tek boyutlu bir dizi oluşturulur.
- İris veri tabanından tüm irislere ait maske verileri art arda eklenerek iris maskeleri için tek boyutlu bir dizi oluşturulur.
- Bilinmeyen iris şablonu 8 kere sola kaydırılır. Her kaydırışta ortaya çıkan yeni şablon art arda eklenerek tek bir dizi oluşturulur. Aynı işlemler iris maske verileri için de tekrarlanır.
- Bilinmeyen iris in orijinal şablonu daha önce 7. Adımda elde edilen diziye eklenir. Aynı işlem orijinal maske verisi için de yapılır.
- Bilinmeyen iris şablon ve maske verileri 8 kere sağa kaydırılır. Her kaydırışta ortaya çıkan yeni şablon ve maskeler orijinal irisin şablon ve maskesinin ardına eklenir.
- Sağlayıcı cihaza bağlı olan cihaz tipi ayarlanır. (GPU, FPGA gibi)
- ”.cl” uzantısı kullanılarak oluşturulmuş olan OpenCL kodu okunur.
- Okunan OpenCL kodu ilgili cihaz üzerinde yapılandırılır.
- Veri tabanındaki tüm irisler için hazırlanan tek boyutlu şablon ve maske verileri için hafıza bölgeleri oluşturulur.
- Bilinmeyen iris ile sağa ve sola 8 er kez kaydırılmasından elde edilen toplamda 17 iris şablon ve maskeleri için hafıza bölgesi oluşturulur.
- Sağlayıcı cihaza bağlı olan işlem gücü yüksek cihazlarda yapılacak işlemlerin sonucunda bulunacak olan “Maske Biti Sayısı” ve “Farklı Bit Sayısı ” değişkenlerinin yazılacağı iki farklı bellek bölgesi oluşturulur.
- İş grubu ve toplam iş nesnesi sayılarının ataması yapılır.
- Belirlenen iş grubu ve iş nesnesi sayılarına göre cihaz üzerinde OpenCL kodu çalıştırılır.
- İşlem sonucunda çıktı olarak ortaya çıkan “Maske Biti Sayısı” ve “Farklı Bit Sayısı ” değişkenlerini tutan bellek bölgeleri okunur.
- Bilinmeyen iris ile sağa ve sola 8 er kez kaydırılmasından ortaya çıkan her iris için işleme tabi tutulan “Toplam Bit Sayısı” bulunur.

- Bilinmeyen iris ile sağa ve sola 8 er kez kaydırılmasından ortaya çıkan her iris için "Farklı Bit Sayısı ", "Toplam Bit Sayısı" na bölünerek HM değeri hesaplanır.
- Veri tabanındaki her iris için 19 ve 20. Adımlar tekrarlanır.

Heterojen hesaplamada performans açısından önemli faktörlerden birisi hafızaya erişim süresidir. Hafızaya erişimin en aza indirilmesi, verilerin tek seferde hedef cihaza gönderilmesi performans açısından kazanç sağlar. Bu nedenle iris verileri tek boyutlu diziler haline getirilerek hafızaya gönderilmiştir. Matris olarak hedef cihaza verileri göndermek için satır satır ya da sütun sütun verilerin gönderilmesi gerekeceğinden pek çok kez hafızaya erişim gerekecek ve hafızaya erişim işlemi gecikmelere sebep olarak performansı olumsuz yönde etkileyecektir. 2 ve 4. adımlarda iris verilerinin tek boyutlu dizilere çevrilmesi işlemi bu nedenle gerçekleştirilmektedir. Yine aynı şekilde 7, 8 ve 9. adımlarda sağa ve sola kaydırılan iris verilerinin art arda eklenmesi de tek aşamada tüm kaydırılmış verilerin ve orijinal verinin hedef cihaza tek aşamada iletilmesini sağlamak içindir. Burada ortaya çıkan tek boyutlu dizi örnek olarak 9 bitlik bir iris için Şekil 5.1 de gösterilmiştir. Şekil 5.1 de kaydırılmış irislerin altında kaç kez kaydırıldıkları yazmaktadır. Ortada orijinal iris verisi (şablon ya da maske) de yer almaktadır. Normalde her kaydırma tek boyutlu bir dizi oluştururken tüm kaydırılmış iris dizileri ve orijinal iris verisi birbirine eklenerek tek bir dizi elde edilmiştir.



Şekil 5.1. İris verilerinden tek boyutlu dizi elde edilmesi

Böylece toplamda bilinmeyen irisin kaydırılmış şekilleriyle 17 adet iris elde edilmiş ve tek bir hamlede tüm veriler hedef cihazın hafızasına yazılmıştır.

Hedef cihaz kodu (Device Code)

Sağlayıcı cihaz tarafından ".cl" uzantılı dosyanın okunup hedef cihaza bu dosyanın içindeki kodların gönderildiğinden daha önce bahsedilmişti. OpenCL ile yazılmış olan bu dosya hedef cihazlar üzerinde koşacak olan "hedef cihaz kodu" dur. Hedef cihaz kodu aynı

zamanda çekirdek (kernel) kodu olarak da bilinir. Aslında çekirdek kodu OpenCL deki bir fonksiyon olarak düşünülebilir. Bu fonksiyon tüm girişler için aynı işlemi yapmaktadır. Aynı işlemin aynı anda yapılmasını sağladığı için paralelleşme ve dolayısıyla hızlanma sağlamaktadır. Hedef cihaz kodunu oluşturan algoritma aşağıda detaylı olarak anlatılmıştır.

Hedef cihaz kod algoritması

- Get_global_id(0) komutuyla her iş nesnesini kendine özgü olan numarası alınır.
- Sağlayıcı kodunda hazırlanmış olan bir boyutlu iris şablonundaki bit ile bilinmeyen irisin şablon biti XOR işlemine tabi tutulur. Sonuç "şablon" değişkenine atanır.
- Sağlayıcı kodunda hazırlanmış olan bir boyutlu iris maskesindeki bit ile bilinmeyen irisin maske biti tersleri AND işlemine tabi tutulur. Sonuç "Maske" değişkenine atanır.
- Eğer "Maske" değişkeninin tersi '1' ise "Maske Biti Sayısı" dizisindeki değer veri tabanındaki ilgili iris için bir arttırılır.
- "Maske" değişkeninin değeri ile "şablon" değişkeni AND işlemine tabi tutulur. Sonuç yeniden "Şablon " değişkenine atanır.
- Eğer "Şablon" değişkeninin değeri '1' ise veri tabanındaki ilgili iris için "Farklı Bit Sayısı" dizisindeki ilgili eleman bir arttırılır.
- İlk 6 işlem tüm kaydırılmış irisler için tekrarlanır.

Hedef cihazda çalışacak olan kodda verilerin organizasyonuna hâkim olmak ve hangi iris için işlem yapıldığını bulabilmek oldukça zor olabilmektedir. İş nesnesinin kendine ait değişmeyen bir tanım numarası vardır. Bu teze konu olan uygulamada veri tabanında yer alan iris şablon bitlerinin toplam sayısı toplam iş nesnesi sayısı olarak kullanıldı. Böylece get_global_id(0) fonksiyonu kullanıldığında iş nesnesinin kendine özgü numarası alınırken aynı zamanda veri tabanında hangi bit ile işlem yapıldığı bilgisi de alınmış olur. Bir irisi oluşturan bit sayısı bilindiği için mod işlemi kullanılarak hangi irise ait bit ile işlem yapıldığı belirlenir ve böylece hangi bit için işlem yapıldığı belirlenir. Her ikili karşılaştırma için "Maske Biti Sayısı" ve "Farklı Bit Sayısı" dizilerinde mevcut bulunan bir eleman bulunmaktadır. 4. adımda anlatıldığı üzere "Maske" değişkeni tersi '1' değerini aldığında üzerinde işlem yapılan iris için "Maske Biti Sayısı" dizisindeki o iris için ayrılmış değer bir arttırılır. Aynı şekilde 6. adımda adı geçen "Şablon" değişkeninin değeri '1' olduğunda bu sefer "Farklı Bit Sayısı" dizisinde üzerinde işlem yapılan iris için ayrılmış elemanın değeri bir arttırılmaktadır. Genel olarak hedef cihaz kodunda dikkat edilmesi gereken noktalardan birisi ise aynı hafıza bölgelerini kullanan işlemlerin yer aldığı kodlarda hafıza bölgesine aynı anda bir iş nesnesinin ulaşmasını sağlamaktır. Aynı

anda birden fazla iş nesnesi bir hafıza bölgesini kullanmaya çalışıyorsa bu hafıza bölgesinde yer alan verilerin güncelliğinden şüphe duyulacaktır. Örneğin bu teze konu olan uygulamada "Maske Biti Sayısı" ve "Farklı Bit Sayısı" dizilerinde her ikili karşılaştırma sonucunun tutulduğu bir eleman mevcuttur. Burada yapılan işlem ise bu dizi elemanlarının bir arttırılması şeklindedir. Dolayısıyla daha önce dizide bulunan ilgili elemanın değeri okunmalı ve bir arttırılmalıdır. Fakat aynı anda işlem yapan iki iş nesnesi bu değeri aynı anda okuyup bir arttırırsa normalde iki artması gereken değerin bir arttırıldığı görülür. Yani bir örnek ile açıklamak gerekirse, "Maske Biti Sayısı" dizisinin 5. iris için ayrılmış olan elemanının değeri 19 olsun. Aynı anda 5. iris için hesaplama yapan iki iş nesnesi 19 değerini alıp kod akışı doğrultusunda bir arttıracak olsa yeni değer iki iş nesnesi için de 20 olacaktır. Fakat aslında iki farklı bit için işlem yapıldığı için 19 sayısının iki kere arttırılarak 21 değerinin elde edilmesi gerekmektedir. Bunu sağlamak için OpenCL de "barrier" isimli komut kullanılmaktadır. "Barrier" komutu, yazıldığı satırdan bir öncesindeki satırda yer alan işlem iş nesneleri tarafından eş zamanlı olarak gerçekleştirilmez. İş nesneleri bu kodun yazıldığı satırın bir önceki satır için birbirlerini bekleyerek bir senkronizasyon sağlarlar. Böylece en güncel değerin işlemlerde kullanılması ve doğru bir sonuç elde edilmesi sağlanmış olur. Hedef cihazda koşmakta olan OpenCL kodu aşağıda verilmektedir.

```
#define KOLON 256
#define SATIR 8
#define SHIFT_NO 8
#define IRIS_SAYISI 1329
__kernel void hamming_distance2( __global int* database_temp, __global int* database_mask, __global int* iris_temp, __global int* iris_mask, __global int* total_diff_bits, __global int* num_of_mask_bits )
{
int index=get_global_id(0);
int temp1=0;
int mask1=0;
int num_mask=0;
int num_tot_bit=0;

for(int i=0;i< 2*SHIFT_NO+1;i++)
{
temp1=
iris_temp[(index%(SATIR*KOLON))+(i*KOLON*SATIR)]^database_temp[index];
mask1=
!iris_mask[(index%(SATIR*KOLON))+(i*SATIR*KOLON)]*!database_mask[index];
        if(!mask1==1){
        num_mask=num_of_mask_bits[(index/(SATIR*KOLON))+i*IRIS_SAYISI]+1;
```

```
        num_of_mask_bits[(index/(SATIR*KOLON))+i*IRIS_SAYISI]=num_mask;
        barrier(CLK_GLOBAL_MEM_FENCE|CLK_LOCAL_MEM_FENCE);
        }
        else
        {
           temp1=temp1&&mask1;
           if(temp1==1){
              num_tot_bit=total_diff_bits[(index/(SATIR*KOLON))+i*IRIS_SAYISI]+1;
              total_diff_bits[(index/(SATIR*KOLON))+i*IRIS_SAYISI]=num_tot_bit;
              barrier(CLK_GLOBAL_MEM_FENCE|CLK_LOCAL_MEM_FENCE);
                        }
         }
      }
  }
```

Yukarıda verilen kodda yer alan “for” döngüsü ile sağa ve sola kaydırılmış irisler için de tüm işlemler yapılmaktadır. Hedef cihazda gerçekleştirilen tüm işlemlerin sonuçları sağlayıcı cihaz kodunda oluşturulmuş olan hafıza bölgesinden okunarak tüm irisler için HM hesabı gerçekleştirilmiş olur. Daha sonra HM değerleri karşılaştırılarak en küçük HM yi sağlayan irislerin eşleştirmesi gerçekleştirilir.

5.4. Performans Karşılaştırması

Heterojen mimari, hem CPU gibi kullanım alanı geniş, hem de GPU ve FPGA gibi özelleşmiş yüksek işlem gücü ve düşük güç tüketimi sağlayan cihazları bir arada barındırmasından dolayı son dönemlerde oldukça tercih edilmektedir. Sağlamış olduğu performans kazançlarından hareketle karmaşık problemlerin çözümü ve büyük veriler içeren işlemlerin gerçeklenmesinde heterojen mimarinin önümüzdeki yıllarda daha da yaygın şekilde kullanılacağı öngörülmektedir. Bu teze konu olan çalışmada performans parametresi olarak zaman alınmıştır. Seri olarak yazılmış “Hamming Mesafesi” algoritması için sadece HM hesabının yapıldığı fonksiyonun tüm irisler için bitirilme zamanı hesaplanarak konsol ekranına yazdırılmıştır. Bulunan HM değerlerinin karşılaştırılması işleminin aldığı zaman işleme katılmamıştır. Paralel olarak yazılmış ve hedef cihaz üzerinde koşacak olan kodun çalışma süresi değerlendirilirken aynı şekilde sadece hesaplamaların yapıldığı kısım yani sadece OpenCL ile yazılmış kısım ele alınmıştır. Sağlayıcı kodunda yer alan veri tabanı organizasyonu ve karşılaştırmanın gerçekleştirilmesi performans karşılaştırması için zaman parametresi olarak kullanılmamıştır. Sonuç itibariyle her iki kod için de aynı işlemi yapan kısımların

karşılaştırılması yapılmış ve elde edilen sonuçlar konsol ekranına basılarak görüntülenmiştir.

C++ kullanılarak seri olarak yazılmış olan kod yalnızca CPU da koşturulmuştur. Paralel olarak yazılmış olan kodda ise sağlayıcı cihaz kodu CPU üzerinde koşturulurken OpenCL ile yazılmış olan hem CPU hem de GPU üzerinde koşturulmuştur. Böylece CPU ile GPU nun paralel performansları karşılaştırılırken aynı zamanda paralel olarak yazılmış olan kod ile seri olarak yazılmış olan kod da birbirleriyle performans açısından karşılaştırılmıştır.

Ayrıca iş grubu sayısının performans üzerine etkisi incelenmiştir. İş grubu sayısının optimum olarak belirlenmesi amacıyla hem GPU'nun hem de CPU'nun desteklediği en yüksek iş nesne sayıları kullanılarak hesaplama gerçekleştirilmiştir. Bu teze konu olan çalışmada CPU olarak Intel Xeon 5650 işlemcisi ve GPU olarak ise Nvidia GT430 kullanılmıştır. Seri kod Xeon 5650 işlemcisinde koşturulurken, paralel olarak yazılmış kod da sağlayıcı kodu Xeon 5650 işlemcisinde, hedef cihaz kodu ise hem Xeon 5650 işlemcisinde hem de Nvidia GT430 grafik işlemcisinde koşturulmuştur.

İris boyutları kullanılan "açık kaynak kodlu iris tanıma sistemi" algoritmasında kullanılan Rubber Sheet modeline göre iris resimlerine bağlı olmak kaydıyla uygun şekilde belirlenebilmektedir. Buna göre kullanılan iris resim veri tabanına uygun şekilde 8x256 bitlik ve 16x480 bitlik olmak üzere iki adet iris boyutu belirlenmiş ve bu boyutlara göre elde edilen iris bit veri tabanı üzerinde işlem yapılmıştır. Yapılan çalışmada bilinmeyen irisin kendisi, 8 kereye kadar sağa kaydırılmış hali (bir bit olarak) ve 8 kereye kadar sola kaydırılmış (bir bit olarak) irislerden oluşan toplamda 17 iris veri tabanındaki bütün irisler ile teker teker karşılaştırılmıştır. Toplam karşılaştırma sayısı Çizelge 5.1 de gösterilmiştir.

Çizelge 5.1. Toplam karşılaştırma sayısı

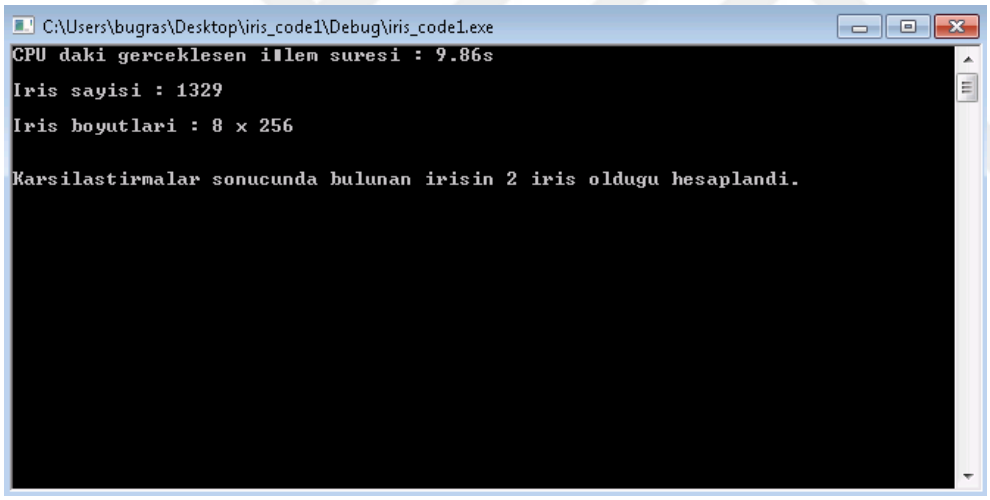
Veritabanındaki toplam iris sayısı	Sağa Kaydırma	Sola Kaydırma	Toplam Karşılaştırma
1329	8	8	22593

Seçilen algoritmanın paralelleştirilmesi aşamasında tüm işlemler tek tek bitler halinde gerçekleştirilmektedir. Bu nedenle toplamda bit sayısı kadar iş nesnesi mevcuttur. Bu teze

konu olan uygulamada her iş grubuna eşit sayıda iş nesnesi gelecek şekilde bir tasarım gerçekleştirilmiştir. Cihazların tasarımında bir iş grubunun ne kadar iş nesnesi barındırabileceği belirlenmiştir. Bu çalışmada hem GPU hem de CPU iş gruplarında 1024 iş nesnesine kadar destek vermektedir. 8x256 bitlik irisler için 886 ve 1024 iş nesnesi, 16x480 bitlik irisler için ise 768 ve 960 iş nesneleri iş gruplarında kullanılmıştır. Tüm bit sayısı sırayla bu sayılara bölünerek iş grubu sayıları bulunabilir. 8x256 bitlik irisler için 3072 ve 2658 adet iş grupları, 16x480 bitlik irisler için ise 13290 ve 10632 adet iş grubu kullanılmıştır.

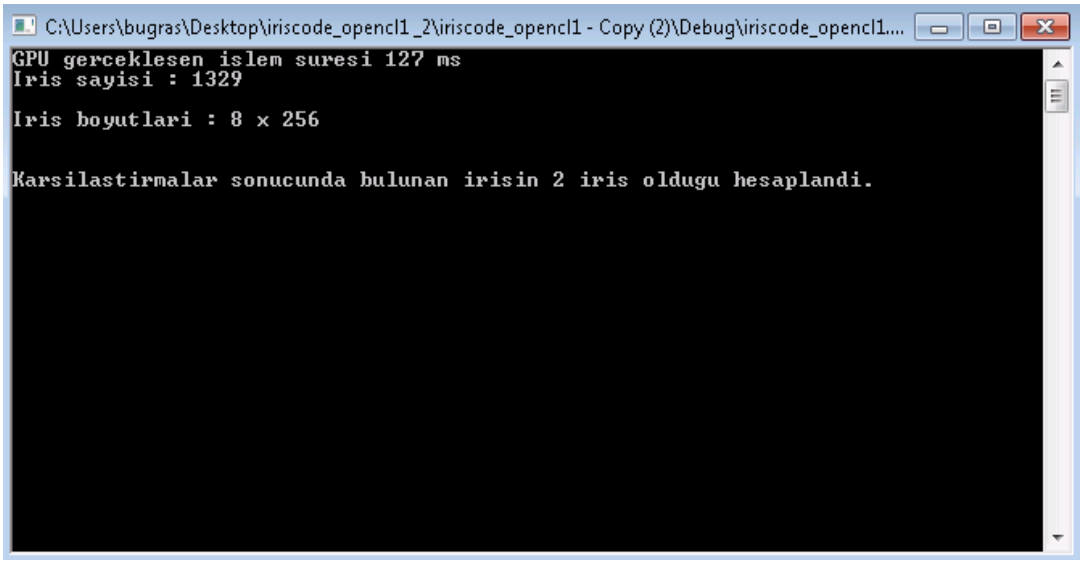
8x256 bitlik iris boyutları için aşağıdaki sonuçlar elde edilmiştir.

Resim 5.6 da seri olarak yazılmış olan “Hamming Mesafesi” algoritmasının Xeon 5650 işlemcisinde gerçekleştirdiği işlem zamanı konsol ekranında görünmektedir.



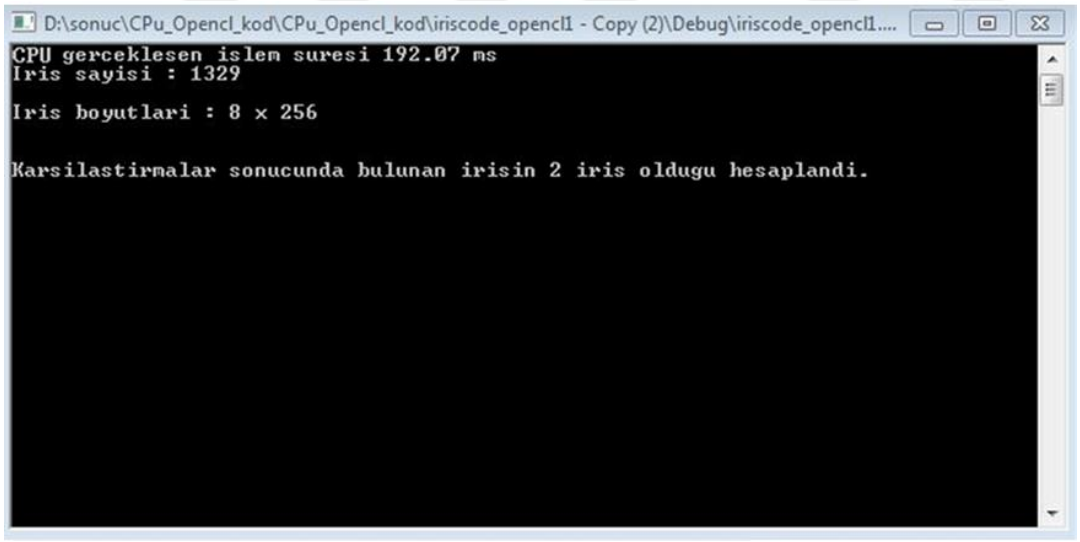
Resim 5.6. Seri olarak kodlanmış algoritmanın konsol çıktısı

Resim 5.7 de paralel olarak 3072 iş grubu kullanacak şekilde yazılmış olan “Hamming Mesafesi” algoritmasının Nvidia GT430 grafik işlemcisinde gerçekleştirdiği işlem zamanı konsol ekranında görünmektedir.



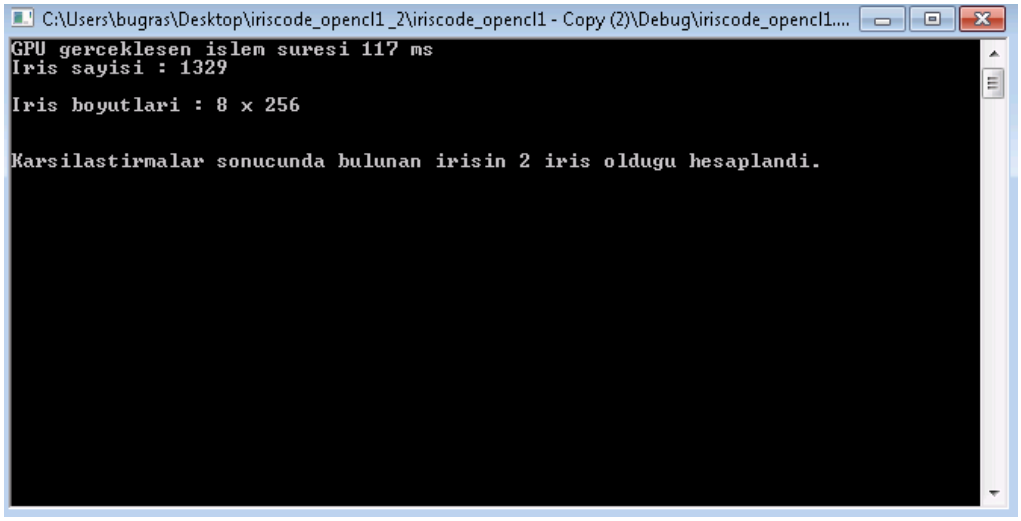
Resim 5.7. Paralel olarak kodlanmış algoritmanın konsol çıktısı (GPU)

Resim 5.8 da paralel olarak 3072 iş grubu kullanacak şekilde yazılmış olan "Hamming Mesafesi" algoritmasının Xeon 5650 işlemcisinde gerçekleştirdiği işlem zamanı konsol ekranında görünmektedir.



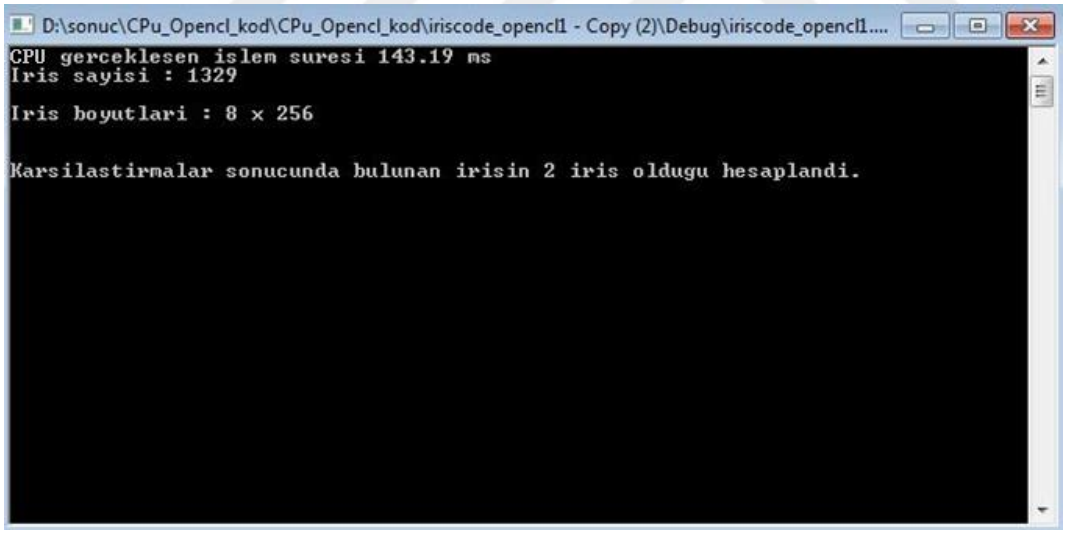
Resim 5.8. Paralel olarak kodlanmış algoritmanın konsol çıktısı (CPU)

Resim 5.9 da paralel olarak 2658 iş grubu kullanacak şekilde yazılmış olan "Hamming Mesafesi" algoritmasının Nvidia GT430 grafik işlemcisinde gerçekleştirdiği işlem zamanı konsol ekranında görünmektedir.



Resim 5.9. Paralel olarak kodlanmış algoritmanın konsol çıktısı (GPU-2)

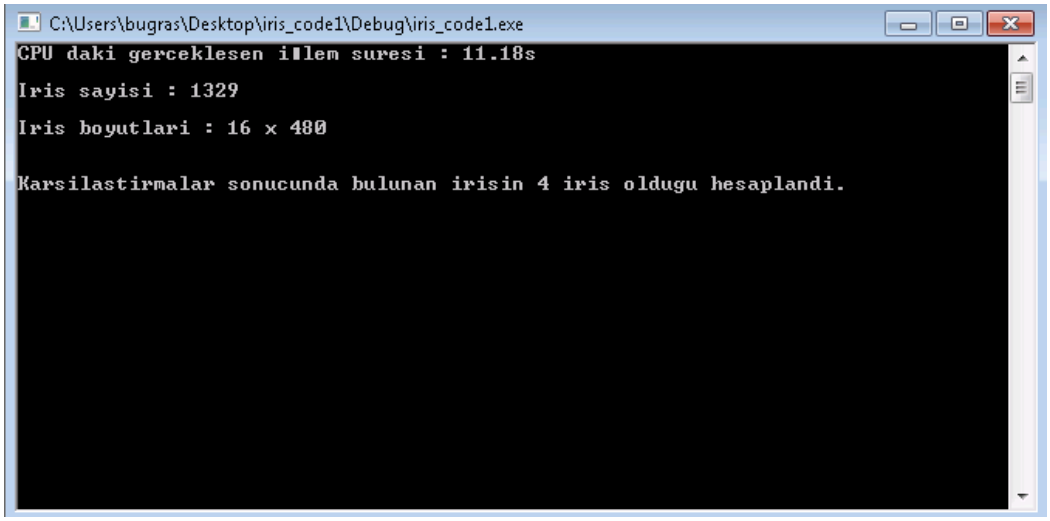
Resim 5.10 de paralel olarak 2658 iş grubu kullanacak şekilde yazılmış olan "Hamming Mesafesi" algoritmasının Xeon 5650 işlemcisinde gerçekleştirdiği işlem zamanı konsol ekranında görünmektedir.



Resim 5.10. Paralel olarak kodlanmış algoritmanın konsol çıktısı (CPU-2)

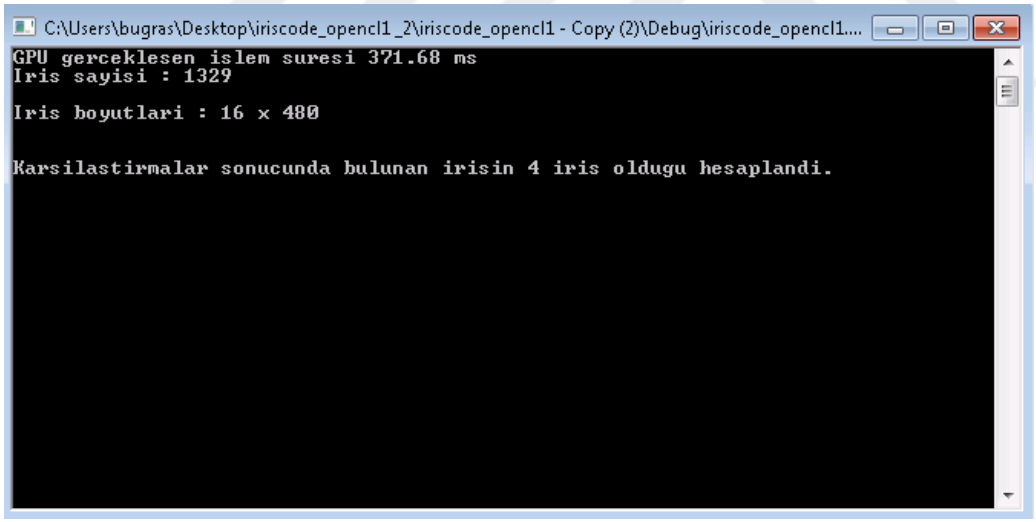
16x480 bitlik iris boyutları için aşağıdaki sonuçlar elde edilmiştir.

Resim 5.11 de seri olarak yazılmış olan "Hamming Mesafesi" algoritmasının Xeon 5650 işlemcisinde gerçekleştirdiği işlem zamanı konsol ekranında görünmektedir. 16x480 bitlik irisler için işlem süresinin 8x256 bitlik irislere göre daha fazla olması beklenmektedir. Nitekim yapılan uygulamada da sonuç tam olarak bu şekilde görülmüştür.



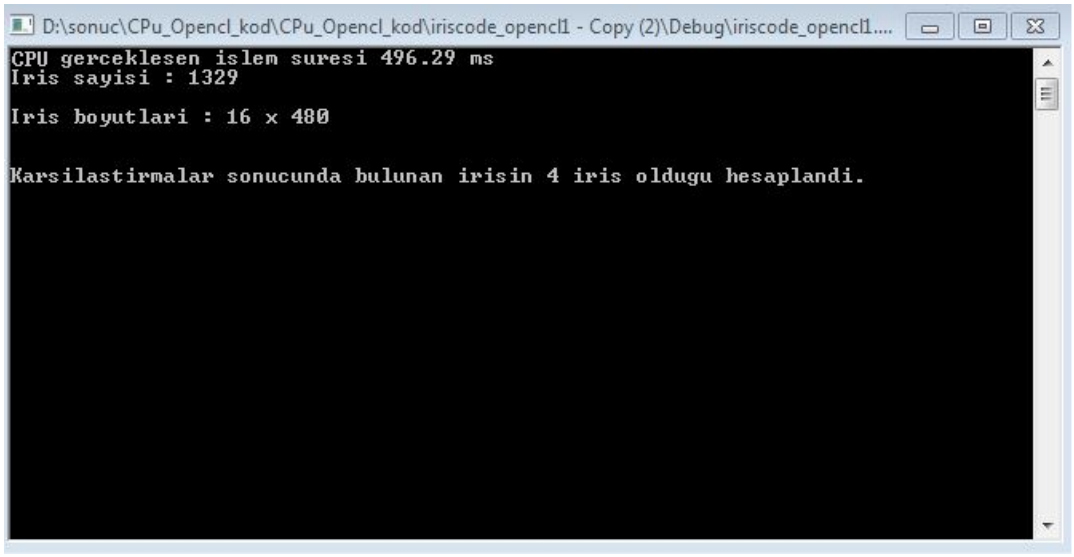
Resim 5.11. Seri olarak kodlanmış algoritmanın konsol çıktısı-2

Resim 5.12 de paralel olarak 13290 iş grubu kullanacak şekilde yazılmış olan "Hamming Mesafesi" algoritmasının Nvidia GT430 grafik işlemcisinde gerçekleştirdiği işlem zamanı konsol ekranında görünmektedir.



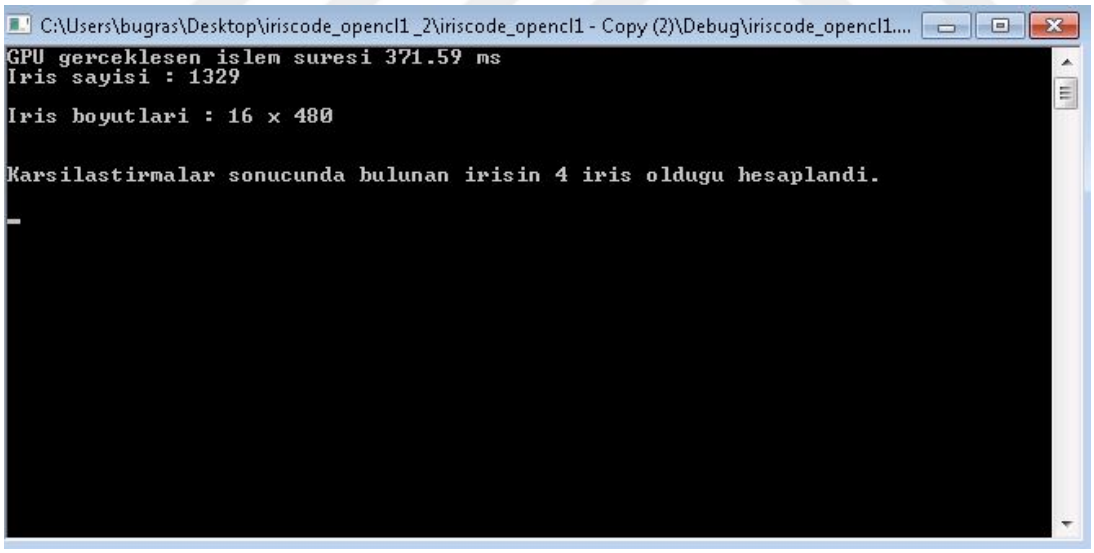
Resim 5.12. Paralel olarak kodlanmış algoritmanın konsol çıktısı (GPU-3)

Resim 5.13 de paralel olarak 13290 iş grubu kullanacak şekilde yazılmış olan "Hamming Mesafesi" algoritmasının Xeon 5650 işlemcisinde gerçekleştirdiği işlem zamanı konsol ekranında görünmektedir.



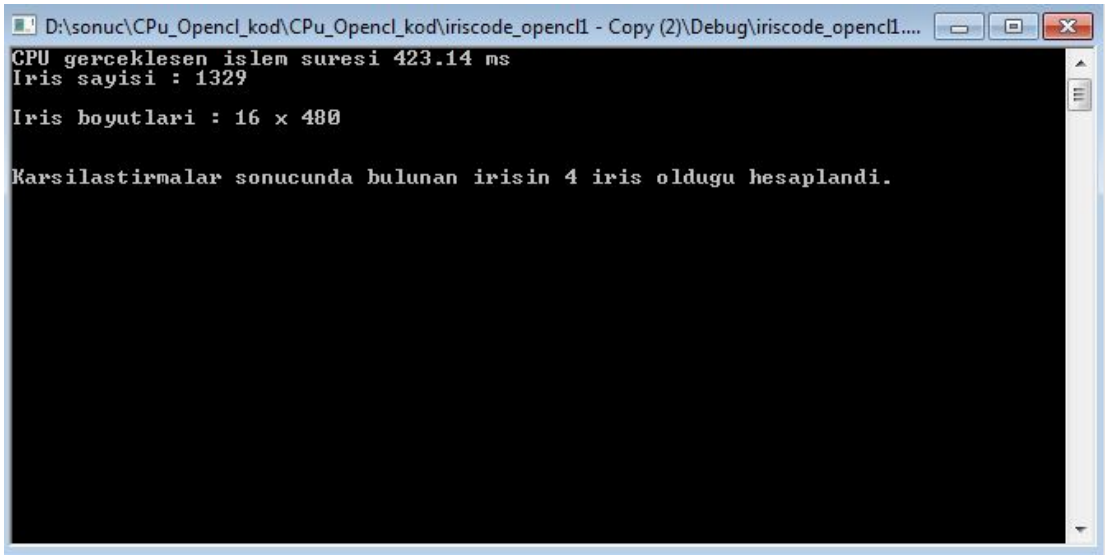
Resim 5.13. Paralel olarak kodlanmış algoritmanın konsol çıktısı (CPU-3)

Resim 5.14 de paralel olarak 10632 iş grubu kullanacak şekilde yazılmış olan "Hamming Mesafesi" algoritmasının Nvidia GT430 grafik işlemcisinde gerçekleştirdiği işlem zamanı konsol ekranında görünmektedir.



Resim 5.14. Paralel olarak kodlanmış algoritmanın konsol çıktısı (GPU-4)

Resim 5.15 de paralel olarak 10632 iş grubu kullanacak şekilde yazılmış olan "Hamming Mesafesi" algoritmasının Xeon 5650 işlemcisinde gerçekleştirdiği işlem zamanı konsol ekranında görünmektedir.



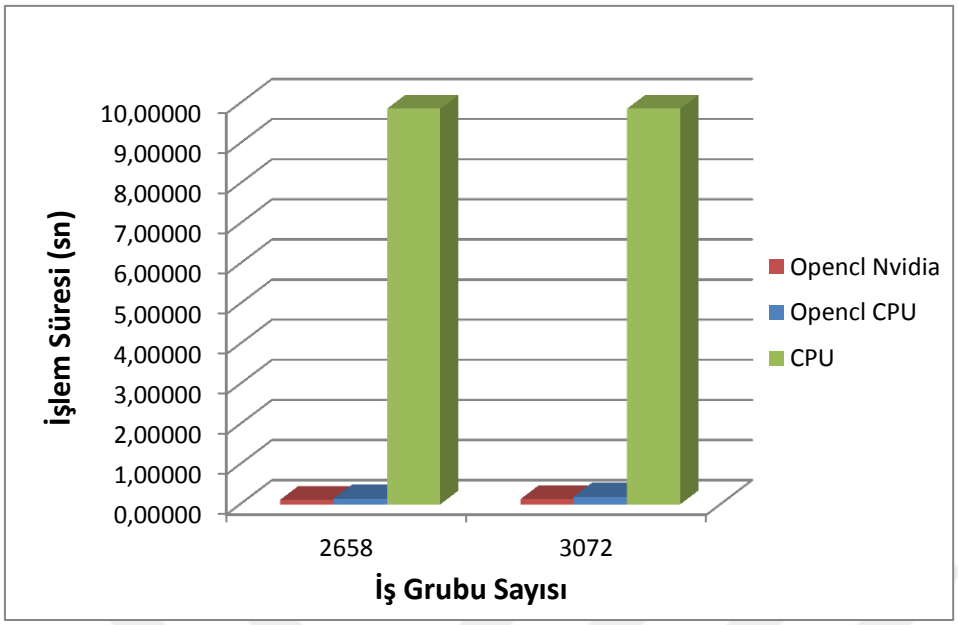
Resim 5.15. Paralel olarak kodlanmış algoritmanın konsol çıktısı (CPU-4)

Tüm bu veriler bir araya getirilerek Çizelge 5.2'de ayrıntılı olarak gösterilmiştir. İlgili çizelgede iris boyutları, iş grubu sayısına göre OpenCL ile kodlanmış paralel kodun GPU ve CPU üzerindeki işlem zamanları gösterilmiştir. Ayrıca seri olarak yazılmış olan "Hamming Mesafesi" algoritmasının işlem süresi de en sağdaki kolonda verilmiştir.

Çizelge 5.2. İşlem süresi (SN)

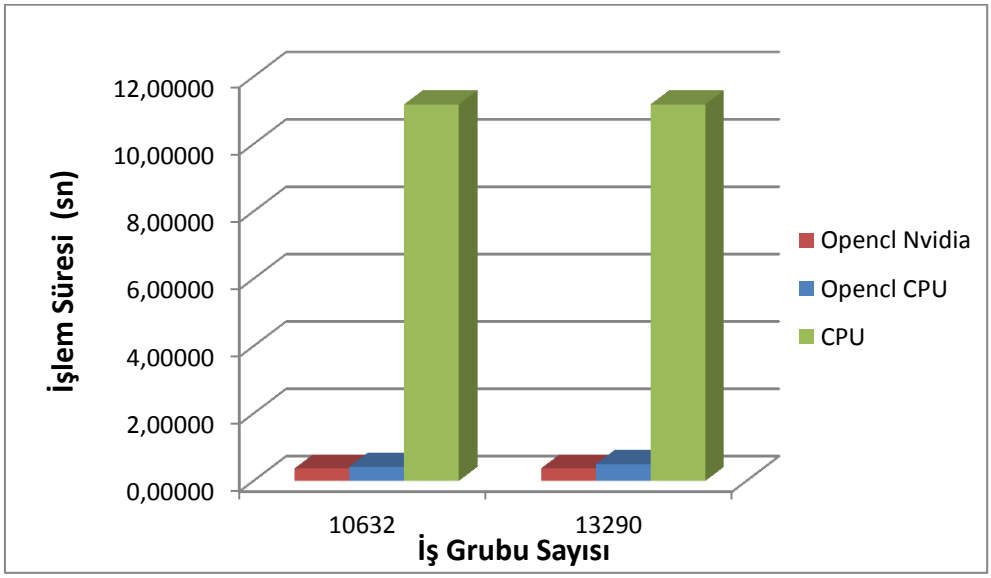
İris Boyutu	İş Grubu Sayısı	OpenCL Nvidia GPU (sn)	OpenCL Intel CPU (sn)	Seri Kod CPU (sn)
8x256	2658	0,11215	0,14319	9,86
8x256	3072	0,127	0,19207	9,86
16x480	10632	0,37159	0,42314	11,18
16x480	13290	0,37168	0,49629	11,18

İşlem süreleri incelendiğinde seri olarak yazılmış olan "Hamming Mesafesi" algoritmasının paralel kodlara göre beklenildiği gibi oldukça yavaş kaldığı gözlenmektedir. 8x256 boyutundaki irisler için işlem süresinin grafiksel olarak gösterimi Şekil 5.2 de verilmiştir.



Şekil 5.2. 8x256 boyutlarındaki iris için işlem süreleri

İlgili grafikten de görüleceği gibi OpenCL kullanılarak üretilen çözüm işlemci üzerinde çalışan seri olarak kodlanmış koddan çok daha az süre tutmaktadır.16x480 boyutundaki irisler için işlem süresinin grafiksel olarak gösterimi Şekil 5.3 de verilmiştir.



Şekil 5.3. 16x480 boyutlarındaki iris için işlem süreleri

16x480 boyutlarındaki iris için de sonuçlar 8x256 boyutlarındaki irisler için olana benzerdir. Bu boyutlar için farklı olan tek şey beklenen şekilde daha fazla bit içeren irisler için daha fazla işlem süresi harcanmasıdır. Dolayısıyla aynı şekilde algoritmanın

gerçekleştirilmesi için daha fazla iş grubu kullanılmıştır. Paralel kodlamada hız artışı seri kodun işlem gerçekleştirme süresinin paralel kodun işlem gerçekleştirme süresine oranı ile bulunur. Amdahl yasası olarak bilinen bu formül;

$$S = \frac{T_s}{T_p} \tag{5.1}$$

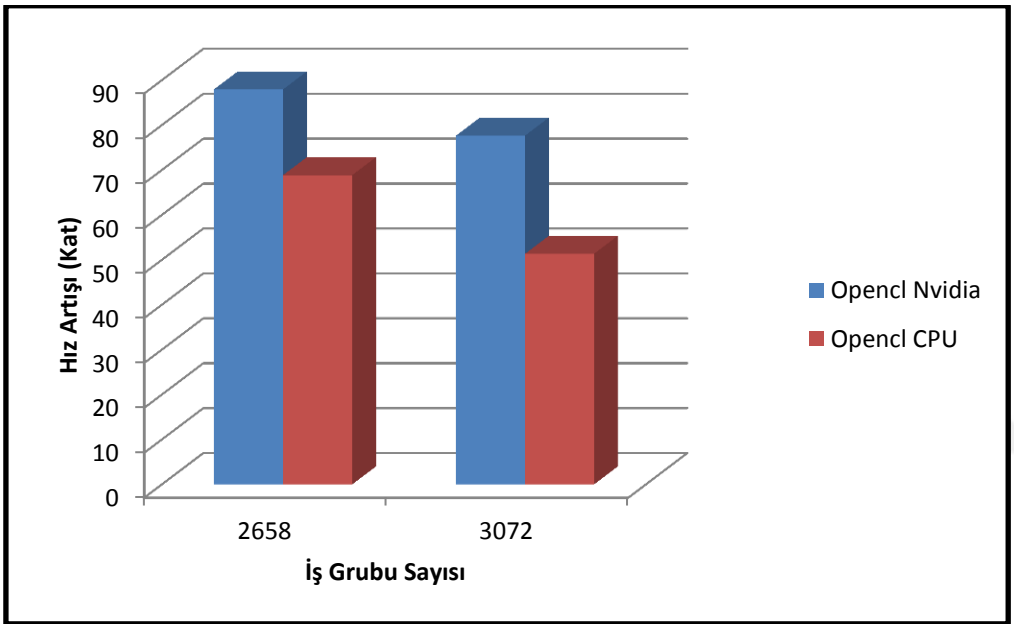
şeklinde yazılır. Burada S hız artışını, T_s seri kodun işlem süresini ve T_p ise paralel kodun işlem süresini ifade etmektedir [23]. Bu formül kullanılarak "Hamming Mesafesi" algoritmasının paralel olarak yazılmış kodunun Intel Xeon 5650 işlemcisi ve Nvidia GT430 grafik işlemcisi üzerindeki hız artışı hesaplanmış Çizelge 5.3'de gösterilmiştir.

Çizelge 5.3. Hız artışı (kat)

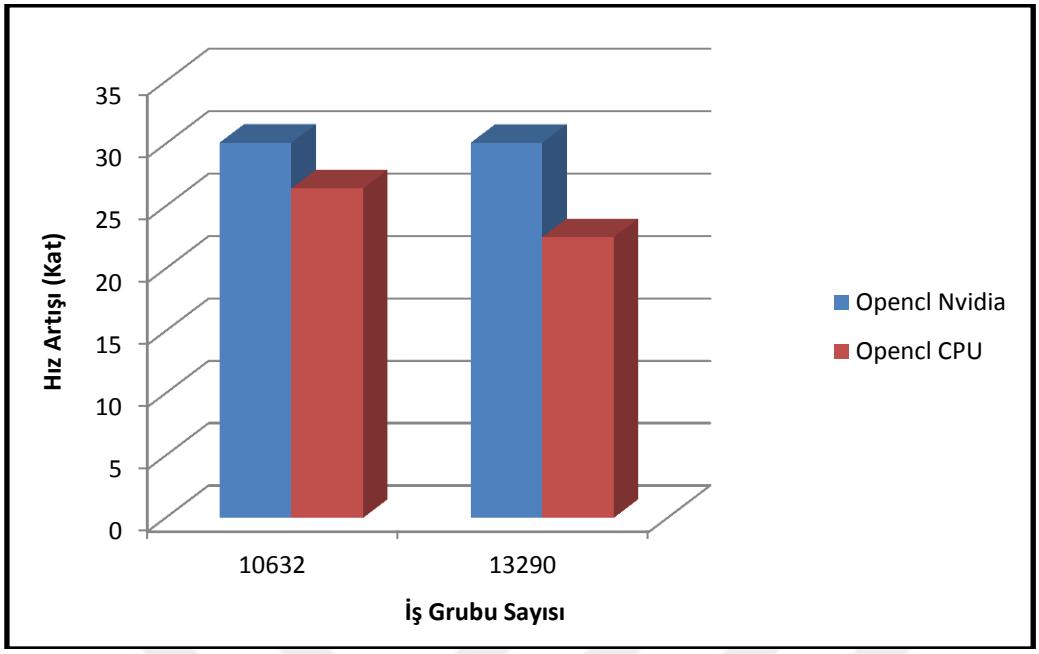
İris Boyutu	İş Grubu Sayısı	OpenCL Nvidia	OpenCL CPU
8x256	2658	87,9	68,85
8x256	3072	77,6	51,3
16x480	10632	30	26,42
16x480	13290	30	22,52

Hız artışı verileri incelendiğinde paralel kodda 88 kata kadar hız artışı olduğu gözlenmiştir. Ayrıca 8x256 boyutlarındaki irisler için iş grubu sayısı azaldığında artışın daha fazla olduğu sonucu da Çizelge 5.3'e göre yapılabilecek bir diğer yorumdur. Bu durum 16x480 boyutlarındaki irisler için kısmen geçerli olmuştur. GPU üzerindeki hız artışı daha az iş grubu kullanıldığında değişmemiştir. Burada GPU için hız artışında bir doyuma yaklaşıldığı düşünülebilir. GPU'nun toplam olarak desteklediği iş nesnesi sayısı sınırına yaklaşıldığı için performanstaki artış da belli bir doyuma ulaşmıştır. Nvidia GT430 grafik işlemcisi bir grupta maksimum olarak 1024 iş nesnesi barındırabilmektedir. Bu sınır 10632 iş grubu kullanıldığında tam olarak sağlanmaktadır. Bu nedenle GPU'nun performans açısından bu algoritma ve bu iris bit veri tabanı seti için doyuma girdiği anlaşılmaktadır. Daha da büyük bir veri tabanı kullanıldığında artışın daha az olacağı aşikârdır. Çünkü bir seferde tüm iş gruplarındaki iş nesneleri ile tüm bitler için işlem yapılamayacak ve dolayısıyla tüm iş gruplarındaki işlemler bittikten sonra tekrardan iş nesneleri iş

gruplarındaki yerlerini alarak kalan işlemi gerçekleştireceklerdir. Bu durum da performans artışının azalmasına neden olacaktır. Çizelge 5.3'de ifade edilmiş olan hız artışının grafiksel olarak gösterimi 8x256 boyutlu irisler için Şekil 5.4 de 16x480 boyutlu irisler için Şekil 5.5 de verilmiştir. Bu şekillerde Nvidia GT430 grafik işlemcisinin bu teze konu olan veri seti için Intel Xeon 5650 işlemcisinden daha başarılı olduğu görülmektedir. Başka veri kümelerinde Intel Xeon 5650 işlemcisinin daha hızlı olduğu durumlar da olabilir. Çok daha az sayıda iş nesnesine gereksinim duyan uygulamalarda işlemcilerin grafik işlemcilerden daha hızlı işlem yaptığı bilinmektedir.



Şekil 5.4. 8x256 boyutlu irisler için hız artışı



Şekil 5.5. 16x480 boyutlu irisler için hız artışı

Elde edilen sonuçlar doğrultusunda en az hız artışının gerçekleştiği durumda bile bu artışın 22 kat olduğu gözlenmiştir.

6. SONUÇ VE ÖNERİLER

İris tanıma sistemlerinin literatürde yerini almasıyla birlikte pek çok sayıda araştırma yapılmış ve her bir araştırma iris tanıma sistemlerinin gelişimine katkı sağlamıştır. İris tanıma sistemlerinin fonksiyonel olarak kullanılmaya başlamasından itibaren yapılan çalışmaların amacı daha çok performans geliştirmeye yönelik olmuştur. Bu tez çalışması da performans geliştirmeye yönelik çalışmalar arasında yer alabilir. Bununla birlikte literatürde yer alan çalışmalardan farkı ise heterojen mimari altyapısına uygun olarak kodlanmış olmasıdır.

Literatürde yer alan önceki çalışmalar CUDA dilinde yapıldığı için yalnızca Nvidia grafik işlemcilerinde çalışabilen platforma bağlı çalışmalardır. Bu çalışmalarda [24] iris tanıma sistemlerinin 14 kata kadar hızlandırıldığı görülmektedir. Bu açıdan bu tezde yapılan çalışma performans arttırma yönünden geçmiş çalışmalara benzerlik göstermektedir. Fakat bundan farklı olarak OpenCL kullanılarak yapılmış bu çalışmada yer alan kod herhangi bir firmaya bağımlı olmadan tüm grafik işlemcilerde çalışabilmektedir. Bunun yanında aynı zamanda CPU üzerinde çoklu işlem özelliğine uygun şekilde paralel olarak işlem yapabilmektedir. Yine heterojen sistem mimarisine uygun olarak FPGA ve DSP kartlarında da çalışabilmektedir. Bunun günümüz teknolojisine daha uygun olduğu ortadadır. Bu yüzden son yıllarda yapılan çalışmalarda gerek heterojen mimariye gerekse de OpenCL kullanımına birçok alanda önem verilmiştir. Bu alanda yapılmakta olan çalışmalar günden güne artmaktadır. Platform bağımlılığının ortadan kaldırılması ve çeşitli işlemci gruplarında aynı kodun çalıştırılabilmesi birçok sektörde maliyet düşürme amaçlı olarak da kullanılmaktadır.

Gelişen teknolojik imkânlar maliyet etkin ve daha hızlı çalışan yazılımların yapılamasına daha fazla altyapı sağlamakla birlikte, bu altyapıların teknolojiye geri dönüşleri de bilim ve insanlık adına oldukça faydalı olmaktadır. Bu tez çalışmasında iris tanıma sistemlerinde kullanılan karşılaştırma algoritmasının daha hızlı olarak, daha az maliyetlerle oluşturulabilecek heterojen sistemlerde daha az anlık güç harcayarak gerçeklenmesini sağlayacak şekilde hazırlanmıştır. Yapılan çalışmalardan elde edilen sonuçlar daha önce konuyla ilgili olarak yapılmış hızlandırma çalışmalarına benzerlik göstermektedir. Diğer çalışmalara ek olarak heterojen sistemlere uygunluk ve platformdan bağımsızlık

sağlanmıştır. Burada sağlanan kazançlardan birisi de herhangi bir firmaya ya da platforma bağımlı olmadan iris karşılaştırma algoritması olarak kullanılan “Hamming Mesafesi” algoritmasının hızlandırılmasıdır.

Bu çalışmanın giriş bölümünde, çalışmada izlenen metottan bahsedilmiştir. 2008 yılında ortaya çıkan C temelli OpenCL dilinin sağladığı paralelleştirme avantajı kullanılarak işlemci çekirdeklerinde Hamming Mesafesi algoritmasının gerçeklenmiş ve bu dili destekleyen tüm işlemcilerde aynı kod çalıştırılabilir bir şekle sokulmuştur. OpenCL kullanılarak en az 22 kat hızlı çalışan bir kod elde edilmiştir.

Çalışmanın ikinci bölümünde; insanın, kişisel ve biyolojik yapısından kaynaklanan (biyometri) özellikleri kullanılarak diğer insanlardan ayırt edilebildiği konusu ele alınmıştır. Parmak izi, retina, DNA, yürüme şekli, yüz yapısı, tuş darbesi, el damarı ve iris gibi ayırt edici özellikler günümüz teknolojisinde kimlik tespiti ve doğrulaması amacıyla kullanılabilen özelliklerdir. Bu teknoloji ülkemiz ve diğer ülkeler tarafından kullanılmakta ve bu teknolojilerin geliştirilmesi amacıyla endüstriyel ve akademik birçok çalışma ortaya konulmaktadır.

Üçüncü bölümde iris tanıma sisteminin genel yapısı ele alınmıştır. İris tanıma sistemi temel olarak; iris özelliklerinin çıkarılması ve elde edilen bu özelliklerin karşılaştırılması bölümlerinden oluşur. Özellik çıkarımı işlemi; iris bölgesinin belirlenmesi ve iris normalleştirilmesi aşamalarından meydana gelir. Bu tez çalışmasında özellik çıkarımı adımları Libor MASEK’ in “Açık Kodlu İris Tanıma Sistemi” adıyla tüm akademik dünyayla paylaşmış olduğu MATLAB kodu kullanılarak gerçekleştirilmiştir. Bu kod C# dilinde otomatize edilmiş ve 1329 iris fotoğrafının özellikleri çıkarılmıştır.

Dördüncü bölümde OpenCL in genel yapısından bahsedilmiştir. OpenCL heterojen platform mimarisinde kullanılabilen ve performans artışı sağlayan C temelli bir programlama dilidir. Sağlayıcı (host) adı verilen bir CPU tarafından, OpenCL destekleyen hedef cihazlara kod dağıtımı yapılır. Hedef cihazlar ise kendilerine gönderilen bu kodları ve hafızalarına yazılan giriş verilerini işleyerek sonucu kendi hafızalarına yazarlar. Sağlayıcı cihaz bu sonucu hedef cihazların hafızasından okuyarak kullanır. Bu işlem sırasında sağlayıcı cihaz çoğunlukla yönlendirici olarak kullanılır. Hedef cihazlar ise çok çekirdekli işlemciler (GPU, FPGA, DSP gibi) olduğu için aynı anda çekirdek sayısı kadar

işlemi gerçekleştirerek kodun paralelleştirilmesini ve performans artışını sağlar.

Beşinci bölümde tez çalışmasında gerçekleştirilmiş olan uygulama anlatılmıştır. Libor MASEK'in "Açık Kodlu İris Tanıma Sistemi" C# programlama dili kullanılarak otomatize edilmiştir. Böylece 1329 iris resmi otomatik olarak bu sisteme girdi olarak sağlanmış ve elde edilen özelliklerden bir veri tabanı elde edilmiştir. Daha sonra C++ programlama dilinde Hamming Mesafesi algoritması seri olarak yazılmıştır. Seri kodun şablonlar üzerindeki karşılaştırma performans verileri elde edilmiştir. Aynı algoritma bir de OpenCL kullanılarak paralel olarak kodlanmış ve bu hızlandırılmış karşılaştırma işleminin performansı, seri kodun performansı ile karşılaştırılmıştır. Elde edilen performans verileri ile hız artışı değerleri hesaplanmıştır.

OpenCL ile yazılmış olan algoritma hem işlemci üzerinde hem de grafik işlemcisi üzerinde en az 22 kata kadar hızlandırılmıştır. Çeşitli iris boyutları ile gerçekleştirilen denemelerde bu artışın 88 kata kadar arttırıldığı gözlenmiştir.

Farklı cihazlar üzerinde ilgili kod denenmemiştir. FPGA ya da DSP kartları üzerinde ilgili kod denenerek sağlanan güç avantajı gelecek çalışmalarda değerlendirilebilir. Güç analizinin yapılabilmesi için sisteme bağlı cihazların anlık olarak çektikleri akımın o anki gerilim değeriyle birlikte izlenmesi gerekmektedir. Tüm sistem için bunu sağlamak zor olabilir. Çünkü işlemci temel olarak işletim sistemi gibi sürekli olarak çalışan sistemsel bazı yazılımları üzerinde çalıştırdığı için hangi uygulamanın ne kadar akım çektiğini izlemek ve buna göre bir yorum yapmak zor olabilmektedir.

Yapılan çalışmanın bilime katkısı, platform ya da cihazdan bağımsız olarak tüm OpenCL destekleyen cihazlarda iris karşılaştırmada da kullanılan Hamming Mesafesi algoritmasının gerçeklenebilir hale getirilmesidir. Ve bunu yaparken aynı zamanda işlemcide koşan seri koda göre performans kazancı sağlamaktadır.

KAYNAKLAR

1. Adler, F.H. (1953). *Physiology of the Eye*. St. Louis: Mosby Company, 143.

2. Flom, L., and Safir, A. (1987). Iris Recognition Systems, **United States Patent,** Patent Number: 4641349.

3. Daugman, J. (1994). Biometric Personal Identification System based on Iris Analysis, **United States Patent**, Patent Number: 5,291,560.

4. İnternet: Masek, L. Recognition of Human Iris Patterns for Biometric Identification. URL:http://www.webcitation.org/query?url=http%3A%2F%2Fwww.peterkovesi.com%2Fstudentprojects%2Flibor%2F&date=2016-11-16, Son Erişim Tarihi:16.11.2016

5. Proenç, H., and Alexandre, L.A. (2005, September). *UBIRIS: A Noisy Iris Image Database*, 13th International Conference on Image Analysis and Processing, 970-977.

6. Jain, A., Hong, L., and Pankanti, S. (2000). Biometric Identification. *Communications of the ACM*, 43(2), 91-98

7. Jain, A.K., Ross, A., and Prabhakar, S. (2004, January). An Introduction to Biometric Recognition, *IEEE Transactions on Circuits and Systems for Video Technology*, 14(1), 4–20.

8. Delac, K., and Grgic, M. (2004). *A Survey of Biometric Recognition Methods*, In Proceedings of the 46th International Symposium Electronics in Marine (ELMAR), 184–193.

9. Zhao, W., Chellappa, R., Phillips, P. J., and Rosenfeld, A. (2003, December). Face Recognition: A Literature Survey, *ACM Computing Surveys*, 35(4), 399–458.

10. Niyogi, S., and Adelson, E. (November 1994). *Analyzing Gait with Spatiotemporal Surfaces*, In Proceedings of the IEEE Workshop Non-Rigid Motion, 24–29.

11. Bhattacharyya, D., Ranjan, R., Alisherov, F., and Choi, M. (September 2009). Biometric Authentication: A Review, *International Journal of u- and e- Service Science and Technology* 2, 19.

12. Sahmoud, S. and A., Abuhaiba, I. S. I. (2011). *Enhancing Iris Recognition*, Master Thesis, Islamic University Gaza Faculty of Engineering, Gaza, 63.

13. Ng, R. Y. F., Tay, Y. H., and Mok, K. M. (2008). *A Review of Iris Recognition Algorithms*, Information Technology, ITSim. International Symposium, 2,1-7.

14. Wildes, R.P. (1997). Iris Recognition: An Emerging Biometric Technology *Proceedings of the IEEE*, (85), 1348-1363.

15. Daugman, J. (1993). High Confidence Visual Recognition of Persons by a Test of Statistical Independence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11), 2.

16. Huang, J., Wang, Y., Tan, T., and Cui, J. (2004). *A New Iris Segmentation Method for Recognition*, Proceedings of the 17th International Conference on Pattern Recognition (ICPR'04), 1051-4651

17. Daugman, J. (2004). How Iris Recognition Works, *IEEE Transaction Circuits Systems Video Technology*, 14(1), 21 -30

18. Hamming, R. W. (1950). Error Detecting and Error Correcting Codes, *Bell System Technical Journal* 29(2), 147–160.

19. İnternet: OpenCL Introduction and Overview Khronos Group, (June 2010). URL: http://www.webcitation.org/query?url=https%3A%2F%2Fwww.khronos.org%2Fassets%2Fuploads%2Fdevelopers%2Flibrary%2Foverview%2FOpenCL-Overview-Jun10.pdf&date=2016-11-16, Son Erişim Tarihi:16.11.2016

20. İnternet: Munshi, A. (2009, June). The OpenCL Specification. URL: http://www.webcitation.org/query?url=https%3A%2F%2Fwww.khronos.org%2Fregistry%2Fcl%2Fspecs%2Fopencl-1.0.pdf&date=2016-11-17 Son Erişim Tarihi: 17.11.2016

21. Tompson, J., Schlachter, K. (2012). *An Introduction to the OpenCL Programming Model*, New York University: Media Research Lab, 1.

22. İnternet: BOYDSTUN, K. (August 2011). Introduction OpenCL. URL: http://www.webcitation.org/query?url=http%3A%2F%2Fwww.tapir.caltech.edu%2F%7Ekboyds%2FOpenCL%2Fopencl.pdf&date=2016-11-16 Son Erişim Tarihi: 16.11.2016

23. Aknan, M. (2014, May). *Iris Localization using Parallel Computing*, Master Thesis, Department of Computer Science and Engineering National Institute of Technology Rourkela, Rourkela, 12.

24. Sakr, F.Z., Taher, M., and Wahba, A.M. (2011). *High Performance Iris Recognition System On GPU*, International Conference on Computer Engineering & Systems (ICCES), 237 – 242.

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, adı : ŞİMŞEK, Buğra
Uyruğu : T.C.
Doğum tarihi ve yeri : 19.03.1991, Ankara
Medeni hali : Evli
Telefon : 0 (506) 738 04 94
e-mail : bugras@aselsan.edu.tr

Eğitim

Derece	Eğitim Birimi	Mezuniyet tarihi
Yüksek lisans	Gazi Üniversitesi /Elektrik Elektronik Müh.	Devam Ediyor
Lisans	Gazi Üniversitesi /Elektrik Elektronik Müh.	2013
Lise	Fethiye Kemal MUMCU Anadolu Lisesi	2009

İş Deneyimi

Yıl	Yer	Görev
2013-Halen	ASELSAN A.Ş.	Mühendis

Yabancı Dil

İngilizce

Yayınlar

-

Hobiler

Uluslararası İlişkiler, Kişisel Gelişim, NLP

DİZİN

GAZİ GELECEKTİR..