7.5.13

Geniş Veri Kümeleri Üzerinde Paralel Öbekleme Uygulaması: Paralel WaveCluster

Ahmet Artu Yıldırım, Cem Özdoğan

Bilgisayar Mühendisliği Bölümü
Çankaya Üniversitesi
artu@computer.org, ozdogan@cankaya.edu.tr

Öz

Depolama boyutlarının artması ve günümüzde bilgi ediniminin daha kritik hale gelmesi ile birlikte geniş veri kümeleri üzerinde paralel öbekleme analizi çalışmaları hızla artmakta ve önem kazanmaktadır. Bu çalışmamızda öbekleme analizi konusunda sıkça kullanılmaya başlayan WaveCluster algoritması temel alınarak geliştirdiğimiz paralel WaveCluster algoritmasını ve elde edilen başarım grafikleri ile kazanımları anlatılacaktır. Geliştirilen algoritma, WaveCluster algoritması konusunda bildiğimiz kadarıyla ilk paralelleştirme çalışmasıdır. Algoritmanın hızlanma, verimlilik başarım ölçütleri ve doğrusallık karakteristiği incelenmesinden elde edilen sonuçlar sunulmuş ve tartışılmıştır. Bu çalışmada elde ettiğimiz sonuçlar, geliştirilen koşut zamanlı (paralel) algoritmanın sıralı (seri) algoritmaya göre sadece çalışma zamanını düşürmediği ayrıca toplam hafıza kullanımında da sağladığı verimlilik ile geniş veri kümeleri için oldukça uygun olduğunu göstermiştir. Kullanılan işlemci sayısı aralığında, daha geniş veri kümelerinin çalışılmasında daha fazla doğrusallık karakteristiği gözlenmiştir.

1. Giriş

Bilişim teknolojilerinin gelişimi ile birlikte, büyük veri kümeleri ile karşılaşmamız artık çok olağan hale geldi. Veri madenciliğini, bu büyük miktardaki veri kümesinin içinde bulunan "gizli" örüntüleri keşfetme süreci olarak tanımlayabiliriz. Temel veri madenciliği tekniklerinden biri olan öbekleme analizi ise veri kümesini oluşturan elemanları, belirli bir benzerlik kıstasına göre önceden tanımlanmamış kümelere ayırmada kullanılan, genel bir tekniktir. Bu teknik; örüntü tanımada, coğrafi bilgi sistemlerinde (GIS), görüntü işlemede, makine öğrenmesinde, web dokümanlarının sınıflandırılması vs. alanlarında kullanılmaktadır.

Günümüzde veri kümelerinin boyutlarının artık terabaytlar seviyelerine gelmiş olması ve kullanılan algoritmanın hesap karmaşıklığına bağlı olarak, mevcut bilgisayarlar (özellikle bellek miktarı ve işlemci gücü açısından) hala yeterli olamamaktadır. Bu nedenle hem çalışma zamanını kısaltmak, daha yüksek sistem büyüklüklüklerini çalışabilmek hem de kaynakları etkin bir şekilde kullanmak için geniş veri kümeleri üzerinde pek çok paralel öbekleme çalışmaları yürütülmektedir.

Izgara tabanlı öbekleme analizi algoritmalarından biri olan WaveCluster algoritması [1], her türlü öbek şeklini tespit edebilme becerisi, veri kümesi içersinde bulunan aykırı değerleri kaldırabilmesi ve bu değerlerden etkilenmemesi, farklı seviyelerde öbekleri keşfedebilmesi ve hızlı işlem yapabilme özelliği ile son zamanlarda sıkça kullanılan bir algoritmadır. Bu çalışmamızda, WaveCluster algoritmasını önce sıralı kod olarak yazdık ve sonrada koşut zamanlı hale getirdik. Paralel WaveCluster çalışmamızı, üzerinde Linux işletim sistemi kurulu bilgisayarlardan oluşan ve yıldız ağ mimarisine sahip bilgisayar öbeği üzerinde yürüttük. Kullanılan bilgisayar öbek sistemi dağıtık-hafıza sistemi tabanlıdır ve işlemciler birbirleri ile mesaj geçirme arayüzünü MPI (Message Passing Interface) [2] kullanarak haberleşmektedir. Geliştirdiğimiz paralel algoritma üzerinde yaptığımız başarım analizi çalışmaları, bu algoritmanın geniş veri kümeleri üzerinde paralel veri madenciliği yapmak için uygun olduğunu göstermiştir.

Bu çalışma şu şekilde yapılanmıştır. Çalışmanın ikinci bölümünde öbekleme analizi algoritmaları hakkında bilgi vereceğiz. Üçüncü bölümünde izlenen paralel veri madenciliği yaklaşımları tanıtılacaktır. Dördüncü bölümde seri WaveCluster algoritmasını, beşinci bölümde ise geliştirdiğimiz paralel WaveCluster algoritmasını tanıtacağız. Altıncı bölümde çalışmamızda, geniş veri kümelerini kullanarak çıkardığımız sonuçlarını açıklayacağız. Yedinci bölümde paralel WaveCluster algoritmasını daha iyileştirebilecek konular tartışılacaktır.

2. Öbekleme Analizi Algoritmaları

Günümüze kadar pek çok öbekleme analizi çalışmaları yapılmıştır. Bu çalışmaları sıralı öbekleme analizi algoritmaları ve paralel öbekleme analizi algoritmaları olarak ikiye ayırabiliriz.

2.1. Sıralı Öbekleme Analizi Algoritmaları

Veri kümesinin k alt kümeye bölündüğü k-means algoritması [3] bölünmeli öbekleme algoritmasıdır. Her öbek, öbek merkezi (ortalama nokta) ile tanımlanır. Başlangıçta k tane öbek merkezi, rastgele olarak veri kümesi içindeki nesnelerden seçilir ve nesneler öbek merkezine yakınlığına göre atanır. k-means algoritmasındaki amaç her nesnenin öbek merkezlerine olan toplam kare-hata değerini minimize etmektir. Algoritma öbek merkezleri değişmeyinceye kadar sürdürülür. Bu algoritma basit bir algoritma olması ile birlikte öbek merkezleri ortalama nokta yaklaşımı ile seçildiğinden aykırı nesnelere çok duyarlıdır ve veri kümesinde küre şeklinde olmayan öbekleri bulmada sorun yaşamaktadır. Seri k-means algoritmasının zaman kompleksitesi O(NkT)'dir (N: sistem büyüklüğü, k: istenen öbek sayısı, T: iterasyon sayısı olmak üzere) [4].

BIRCH [5] algoritması hiyerarşik öbekleme algoritmasıdır. BIRCH çok boyutlu veri kümesi nesnelerini artımsal ve devinimsel bir şekilde kaynak yeterliliğini göz önüne alarak (mevcut bellek miktarı, zaman kısıtlaması gibi) öbekler. BIRCH veri kümesinin bir sefer taranması ile iyi öbekleme sonuçları vermektedir ve veri kümesi üzerinde yapılan ek

tarama işlemleri bu kaliteyi artırmaktadır. BIRCH aynı zamanda aykırı nesnelerin de üstesinden gelebilen ilk öbekleme algoritmasıdır. Öbekleme yaparken CF (Clustering Feature) ağacı veri yapısından faydalanır. CF ağacı öbek ile ilgili özellikleri tutan bir veri yapısıdır. Bu özellikler öbek içindeki nesne sayısı, bu nesnelerinin doğrusal toplamı ve karesi toplamıdır. BIRCH algoritmasının zaman kompleksitesi $O(N^2)$'dir.

DBSCAN [6] algoritması, yoğunluk tabanlı bir öbekleme algoritmasıdır. Bu algoritmada öbek, yoğun bağlı nesneler kümesi olarak tanımlanır. Yoğunluk parametreleri olarak, çember yarıçap değeri ε ve bu ε yarıçapına sahip çember içindeki nesnelerin yoğun olarak kabul edilmesi için, çember içindeki en az nesne sayısı, MinPts değerinin tanımlanması gerekir. DBSCAN algoritması noktaların komşuluklarını yarıçap tabanlı yaptığı için karmaşık öbek şekillerini tespit edememektedir. Ayrıca öbekleme sonucu seçilen ε değerine ve MinPts değerine göre farklılık göstermektedir. DBSCAN algoritmasının zaman kompleksitesi $O(N\log N)$'dir.

2.2. Paralel Öbekleme Analizi Algoritmaları

Paralel k-means çalışmasında [7], veri kümesi tüm işlemciler arasında eşit olarak dağıtılır. Her işlemci, kendi yerel veri kümesi üzerinde merkez noktalarına göre yerel hata kareleri toplamını hesaplar ve diğer işlemcilerle sonuçları değiş tokuş eder. Bu yerel hata kareleri toplamları eklenerek genel hata kareleri toplamları hesaplanır. Algoritma genel hata kareleri toplamları değişmeyinceye kadar devam eder.

Paralel DBSCAN [8], yoğunluk tabanlı bir paralel öbekleme algoritmasıdır. DBSCAN algoritmasının yürütülmesinde en fazla zaman %95 oran ile nesnelerin komşularının belirlenmesi işlemidir. Bu nedenle paralel DBSCAN algoritmasında, ana işlemci öbek atamalarını yaparken, yardımcı işlemciler noktaların komşuluk sorgulamasını yürütür.

PBIRCH algoritmasında [9] öncelikle veri kümesi işlemciler arasında eşit olarak paylaştırılır ve her yeni veri ulaştığında bu veriler periyodik olarak işlemciler arasında dağıtılır. Her bir işlemci CF ağacını kendi yerel veri kümesini kullanarak paralel olarak oluşturur ve işlemciler çıkan yerel sonuçları genel ortalama verilerini hesaplamak için değiş tokuş ederler. PBIRCH algoritması, işlemci sayısı arttıkça haberleşme zamanı baskın geldiği için düşük hızlanma değeri vermektedir.

Bu algoritmaların dışında öbekleme analizi için genel bir paralel yöntembilimi de geliştirilmiştir [10]. Bu yöntembiliminde, veri kümesi eşit alt veri kümelerine ayrılır ve her alt veri kümesi üzerinde seçilen öbekleme algoritması uygulanır. Bu sürecin sonucunda her alt veri kümeleri için öbek merkezleri oluşur. Daha sonraki adımda bu öbek merkezleri birleştirilerek meta tablo oluşturulur. Bu meta tablo üzerinde, tekrar seçilen öbekleme algoritması uygulanır ve meta-öbek merkezleri oluşturulur. Bu meta-öbek merkezlerinin kesin sonuç olarak kabul edilmesi için her alt veri kümesinin öbek merkezleri ile meta-öbek merkezleri belli bir uzaklık sınır eşiği temel alınarak karşılaştırılır. Eğer başarı yüzdesi en az önceden tanımlanan başarı yüzdesi eşiği kadar ise meta-öbek merkezleri kesin sonuç olarak kabul edilir. Değilse alt veri kümelerinin sayısı başlangıçta seçilen alt veri kümesinin sayısının katları olacak şekilde artırılır ve tüm işlem tekrar uygulanır. Doğal olarak alt veri kümeleri benzer oldukça başarı yüzdesi daha fazla olacaktır.

3. Paralel Veri Madenciliği Yaklaşımları

Mevcut veri madenciliği algoritmaları üzerinde günümüze kadar pek çok iyileştirmeler yapılmasına karşın, geniş veri kümeleri üzerinde veri madenciliği algoritmalarının çalışma süresinin kısaltılması hala gerekmektedir. Paralel veri madenciliğinin amacı, sıralı çalışan veri madenciliği algoritmasının yaptığı işi, işlemciler arasında dağıtarak koşut zamanlı hale getirmektir. Böylece çalışma zamanının kısaltılması ve çalışılabilecek veri kümesi büyüklüğünün artırılabilmesi mümkün olacaktır. Veri madenciliği algoritmalarının koşut zamanlı hale getirilmesinde izlenen üç farklı yaklaşımdan bahsedebiliriz [11]. Bunlar bağımsız arama, paralelleştirilmiş sıralı veri madenciliği ve tekrarlı sıralı veri madenciliği yaklaşımlarıdır. Bu yaklaşımlar arasında hangisinin daha iyi sonuç verdiği, kullanılan veri madenciliği algoritmasına göre farklılık göstermesiyle birlikte, genelde tekrarlı sıralı veri madenciliği yaklaşımı daha iyi sonuç vermektedir [11].

3.1. Bağımsız Arama Yaklaşımı

Bağımsız arama yaklaşımında, her işlemcinin bütün veri kümesine erişimi vardır ve tüm işlemciler birbirinden bağımsızdır. Her işlemci için rastgele arama uzayı belirlenmesinin ardından, işlemci aynı algoritmayı ρ defa çalıştırır. ρ sayısı deneyimsel saptamalara göre belirlenebilir. Son olarak işlemciler çıkardıkları sonuçları birbirleri ile değiş tokuş ederler ve bu sonuçlar arasından en iyi sonuç seçilir. Bu metodun en büyük yarar yitimi tüm işlemcilerin bütün veri kümesine erişebiliyor olmasıdır. Bu açıdan oldukça maliyetli bir yoldur. Her makinede veri tabanının tamamının tutulması gerekebilir. Bu da hafıza kullanımını arttırmaktadır. Diğer taraftan algoritmada, işlemciler sadece hesaplamanın en sonunda elde ettikleri sonuçları dağıttıklarından, işlemciler arasındaki veri iletişiminin de en az olduğu yaklaşımdır.

3.2. Paralelleştirilmiş Sıralı Veri Madenciliği Yaklaşımı

Bu yöntem "kavram bilgilerini dağıtma" yöntemi üzerine kuruludur. Öncelikle birincil kavramlar işlemciler arasında dağıtılır. Daha sonra her işlemci veri kümesinin tamamı üzerinde veya bir kısmı üzerinde veri madenciliği algoritmasını çalıştırır. Algoritmanın çalıştırılmasından sonra oluşan yeni kavramlar işlemciler arasında değiş tokuş edilir ve genel olarak doğru olmayan kavramlar elenir. Bu yöntem, işlemciler arasında değiş tokuş edilecek hiç bir kavram olmayana dek devam eder. Bu algoritma işlemcilerin her iterasyonda kavram bilgilerini değiş tokuş etmelerinden dolayı veri iletişimi açısından pahalı bir algoritmadır. Bu metoda örnek olarak birliktelik kuralları bulma algoritmasının paralelleştirilmesinde kullanılan veri dağıtımı algoritmasını (Data Distribution) verebiliriz [12].

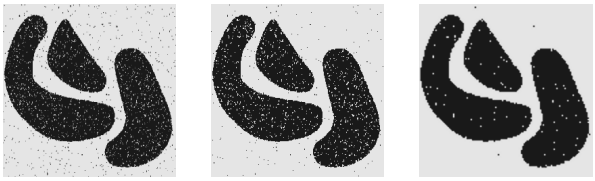
3.3. Tekrarlı Sıralı Veri Madenciliği Yaklaşımı

Bu yöntemde, veri kümesi tüm işlemciler arasında paylaştırılır ve her işlemci veri madenciliği tekniğini kendi yerel veri kümesi üzerinde uygular. Sonuç yerel olarak doğru iken genel olarak doğru olmayabilir. Bu yüzden algoritmanın sonunda her işlemci kendi çıkardığı yerel değerleri diğer işlemciler ile değiş tokuş eder ve bu yerel değerler birleştirilerek genel olarak doğru olan değerler oluşturulur. Bu yaklaşıma örnek olarak yine birliktelik kurallarının paralelleştirilmesinde

kullanılan sayma dağıtımı algoritmasını (Count Distribution) verebiliriz [12].

4. WaveCluster Algoritması

WaveCluster yöntemi, geniş veri kümeleri üzerinde wavelet dönüşümü uygulayarak farklı örüntülere sahip olan karmaşık kümeleri bile keşfedebilme özelliğine sahip, ızgara tabanlı bir öbekleme algoritmasıdır [1]. Veri kümesi üzerinde wavelet dönüşümünün uygulanmasının temel amacı, bu dönüşümü uygulayarak veri kümesi içerisinde bulunan nesnelerin arasındaki uzaklığı azaltmak ve bu şekilde dönüştürülmüş daha yoğun ve aykırı değerlerden arınmış bir veri kümesi elde etmektir. Bu çalışmamızda Haar wavelet dönüşümünü [13,14] kullandık. Diğer wavelet yöntemlerinin Haar wavelet yönteminden daha iyi çözünürlüğe sahip olduğu bilinmektedir fakat daha ağırdırlar. Bizde bu çalışmamızda wavelet dönüşümü işleminin hızlı olması ve belleğin daha etkin kullanılmasını sağladığı için Haar wavelet dönüşümünü kullandık. Örnek olarak, Haar wavelet dönüşümü kullanılan WaveCluster algoritmasını Şekil 1(a)'da görülen 512x512 çözünürlüğündeki veri kümesi üzerinde uyguladık. Sonuç olarak dönüştürülmemiş orijinal veri kümesinde 378 öbek keşfedilmişken, bir defa Haar wavelet dönüşümü uygulandıktan sonra (Şekil 1(b)) bu sayı 130'a düşmüş, iki defa Haar wavelet dönüşümü uygulandıktan sonra ise (Şekil 1(c)) 6 tane öbek keşfedilmiştir.



a: Veri Kümesi *b:* $\rho = 1$ *c:* $\rho = 2$

Şekil 1: Orijinal veri kümesi (a) ve ρ defa Haar wavelet dönüşümü sonucu oluşan düşük frekanslı bileşenler (b,c)

WaveCluster algoritmasında ilk adım olarak η boyuta, $\eta \geq 1$ olmak üzere, sahip öznitelik uzayının her bir boyutunun önceden belirlenmiş aralık değer kümelerine göre sayısallaştırılması işlemi yapılır ve veri kümesi oluşturulur. Bu işlem aynı zamanda farklı aralık değer kümeleri için başarım ve kümeleme kalitesini de etkileyen önemli bir faktördür. İkinci adım olarak da, veri kümesi üzerinde wavelet dönüşümü uygulanır. Örnek olarak, wavelet dönüşümünün 2 boyutlu veri kümesine uygulanması sonucunda 3 adet detay sinyali olarak bilinen yüksek frekanslı bileşenler ve 1 adet ortalama sinyali olarak bilinen düşük frekanslı bileşen ortaya çıkar. Bu adım farklı doğruluk derecelerinde küme elemanlarını tespit etmek için birden fazla uygulanabilir. Üçüncü adımda, düşük frekanslı bileşen üzerinde bağlı parçaları işaretleme (connected components labeling) [15] algoritması uygulanır ve her bir parçanın komşuluk özelliğine bağlı olarak küme elemanları tespit edilir. Küme elemanları sayısallaştırılmış orijinal veri kümesindeki nesneleri değil, dönüştürülmüş öznitelik uzayındaki nesneleri temsil ederler. Bu nedenle WaveCluster algoritması, dönüştürülmüş öznitelik uzayındaki nesneleri, orijinal veri kümesindeki nesnelerle ilişkilendirmek için arama tablosu oluşturur. Bu arama tablosu kullanılarak orijinal veri kümesindeki nesneler öbeklerler ilişkilendirilirler.

WaveCluster algoritmasının zaman kompleksitesi O(N)'dir. Koşut zamanlı algoritmayı geliştirmeden önce yazdığımız sıralı çalışan yazılımın algoritması aşağıdaki gibidir.

Seri WaveCluster Algoritması:

Girdi: Çok boyutlu veri nesneleri örnek vektörleri

Çıktı: Öbeklere ayrılmış nesneler

1. Öznitelik uzayındaki nesneleri birimlere atayarak sayısallaştır
2. Sayısallaştırılmış veri kümesi üzerinde wavelet dönüşümü uygula
3. Dönüştürülmüş örnek uzay bileşeni üzerinde farklı seviyelerde bağlı parçaları bul
4. Bağlı ünitelere öbek numarası ile etiket ata
5. Arama tablosunu oluştur
6. Arama tablosunu kullanarak nesneleri öbeklerle ilişkilendir.

5. Paralel WaveCluster Algoritması

WaveCluster algoritması etkin bir algoritma olmasına karşın küme tespiti için kullanılan veri kümelerinin çok büyük boyutlarda olması durumunda bellek alanları yeterli olamamaktadır. Bu sorunun üstesinden gelmek, daha geniş veri kümelerini çalışabilmek ve eldeki kaynakları daha etkin kullanarak algoritmanın çalışma zamanını kısaltmak için tekrarlı sıralı veri madenciliği yaklaşımı izlenmiş ve paralel WaveCluster algoritması geliştirilmiştir.

Algoritma C programlama dili kullanılarak geliştirildi. Kullanılan bilgisayar öbek sistemi dağıtık-hafıza sistemi tabanlı olduğundan işlemcilerin birbirleri arasında haberleşmeyi sağlamak için mesaj tabanlı arayüze sahip olan MPI kütüphanesi kullanmalıdır (MPI kütüphanesi ayrıca paylaşımlı hafıza modelini de desteklemektedir). Bu iletişimin sağlanması için OpenMPI [16] kütüphanesi; veri yapılarından ve bazı yardımcı fonksiyonlarından faydalanmak için ise Glib [17] kütüphanesi kullanıldı. Dağıtık-hafıza yapısı içersinde, her bir süreç hafızada farklı adres uzaylarına sahiptir ve süreçler birbirleri ile mesaj gönder-al mantığıyla haberleşirler.

Paralel algoritmanın geliştirilmesinde ana işlemci-alt (yardımcı) işlemci yöntemi izlenmiştir. Bilgisayar öbeği altındaki işlemciler birbirlerine hızlı eternet anahtarlama cihazı ile bağlıdır. Tüm işlemciler geniş veri kümesinin bulunduğu dosya sunucusuna ağ dosya sistemi (NFS) üzerinden erişebilmektedirler.

Algoritmamızda alt işlemcinin görevi yerel veri kümesi üzerinde öbekleme analizi yapmaktır. Ana işlemcinin görevi ise alt işlemcilerden gelen sınır veri kümelerini karşılaştırarak oluşturduğu birleştirme tablolarını alt işlemcilere bildirmektir. Ayrıca ana işlemci de, kaynak kullanımında verimliliği artırmak için öbekleme analizi işlemini gerçekleştirmektedir. İşlemcilerin geniş veri kümesi üzerindeki yerel veri kümelerini etkin bir şekilde hafızalarına kopyalamalarını sağlamak için ana işlemci her alt işlemciye geniş veri kümesi üzerinde koordinat bilgisini gönderir. Koordinat bilgisi işlemcilerin veri kümesi üzerinde kendi dosya göstericisini konumlandırmasını sağlar (veri kümesi işlemciler arasında eşit olarak paylaştırılmaktadır). Tüm işlemciler veri kümesine ağ üzerinden erişebildiklerinden, veri kümesini okuma işlemi paralel olarak gerçekleşmektedir. İşlemciler yerel veri kümesi üzerinde WaveCluster algoritmasını ρ, $\rho \geq 1$, defa uygular. İlgili ρ wavelet uygulama sayısı arttıkça veri

kümesinden daha kaba özellikte öbekler keşfedilecektir. Ayrıca her wavelet dönüşümü uygulandığında düşük frekanslı bileşenin nesne sayısı daha az olacağından bağlı parçaları işaretleme algoritmasının çalışma süresi de daha az olacaktır. Sonraki adımda işlemciler wavelet dönüşümü sonucu oluşan düşük frekanslı bileşen üzerinde bağlı parçaları işaretleme algoritmasını uygular ve yerel öbekleri tespit ederler. Paralel algoritmada, standart 2 geçişli bağlı parçaları işaretleme algoritması kullanılmıştır [18]. Bu algoritmada öbek elemanları bağlı liste veri yapısından faydalanılarak bulunmaktadır. Alt işlemciler düşük frekanslı bileşenin sınır verilerini ana birime yollar ve ana birimden aldığı birleştirme tablosunu kullanarak küme numaralarını günceller. Her i nolu işlemcinin, $0 \leq i < k$, tespit ettiği öbeklere atadığı öbek numarası c^n, $(i-1)* MC < c^n < i*MC$ arasında olmalıdır. Bunun nedeni her işlemcinin tespit ettiği öbek numaralarının benzersiz olmalarını sağlamak ve bu sayede ana işlemcinin genel olarak geçerli olan birleştirme tablolarını oluşturabilmesini sağlamaktır. Son adım olarak işlemciler yerel arama tablosu oluşturur. Yerel arama tablosu ile dönüştürülmüş veri kümesindeki öbek numaraları orijinal veri kümesindeki nesnelerle ilişkilendirilir. Paralel WaveCluster algoritmamızda kullanılan sembollerin bazıları Tablo 1 de verilmiştir.

Tablo 1: Paralel algoritmada kullanılan notasyon

Sembol	Açıklama
K	İşlemci sayısı
ρ	Wavelet dönüşümü uygulama sayısı
MC	İşlemcilerin üretebileceği en fazla öbek sayısı
D	Veri kümesi boyutu

Paralel WaveCluster Algoritması:

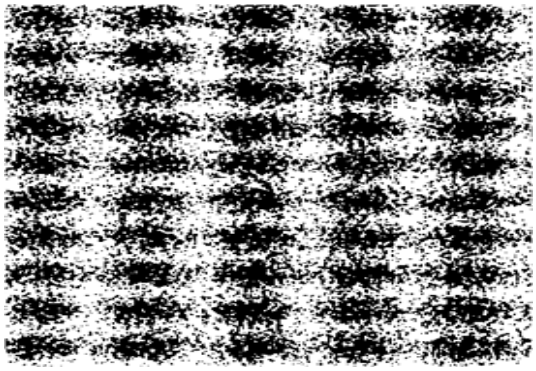
Girdi: d boyutlu geniş veri kümesi, MC, ρ

Çıktı: Öbeklere ayrılmış nesneler

1. Ana işlemci MC ve ρ değerlerini yayınlar
2. Ana işlemci, her alt işlemciye geniş veri kümesi üzerindeki koordinat bilgisini gönderir.
3. İşlemciler koordinat bilgisini kullanarak geniş veri kümesi üzerinde kendi yerel veri kümesini okur ve yerel hafızaya kopyalar
4. İşlemciler veri kümesi üzerinde ρ defa d boyutlu wavelet dönüşümü uygular ve düşük frekanslı bileşeni oluşturur
5. İşlemciler düşük frekanslı bileşen üzerinde bağlı parçaları işaretleme algoritmasını uygular ve bileşende bulunan her nesne için c^n öbek numarasını atar, $(i-1)*MC < c^n < i*MC$, $0 \leq i < k$
6. İşlemciler düşük frekanslı bileşenin sınır veri kümesini oluşturur
7. Alt işlemciler ana işlemciye sınır veri kümesini gönderir
8. Ana işlemci, tüm işlemcilerin sınır veri kümesini kullanarak, komşu nesneleri bulur ve alt işlemcilere birleştirme tablolarını yollar
9. İşlemciler birleştirme tablosunu kullanarak öbek numaralarını yeniler
10. İşlemciler yerel arama tablosunu oluşturur
11. İşlemciler yerel arama tablosunu kullanarak orijinal veri kümesindeki nesneleri öbeklerle ilişkilendirir

6. Sonuçlar

Geliştirdiğimiz yazılım, paralel WaveCluster algoritmasının veri madenciliği yapılmasına uygunluğunu değerlendirmek için Çankaya Üniversitesinde bulunan bilgisayar öbeğinde her biri 2.34 GHz gücünde 32 işlemciden faydalanarak gerçekleştirildi [19]. Deneylerde Şekil 2'de görülen 2 boyutlu kaynak veri kümesi, doğrusal aradeğerleme metodu kullanılarak 8192x8192 (67.108.864 nesne sayılı DS8 veri kümesi), 16384x16384 (268.435.456 nesne sayılı DS16 veri kümesi), 32768x32768 (1.073.741.824 nesne sayılı DS32 veri kümesi) ve 65536x65536 (4.294.967.296 nesne sayılı DS65 veri kümesi) çözünürlüklerinde oluşturulmuştur. Kesikli wavelet dönüşümünde, her bir iterasyonda oluşan bileşen çözünürlüğü, bir önceki iterasyondaki bileşen çözünürlüğüne kıyasla yarısı kadar olmaktadır. Bu nedenle, tam bir çoklu-çözünürlük analizi yapabilmek için kullanılan veri kümelerinin her bir boyutunun ikinin katları olacak şekilde olması sağlanmıştır. Çalışmada kullanılan kaynak veri kümeleri Sheikholeslami et al. [1] makalesinden alınmıştır.



Şekil 2: Örnek Veri Kümesi

Çalışmamızda veri kümesi üzerinde farklı ρ değerleri için wavelet dönüşümü uygulanmış ve başarım faktörleri ölçülmüştür. Başarım grafiklerinin oluşturulmasında I/O (okuma/yazma) işlemleri hariç tutulmuştur. I/O işlemi geniş veri kümesinin okunması ve öbek sonuçlarının diske yazılmasından oluşmaktadır. Hariç tutulmasının nedeni geniş veri kümelerinin diske yazılması süresinin uygulama amacına göre ve seçilen dosya formatına göre farklılık göstermesidir. Bu da incelenecek başarım ölçütlerinin tutarsız olmasına yol açabilecektir. Tablo 2'de I/O dahil ve hariç olarak ölçtüğümüz çalışma süreleri sunulmuştur. Bu tabloda sadece $\rho = 2$ için elde edilen değerler verilmiştir. Her bir veri kümesi için bulunan çalışma süreleri kaç defa kesikli wavelet algoritmasının uygulandığına bağlı olarak artmakta ya da azalmakatadır. Bundan dolayı sadece $\rho = 2$ durumundan elde edilen sonuçlar sunulmaktadır. Diğer çalışılan $\rho = 3$ ve $\rho = 4$ değerleri de için benzer bir eğilim gözlenmiştir. Tablodan görüleceği gibi bir ve iki işlemci ile çalışılması mümkün olmayan (yüksek bellek ihtiyacından dolayı) DS65 veri kümesinin, paralelleştirilen algoritma ile dört işlemciden itibaren çalışılabilmesi mümkün olmuştur. Sonraki tüm tablo ve şekillerimizde I/O işlemi hariç sonuçlarımız sunulmuştur.

Tablo 2: Çalışma süresi (sn) değerleri (I/O dahil ve hariç); değişen işlemci sayısı (n), veri kümesi boyutu, ρ = 2 için)

	n=1	n=2	n=4	n=8	n=16	n=32
DS8 (I/O)	238	133	105	95,1	91,6	90,4
DS16 (I/O)	993	539	423	380	365	360
DS32 (I/O)	3937	2159	1700	1516	1459	1437
DS65 (I/O)	-	-	7831	6430	5968	5963
DS8	33	11	5,4	2,7	1,9	1,6
DS16	135	40	17	7	3,8	2,5
DS32	600	165	65	24	10	5
DS65	-	-	491	103	24	15

Hızlanma faktörü birden fazla işlemcili paralel sistemin, tek işlemcili sıralı sisteme kıyasla göreceli başarım artışıdır. Tablo 3'de hızlanma oranları verilmiştir. Algoritmamız DS65 veri kümesinde bir ve iki işlemcili bir sistemde çalışmadığı için hızlanma değeri dört işlemcili sisteme göre hesaplanmış ve çıkan değerler teorik hızlanma değeri (dört) ile hizalanmıştır. Tüm değerler Tablo 2'den elde edilebilir. Örnek olarak DS8 veri kümesi alınırsa; n=1 (n: işlemci sayısı olmak üzere) için çalışma süresi 33 sn iken n=2 olduğunda bu çalışma süresi 11 sn olmuştur. Bunların hızlanma oranları ise 3 tür. Hızlanma faktörünün teorik olarak alması beklenen en yüksek değer sistemde kullanılan işlemci sayısı kadardır. Oysa Tablo 3'e bakıldığında elde edilen değerlerin beklenen değerlerden çok daha fazla olduğu görülecektir. Bu eğilim aynı şekilde bir diğer başarım ölçütü olan verimlilik değerlerinde de görülmektedir. Olası sebepleri verimlilik ölçütleri sunulurken tartışılacaktır.

Tablo 3: Hızlanma faktörü değerleri (I/O hariç); değişen işlemci sayısı (n) , veri kümesi boyutu, ρ = 2 için)

	n=2	n=4	n=8	n=16	n=32
DS8	3	6,11	12,22	17,36	20,62
DS16	3,37	7,94	19,2	35,52	54
DS32	3,63	9,2	25	60	120
DS65	-	-	19,06	81,83	130,93

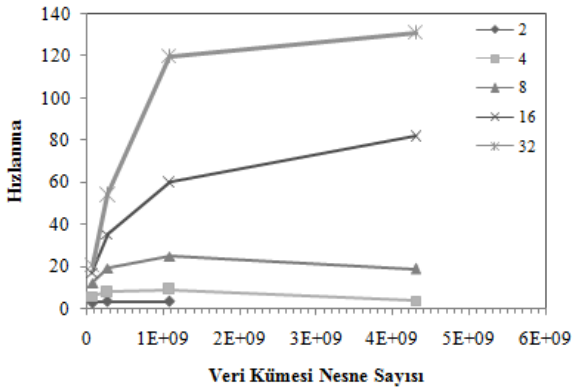
Verimlilik ölçütü ise, paralel sistemde işlemcilerden ne kadar fayda sağlandığının göstergesidir. Verimlilik ölçütü ayrıca belirli bir veri kümesi üzerinde çalışan paralel algoritmada ne kadar daha işlemci artırımı yapılabileceği hakkında fikir vermektedir. Eğer işlemci sayısının arttırılmasıyla elde edilen verimlilik azalmaya başlarsa, sisteme yeni işlemci katılmaması gerektiği anlaşılabilir. Tablo 4'de elde edilen verimlilik faktörleri gösterilmiştir. Bu değerler Tablo 3'de verilen hızlanma değerlerinin kullanılan işlemci sayısına bölünmesiyle elde edilmiştir. Yine örnek olarak DS8 veri kümesi alınırsa; n=2 için bulunan hızlanma faktörü 3 tür. Bu sayıyı kullanılan işlemci sayısı olan 2'ye bölersek, verimlik değeri için 1,5 bulunur. Verimlik değerinin teorik olarak beklenilen en yüksek değeri 1 dir.

Geliştirilen algoritmanın başarım ölçütlerinden elde edilen diğer bir analiz ise verimlik grafiklerinin doğrusallık niteliğine sahip olup olmamasıdır. Yani sisteme katılan her yeni işlemcinin katkısının en az diğerleri kadar olabilmesidir. Böylece verimlilik doğrusal bir karakter gösterir. Tablo 4'e bakıldığında ise çalışmamızda bulunan tüm değerlerin 1'den yüksek olduğu gözükmektedir. Bu durum pratikte kullanılan algoritmanın koşut zamanlı çalışmaya uygunluğuna bağlı olarak zaman zaman gözlenebilmektedir ve süper doğrusallık olarak adlandırılır. Temel sebebi artan hafıza miktarı ve her bilgisayara düşen yerel veri kümesinin küçülmesinin getirdiği aranılan verinin yerel önbellekde kolayca bulanabilmesidir. Yani bellek yönetimin çok iyileşmesidir.

Tablo 4: Verimlilik Değerleri (I/O hariç); Değişen işlemci sayısı (n) , veri kümesi boyutu, ρ = 2 için)

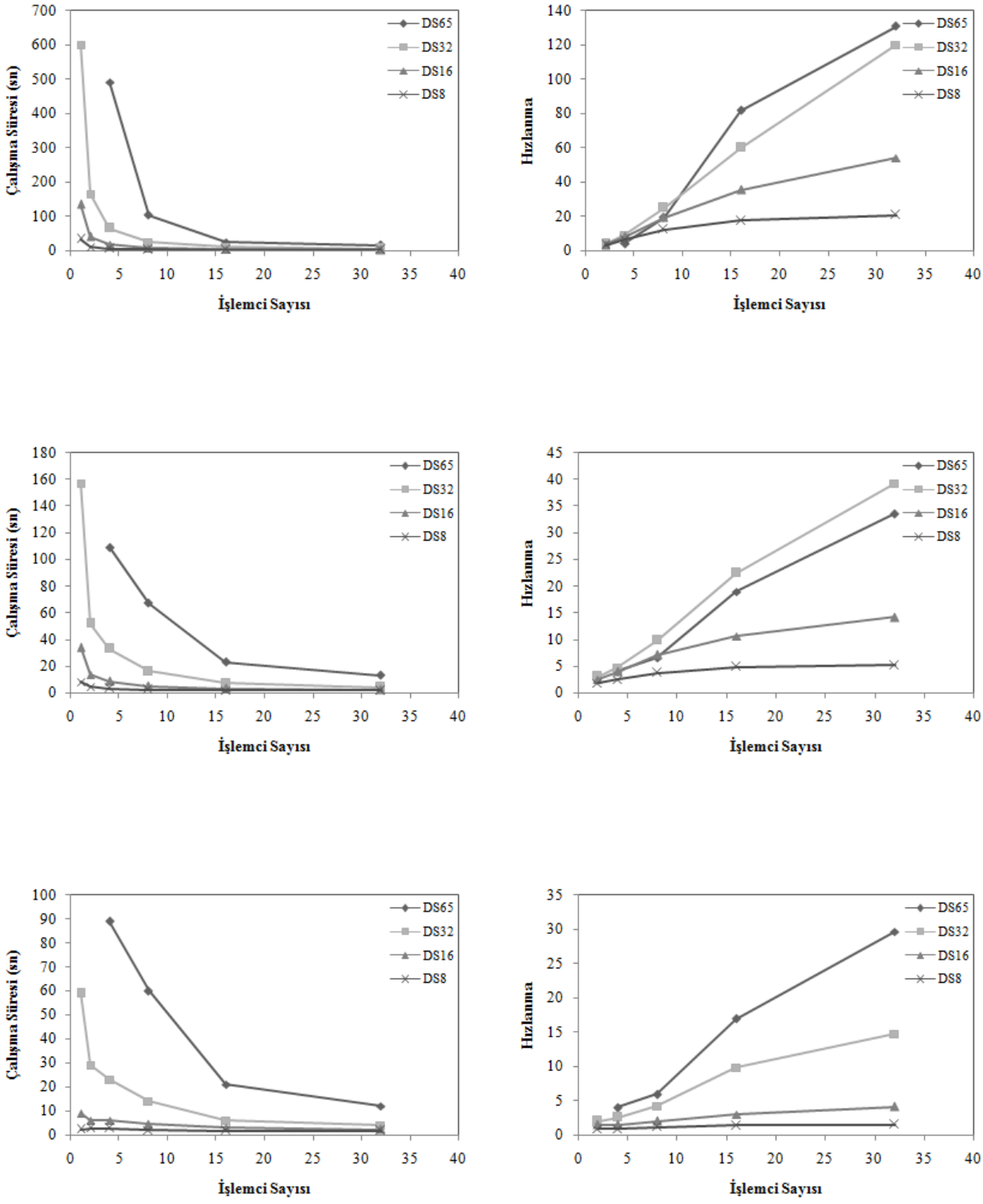
	n=2	n=4	n=8	n=16	n=32
DS8	1,50	1,52	1,52	1,08	0,64
DS16	1,68	1,98	2,41	2,22	1,68
DS32	1,81	2,30	3,12	3,75	3,75
DS65	-	-	2,38	5,11	4,09

Tablo 4'den görüleceği gibi yaptığımız deneylerde veri kümesinin nesne sayısı ve işlemci sayısı arttıkça verimlilik faktörünün süper doğrusallık karakteristiği gösterdiği gözlenmiştir. Bunun nedeni büyük veri kümelerinde bağlı parçaları işaretleme algoritmasının yoğun bellek erişimine ihtiyaç duyması ve veri kümesinin işlemciler arasında bölünmesi ile işlemci önbelleğinden daha fazla faydalanarak bellek erişimi süresinin dramatik bir biçimde düşmesidir. Bu duruma tek aykırılık olarak da Tablo 4'den görülebileceği gibi (Şekil 4'de ρ = 2 için hızlanma grafiğinde de bu aykırılık gözükmektedir) DS65 için n=8'de verimlik bir önceki DS32'ye göre daha az bulunmuştur. Bu da her ne kadar 8 işlemci ile bu büyüklükteki bir veri kümesi üzerinde çalışabiliyor olsak da verimli bir kullanım için daha fazla işlemci kullanılması gerektiğini gösterir. Ayrıca geliştirdiğimiz algoritmanın veri bölümlemesi ile aranılan verinin yerel önbellekte bulunma oranının azalmasıyla da açıklanabilir.



Şekil 3: Veri kümesi nesne sayısına göre hızlanma grafiği (farklı işlemci sayıları ve ρ = 2 için)

Şekil 3'de işlemci sayıları için ρ = 2 olmak üzere veri kümesindeki nesne sayısına göre hızlanma değerleri incelendi. Bu grafikten ilk gözlemlenen geliştirilen algoritmanın daha geniş veri kümeleri için daha iyi bir hızlanma sağlayabildiğidir. Bunun sebebi yukarıda bahsedilen önbellekteki yerel bilginin bulunma sıklığındaki artıştır. İşlemci sayısı 16 ve 32 iken



Şekil 4: Farklı ρ (wavelet dönüşümü uygulama sayısı), veri kümesi büyüklüğü ve işlemci sayılarına göre elde edilen I/O hariç çalışma süresi (sn) ve hızlanma grafikleri (Üst ρ = 2, Orta ρ = 3, Alt ρ = 4 için)

nesne sayısının artmasıyla hala bir hızlanma artışı görülmektedir. Bu artış yavaşlasa da 32 işlemci için yavaşlama hızı daha düşüktür. Bu da bize daha geniş veri kümelerinde doğrusallık karakteristiğinin daha fazla görülebileceğini gösterir. İşlemci sayısı 16 dan küçük olduğu durumlarda ise nesne sayısı arttıkça hızlanma faktörü düşme eğilimi göstermektedir. Bunun bir nedeni tüm işlemcilerin öbek numaralarını güncellemesi için ana işlemciyi beklemek zorunda olması ve sınır veri kümesi boyutunun artmasıyla ana işlemcide darboğaz oluşmasıdır. Diğer nedeni ise yukarıda bahsedilen yerele bilgi bulunma sıklığıdır.

Şekil 4'de farklı ρ (wavelet dönüşümü uygulama sayısı), veri kümesi büyüklüğü ve işlemci sayılarına göre elde edilen I/O hariç çalışma süresi (sn) ve hızlanma grafikleri (Üst ρ = 2, Orta ρ = 3, Alt ρ = 4 için) verilmiştir. Genel olarak görülen işlemci sayısının artmasıyla çalışma sürelerinin her veri kümesi için azalmasıdır. Aynı zamanda DS65 ve DS32 veri kümeleri için hızlanma oranında görülen doğrusallık eğilimi DS16 ve DS8'den elde edilenden daha fazladır. Bu durumda genel olarak daha geniş veri kümelerinde bu algoritmayla daha verimli olarak çalışılabilineceğini göstermektedir.

7. Tartışma

Bu çalışmada veri madenciliğinde öbekleme analizi yapmak için geliştirdiğimiz paralel WaveCluster algoritması sunulmuştur. Paralel algoritmanın başarımını artırmak için işlemciler arasındaki haberleşme en az düzeyde tutulmaya çalışıldı. Bununla birlikte ana işlemcinin alt işlemcilerden gelen sınır veri kümelerini alması ve her alt işlemci için birleştirme tablosu oluşturması işleminde algoritmanın doğasından gelen darboğaz oluşturduğu görüldü. Bu sorun işlemcilerin sınır verilerini diğer işlemcilere yayınlaması ve her işlemcinin birleştirme tablosunu kendisinin oluşturması ile çözülebilir.

Yapılan deneysel çalışmalar algoritmamızın doğrusallık karakteristiği taşıdığını göstermekle birlikte, aynı zamanda algoritmanın büyük veri kümelerinde bellek miktarının yetmediği durumlarda da kullanılabileceğini göstermiştir.

Paralel WaveCluster algoritmasında, ana işlemci tüm işlemcilere özel birleştirme tablosu oluşturmak için tüm işlemcilerin sınır veri kümelerini yollamasını beklemek durumundadır. O nedenle, sistem en kötü başarıma sahip işlemci hızı kadar hızlı olabilmektedir. Algoritmamızın başarımını analiz etmek için çalışmamızda öbek tespitinin zor olduğu bir veri kümesi kullanılmıştır. Daha az karmaşık öbek şekillerini içeren veri kümelerinde paralel algoritmanın doğrusallık karakteristiğinde bir farklılık görülmediği için bu sonuçlara makalemizde yer verilmemiştir.

Çalışmamızda standart çift geçişli bağlı parçaları işaretleme algoritması kullanılmıştır. Bu algoritmanın uygulanmasında bağlı liste veri yapısından faydalanılmıştır. Bağlı liste veri yapısı yerine standart dizi veri yapısının kullanılmasında çok daha iyi bir başarım elde edilebilir. Ayrıca çift geçişli bağlı parçaları işaretleme algoritması yerine iyileştirilmiş tek geçişli bağlı parçaları işaretleme algoritmasının [20] kullanılması ile de hem sıralı hem de paralel WaveCluster algoritmasının çalışma zamanlarında iyileşme meydana gelebilir.

8. Kaynakça

[1] Sheikholeslami, G., Chatterjee, S., Zhang, A., "Wavecluster: A wavelet-based clustering approach for spatial data in very large databases", The VLDB Journal, 83, 4, 289-304, 2000.

[2] Gropp, W., Lusk, E. ve Skejellum, A., "Using MPI: Portable parallel programming with the Message Passing Interface", MIT Press, Cambridge, MA, 1994.

[3] MacQueen, J.B., "Some methods for classification and analysis of multivariate observations", Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, University of California Press, 1, 281-297, 1967.

[4] Zamir, O. ve Etzioni, O., "Web document clustering: a feasibility demonstration", *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 46-54, 1998.

[5] Zhang, T., Ramakrishnan, R., ve Livny, M., "BIRCH: an efficient data clustering method for very large databases", *SIGMOD Rec.*, 25, 2, 103-114, 1996.

[6] Ester, M., Kriegel, H.P., Sander, J., ve Xu, X., "A density-based algorithm for discovering clusters in large spatial databases with noise", *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 226–231, 1996.

[7] Dhillon, I. ve Modha, D., "A data-clustering algorithm on distributed memory multiprocessors", *Large-Scale Parallel Data Mining, Lecture Notes in Artificial Intelligence*, 1759, 245–260, 1999.

[8] Arlia, D. ve Coppola, M., "Experiments in parallel clustering with DBSCAN", *In Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing*, 2150, 326-331, 2001.

[9] Garg, A., Mangla, A., Gupta, N., ve Bhatnagar, V., "PBIRCH: A scalable parallel clustering algorithm for incremental data", *In Proceedings of the 10th International Database Engineering and Applications Symposium, IEEE Computer Society*, 2006.

[10] Boutsinas, B. ve Gnardellis, T., "On distributing the clustering process", *Pattern Recognition Letters*, 23, 8, 999-1008, 2002.

[11] Skillicorn, D., "Strategies for parallel data mining", *IEEE Concurrency*, 7, 4, 26-35, 1999.

[12] Agrawal, R. ve Shafer, J. C., "Parallel mining of association rules", *IEEE Transactions on Knowledge and Data Engineering*, 8, 6, 962-969, 1996.

[13] Hosur, S. ve Tewfik, A. H. "Wavelet Transform Domain Adaptive FIR Filtering", *IEEE Transactions on Signal Processing*, 45, 3, 617-630, 1997.

[14] Stollnitz, E. J., DeRose, T. D., ve Salesin, D. H., "Wavelets for Computer Graphics: A Primer, Part 1", *IEEE Computer Graphics and Applications*, 15, 3, 76-84, 1995.

[15] Horn, B. K. P., "Robot vision", *The MIT Press*, 69-71, 1986.

[16] The Open MPI Project, "Open MPI: Open Source High Performance Computing", 2010. Adres: http://www.open-mpi.org/, [Ulaşıldı: 07 Nisan 2010].

[17] The Gnome Project, "GLib Reference Manual", 2010. Adres: http://library.gnome.org/devel/glib/, [Ulaşıldı: 07 Nisan 2010].

[18] Shapiro, L., ve Stockman, G., "Computer Vision", *Prentice Hall*, 69–73, 2001.

[19] Çankaya Üniversitesi, "Ganglia:: Cluster Report", Çankaya Üniversitesi Boron Öbeği Web Sayfası, 2010.

Adres: http://siber.cankaya.edu.tr/boron-ganglia/, [Ulaşıldı: 07 Nisan 2010].

[20] Ma, N., Bailey, D.G. ve Johnston, C.T., "Optimized single pass connected component analysis", *IEEE International Conference on Field-Programmable Technology*, 185-192, 2009.