Karesel Atama Problemi için
Grafik İşlem Birimleri Üzerinde
Paralel Bir Evrimsel Algoritma

Erdener ÖZÇETİN

Yüksek Lisans Tezi

Endüstri Mühendisliği Anabilim Dalı

Ocak-2013

Bu tez çalışması Anadolu Üniversitesi Bilimsel Araştırma Projeleri Komisyonu Başkanlığı tarafından desteklenmiştir. Proje No: 1109F151

JÜRİ VE ENSTİTÜ ONAYI

Erdener Özçetin'in "Karesel Atama Problemi için Grafik İşlem Birimleri Üzerinde Paralel bir Evrimsel Algoritma" başlıklı **Endüstri Mühendisliği** Anabilim Dalındaki, Yüksek Lisans Tezi 14.12.2012 tarihinde, aşağıdaki jüri tarafından Anadolu Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

	Adı Soyadı	**İmza**
Üye (Tez Danışmanı)	**: Yard. Doç Dr. Gürkan ÖZTÜRK**	**……..**
Üye	**: Prof.Dr. Refail KASIMBEYLİ**	**……..**
Üye	**: Doç.Dr. Özden ÜSTÜN**	**……..**

Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun …….. tarih ve …….. sayılı kararıyla onaylanmıştır.

Enstitü Müdürü

ÖZET

Yüksek Lisans Tezi

KARESEL ATAMA PROBLEMİ İÇİN GRAFİK İŞLEM BİRİMLERİ ÜZERİNDE PARALEL BİR EVRİMSEL ALGORİTMA

Erdener ÖZÇETİN

Anadolu Üniversitesi

Fen Bilimleri Enstitüsü

Endüstri Mühendisliği Anabilim Dalı

Danışman: Yard. Doç. Dr. Gürkan ÖZTÜRK

2012, 60 Sayfa

Bu çalışmada zor bir kombinatorik optimizasyon problemi olan karesel atama problemi yeni bir teknolojiyle çözülmek üzere ele alınmıştır. Matematiksel programlama yaklaşımları ile bazı küçük ve orta ölçekli problemlerin çözümlerinde dahi makul sürelerde en iyi sonuçlar elde edilememektedir. Bilgisayarların ekran kartları üzerinde yer alan grafik işlem birimleri büyük boyutta verileri eş zamanlı işleyerek, işlem zamanlarında anlamlı azalmalar sağlayabilmektedir. Bu yüzden, grafik işlem birimlerinin eş zamanlı işlem yapabilme gücünden yararlanarak, karesel atama probleminin kısa sürede etkin şekilde çözümü için paralel bir evrimsel algoritma geliştirilmiştir. Bu paralel algoritma ve merkezi işlem birimi üzerinde sıralı olarak çalışan hali, literatürde yer alan 59 test problemi üzerinde denenmiş ve elde edilen sonuçlar karşılaştırılmıştır. Test problemlerinin 43'ünde bilinen en iyi sonuca ulaşılmıştır. Bunun yanı sıra, paralel algoritmanın sıralı algoritmadan ortalaması 17 kat olmak üzere 51 kata kadar daha hızlı sonuç verebildiği gözlemlenmiştir.

Anahtar Kelimeler: Karesel atama problemi (KAP), evrimsel algoritmalar, paralel programlama, grafik işlem birimleri, CUDA.

ABSTRACT

Master of Science Thesis

A PARALLEL EVOLUTIONARY ALGORITHM FOR QUADRATIC ASSIGNMENT PROBLEM ON GRAPHICS PROCESSING UNITS

Erdener ÖZÇETİN

Anadolu University

Graduate School of Sciences

Industrial Engineering Program

Supervisor: Assist. Prof. Dr. Gürkan ÖZTÜRK

2012, 60 Pages

In this study, quadratic assignment problem, which is a hard combinatorial optimization problem, is examined to solve by a new approach. To reach the optimal results by using mathematical programming approaches cannot be possible even for some sorts of small and middle scaled problems in a reasonable time interval. Huge amounts of data are being progressed simultaneously by graphics processing units located on computers' graphics card. Therefore, a parallel evolutionary algorithm has been proposed to solve the quadratic assignment problem by using graphics processing units' simultaneously progressing property. This parallel algorithm and the sequential one on central processing units are tested and compared for 59 problems in literature. Best known solutions are obtained for 43 of these problems. Indeed, it is observed that the parallel algorithm works averagely 17 times and up to 51 times faster than sequentially one.

Key Words: Quadratic assignment problem (QAP), evolutionary algorithms, parallel programming, graphics processing units (GPU), CUDA.

TEŞEKKÜR

Öncelikle tüm yaşamım boyunca hiçbir desteğini esirgemeyen bugünlere gelmemde en büyük katkıya sahip aileme teşekkürü bir borç bilirim. Ayrıca tanıştığımdan beri birçok alanda gelişmemde pay sahibi olan, tez çalışmam süresince hiçbir yardımını esirgemeyen ve her karamsarlığa düştüğümde beni güçlendiren danışman hocam Gürkan Öztürk'e sonsuz teşekkürlerimi sunarım.

Öte yandan, tezimi 1109F151 numaralı projeyle destekleyen Anadolu Üniversitesi Bilimsel Araştırma Projeleri Komisyonu Başkanlığı'na, hocalarım Refail Kasımbeyli, Zehra Kamışlı Öztürk ve Ilgın Acar'a, arkadaşlarım Emre Çimen, Gülçin Yalçın, Emine Akyol ve Cenk İçöz'e manevi desteklerinden dolayı ayrı ayrı teşekkür ederim.

Erdener ÖZÇETİN

Aralık 2012

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

1. GİRİŞ

Grafik işlem birimleri (GİB), bilgisayar sistemlerinde görselleştirme aracı olarak kullanılmaktadır. Günümüzde gelişmiş ve çok güçlü ekran kartları üzerinde yüzlerce bulunan GİB, kitleler halindeki veriyi tamamen paralel bir yapıya dönüştürerek görselleştirmektedir [1]. Bununla birlikte, GİB yalnızca verilerin görselleştirmesi için kullanılmakla kalmayıp, özellikle yoğun aritmetik işlem gerektiren hesaplamalarda da kullanılabilmektedir. Bu tür hesaplamalar GİB üzerinde gerçekleştirilerek, işlem süreleri kayda değer şekilde azaltılabilmektedir. Fakat merkezi işlem birimlerinden (MİB) farklı olarak, GİB verilerin paralel hesaplamaya uygun hale getirilmesine ihtiyaç duymaktadır. Bu durum, GİB kullanımında dikkat edilmesi gereken en önemli adım ve üzerinde en çok emek sarf edilen kısımdır.

Kombinatorik optimizasyon problemlerinden biri olan karesel atama problemi (KAP) ilk olarak bölünmez ekonomik faaliyetlerin yerinin belirlenmesinin matematiksel modellenmesi olarak ortaya atılmıştır [2]. KAP sıklıkla aday tesislerin aynı sayıda yer kümesine, mesafe ve akıştan oluşan bir maliyet fonksiyonunu en küçükleyerek atanması şeklinde bilinmektedir. KAP, tesis yerleşimi dışında, klavye tasarımı, çizelgeleme, elektronik devre tasarımı ve uçakların kapılara atanması gibi çeşitli biçimlerde çalışılan bir problemdir.

Sahni ve Gonzalez [3] 1976 yılında yaptıkları çalışmada KAP'ın karmaşıklığının *NP-Zor* olduğu ortaya koymuş ve probleme ϵ yaklaşık çözüm bulmanın dahi oldukça zor olduğu açıklamıştır. Problemin son derece zor olması sebebiyle Dal-Sınır, Kesme Düzlemi gibi kesin yöntemler ancak belirli problem boyutlarına kadar makul sürelerde en iyi çözüme ulaşmaktadır. Bu sebeple, kesin çözümü garanti etmeyen fakat kısa zamanda yakın sonuca götüren metasezgisel algoritmalar geliştirilmektedir.

Kombinatorik optimizasyon problemlerinin çözümleri için geliştirilen metasezgisel yöntemler, GİB'in kullanımına uygun olarak yoğun aritmetik işlemlere ihtiyaç duymaktadır. Literatürde araştırmacılar, problemlerin çözüm sürelerinde anlamlı bir kısalma beklentisiyle çalışmalar yapmışlardır. GİB'de

yapılan bazı çalışmalar şöyledir: sırt çantası problemi [4], çizelgeleme problemi [5], araç rotalama problemi [6, 7, 8], gezgin satıcı problemi [9] [10].

Çalışmada GİB'in eş zamanlı işlem yapabilme gücünden faydalanılarak, makul sürede etkin çözümler sunan evrimsel bir algoritma önerilmiştir. Algoritma ana hatlarıyla genetik algoritma (GA) temelindedir ve seçim, çaprazlama, elitizm operatörleri kullanılmıştır. Bu operatörlerin yanı sıra algoritmanın başarısını artırmak için bir yerel arama ve mutasyon operatörleri önerilmiştir.

Algoritmanın belirli kısımları paralel işlem yapmaya uygun hale getirilerek GİB'de çalışacak şekilde ve sıralı olarak MİB'de kodlanmıştır. Sıralı ve paralel algoritmaların etkinliği literatürdeki problemler üzerinde sınanmıştır. Sıralı ve paralel algoritmaların literatürdeki problemlere uygulanması sırasında çözüm süreleri tutulmuştur. Tüm denemelerden sonra eldeki çözüm süreleri karşılaştırılmıştır. GİB'de kodlanan paralel algoritmanın ortalaması 17 kat olmak üzere 51 kata kadar sıralı algoritmadan daha hızlı çalıştığı ortaya konmuştur.

Çalışmanın ikinci bölümde karesel atama problemiyle ilgili bilgi verilmiştir. Sonraki bölümde grafik işlem birimlerine, mimarisine ve grafik işlem birimlerinin gücünü gösteren hesaplamalı örneklere yer verilmiştir. Dördüncü bölümde önerilen evrimsel algoritma açıklanmıştır. Beşinci bölümde önerilen yaklaşım literatürde yer alan test problemi üzerinde denenmiştir, sıralı ve paralel algoritmaların çözüm süreleri karşılaştırılmıştır. Altıncı ve son bölümde ise genel değerlendirmeler yapılmış ve gelecekte yapılabilecek çalışmalar ile ilgili bilgiler verilmiştir.

2. KARESEL ATAMA PROBLEMİ

Kombinatorik optimizasyon, uygulamalı matematik ve teorik bilgisayar bilimlerinde, sonlu elemanlı bir kümeden en iyi değerin bulunması ile ilgilenen bir çalışma alanıdır. Gezgin satıcı problemi, araç rotalama problemi, atama ve çizelgeleme problemleri kombinatorik optimizasyon problemlerinden bazılarıdır [11].

KAP, sıklıkla çalışılan kombinatorik optimizasyon problemlerinden biridir. Bilimsel çalışmalardaki öneminin yanı sıra üretim ve hizmet sektöründe uygulamaları da artan rekabet ortamında son derece önemlidir. Örneğin, bir işletme yeni bir fabrika kurmaya karar vermiş olsun. Başlangıçta fabrika içine yerleştirilecek tezgahların, robotların ve benzeri üretim araçların tesis içerisine uygun planlanması bir çok kalemde maliyetlerin azalmasını sağlayabilmektedir.

KAP'ın, 1957 yılında Koopmans ve Beckman [2] tarafından ilk defa ortaya atılmasından sonra, matematikçiler, bilgisayar mühendisleri, ekonomistler ve yöneylem araştırması alanında çalışan kişiler tarafından çeşitli optimizasyon problemleri olarak kullanılmıştır. Çoğunlukla tesis yerleşimi problemi olarak ele alınmasına karşın, bir elektronik devredeki ekipmanların uygun yerinin bulunması, bir hastanede kliniklerin yerleştirilmesi ve bir hava alanında uçakların kapılara atanması KAP için verilebilecek örneklerden bazılarıdır.

2.1. Matematiksel Model

KAP en bilinen şekliyle n sayıda tesisin n sayıda aday yere akış ve uzaklıktan oluşan bir maliyet fonksiyonunu en küçükleyerek atanmasıdır [12]. Başlangıçta nxn boyutlu tesisler arası akış matrisi $F = (f_{ij})$ bulunmaktadır. Burada f_{ij}, i tesisinden j tesisine olan akışı göstermektedir. Akış matrisinin yanı sıra yine nxn boyutlu aday yerler arası uzaklık matrisi $D = (d_{kl})$ bulunmaktadır. Burada d_{kl}, k aday yerinin l aday yerine olan uzaklığını göstermektedir. Problemin matematiksel modeli izleyen şekilde verilmektedir.

$$\min z = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n} f_{ij} d_{kl} x_{ik} x_{jl} \qquad (1.1)$$

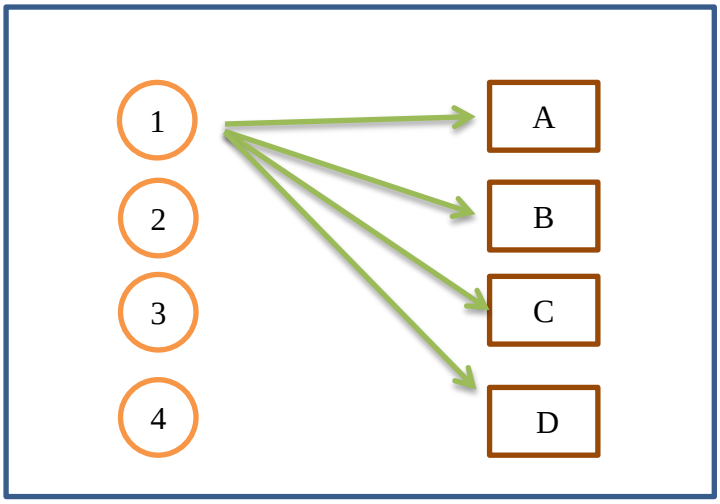
$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = \overline{1,n} \qquad (1.2)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = \overline{1,n} \qquad (1.3)$$

$$x_{ij} \in \{0,1\}\ 1 \leq i,j \leq n \qquad (1.4)$$

Amaç fonksiyonu, (1.1) denkleminde olduğu gibi, *i* tesisinin *k* yerine ve *j* tesisinin *l* yerine atanmasından oluşan *i*-*j* tesisleri arasındaki akış ile *k*-*l* yerleri arasındaki uzaklık çarpımlarının toplamını en küçüklemek şeklinde ifade edilir. (1.2) denklemi, her yere yalnızca bir tesis ve (1.3) denklemi her tesisin yalnızca bir yere atanmasını sağlayan kısıtlardır.

Kısıtlar altında amaç fonksiyonu en küçüklendiğinde elde edilen hangi tesisin hangi aday yere atandığını gösteren çözüm, bir permütasyon olarak da ifade edilebilmektedir. Şekil 2.1'de verilen örnekte dört tesis 'A', 'B', 'C' ve 'D' aday yerlerine atanmaktadırlar. Problemin çözümünde '1' numaralı tesis 'C' yerine, '2' numaralı tesis 'A' yerine, '3' numaralı tesis 'D' yerine ve '4' numaralı tesis 'B' yerine atanmış olsun. Bu durumda elde edilen çözüme karşı gelen permütasyon 2,4,1,3 olur.



Şekil 2.1. Tesislerin yerleşimi

2.2. Karmaşıklık

Kombinatorik optimizasyon problemlerinin çözüm uzayı problemin boyutundaki doğrusal artışa karşın hızla büyümektedir. Örneğin *n* boyutlu bir KAP için *n!* kadar olası çözüm bulunmaktadır. Bu durumu vurgulamak için, tarihte bir hikaye yer almaktadır. Satrancı bulan kişi krala oyunu sunduğunda kraldan karşılık olarak buğday istemektedir. Elinde bulunan tek bir buğday tanesini ilk kareye yerleştirmiş ve kraldan satranç tahtasında kalan 63 karenin her biri için iki kat buğday talep etmiştir. Kralda bu talebi gülümseyerek karşılayarak hemen kabul etmiştir. Satrancı bulan kişinin istediği buğday adedi $2^1 + 2^2 + \cdots + 2^{63} - 1$ kadardır. Bu toplam $1.84*10^{19}$ rakamına karşılık gelmektedir. Bir buğday tanesinin yaklaşık olarak 0.03 gram olduğu düşünüldüğünde bu buğday tanelerinin $5.53*10^{11}$ tona karşılık gelmektedir. Dünyada yıllık buğday üretimi yaklaşık olarak 680 milyon tondur. Buradan istenilen miktarın dünyada 800 yılı aşkın bir üretime karşılık geldiği anlaşılmaktadır. Bu örnekteki artış, $2^k + 2^{k-1} + \cdots + 2^1 - 1$ şeklindedir. Oysa KAP'ın çözüm uzayı için geçerli olan *n!* şeklindeki bir artış, örnektekine göre çok daha hızlıdır.

KAP'ın çözümü, problemin yapısından kaynaklı amaç fonksiyonunun doğrusal olmayışı sebebiyle oldukça zordur. Literatürde yapılan teorik çalışmalarda bu durumu kanıtlamaktadır [12]. Sahni ve Gonzalez [3], 1976 yılında yaptıkları çalışmada KAP'ın çözümü ve çözüme yaklaşmanın karmaşıklığıyla ilgili ortaya sonuçlar koymuşlardır. Bu sonuçlara göre KAP *NP-Zor* türünde bir kombinatorik optimizasyon problemidir. Bununla beraber, problemin çözümüne ulaşmak ve hatta çözüme ϵ yaklaşmanın dahi polinom sürede mümkün olmadığı belirtilmiştir.

2.3. Çözüm Yöntemleri

KAP'ı çözmek için yöntem geliştirme arayışlarında ilk durak doğrusallaştırma yöntemleri olmuştur. Doğrusallaştırma yöntemleri ile problem 0-1 doğrusal programlamaya uygun hale getirilmiştir. Böylelikle problem karma

tam sayılı programlama yöntemleriyle çözülebilecek hale gelmiştir [12]. Ancak doğrusallaştırma yöntemleri çok sayıda değişkeni ve kısıtı modele eklediği için üzerinde çalışılmasını zor kılmıştır. İlk olarak Lawler'ın [13], sonrasında Kaufman ve Broeckx'in [14], Frieze ve Yadegar'ın [15], Adams ve Johnson'ın [16] doğrusallaştırması literatürde en çok bilinen doğrusallaştırma yöntemlerdir.

Doğrusallaştırmanın yanı sıra, araştırmacılar KAP'ın çözümü için çok yüzlü bir tanım ortaya koymuşlardır. KAP politopu üzerinde son zamanlarda yapılan çalışmalarda, problem grafik problemi dönüşümü [17] ve afin dönüşümleriyle [18] probleme çözüm aranmıştır.

Kesin çözüm yöntemlerinin yanı sıra, probleme özgü geliştirilen kesin çözümü garanti etmeyen sezgisel algoritmalar ve metasezgisel algoritmalar bulunmaktadır. Çözüm yöntemlerine ayrıntılarıyla Çela'nın [19] 1998 yılında yaptığı çalışmada yer almaktadır. KAP için geliştirilen kesin ve sezgisel yöntemlerden bazıları izleyen bölümlerde açıklanmaktadır.

2.3.1. Kesin çözüm yöntemleri

Bir kombinatorik optimizasyon problemi için geliştirilen kesin yöntemler, problem için bütünsel en iyi çözümü bulmayı garanti ederler. KAP'ın çözümü için geliştirilen kesin algoritmalardan bazıları, dal sınır algoritması, kesme düzlemi ve subgradient yöntemidir.

Dal sınır algoritması

Dal sınır algoritması, kombinatorik optimizasyon problemlerinin bir çoğunun çözümü için üzerinde çalışılan bir yöntemdir. Literatürde KAP'ın çözümü için çeşitli dal sınır algoritmaları bulunmaktadır. Bunlardan en etkili olanı Gilmore-Lawler'ın geliştirdiği dal sınır algoritmasıdır. Bunun sebebi ise kullanılan sınır yönteminden dolayı çözüm sürelerinin daha kısa olmasıdır [12]. Hahn ve Grant [20] ve Pardalos ve ark. [21], KAP için geliştirdikleri dal sınır algoritmaları da literatürde yer alan diğer çalışmalardandır. Özellikle Pardalos ve

ark. [21] çalışmalarında KAP için en çok kullanılan test problemlerinden bazılarının optimal çözümlerini bulmuşlardır.

Kesme düzlemi yöntemi

Kesme düzlemi yöntemi, temel olarak problemi karma tam sayılı doğrusal programlama yöntemiyle çözmeye çalışmaktadır. Bazaraa ve Kirca [22] ve Balas ve Mazzola [23] çalışmalarında, başlangıçta bir sezgisel ile ilk atamaları yaptıktan sonra, problemi alt bir problem halinde tam sayılı matematiksel programlama ile çözmeye çalışmışlardır. Eğer çözüm sonunda tüm kısıtlar sağlandıysa, programı sonlandırarak aksi takdirde modele yeni kısıtlar ve değişkenler ekleyerek çözüme ulaşmaya çalışmışlardır.

Subgradient yöntemi

Subgradient yöntemi, dışbükey enküçükleme problemlerini çözmek üzere geliştirilmiş iterasyon tabanlı bir yöntemdir. Yöntem, özellikle türevlenemeyen amaç fonksiyonuna sahip ve kısıtsız optimizasyon problemleri üzerinde etkin çalışmaktadır. Üstün [24] tarafından, KAP'ın çözümü için yapılan çalışmada yönteme farklı özellikler katılarak Lagrange fonksiyonu yardımıyla KAP ikil probleme dönüştürülmüştür. Böylelikle problemde bulunan kısıtlar genişletilmiş Lagrange fonksiyonunun yardımıyla amaç fonksiyonuna dahil edilerek, KAP kısıtsız bir optimizasyon problemine dönüştürülmüştür. Ayrıca, problemde bulunan 0-1 tam sayılı değişkenler sürekli hale getirilmiştir. Yöntemin avantajları arasında, dışbükeylik veya türevlenebilirlik aramaması ve Lagrange fonksiyonunun herhangi bir ceza parametresi içermemesi yer almaktadır. Her iterasyonda kısıtsız dışbükey ve doğrusal olmayan bir problemle karşılaşılması ise yöntemin uygulamasındaki zorluk olarak vurgulanmıştır. Gasimov ve Üstün [25] tarafından 2007 yılında sunulan çalışmada yöntem modifiye edilerek büyük boyutlu problemler üzerinde uygulanmıştır.

2.3.2. Sezgisel yöntemler

Her ne kadar kesin yöntemlerle ilgili çalışmalar ve önemli gelişmeler olsa da KAP'ın ve diğer bir çok kombinatorik optimizasyon probleminin doğasından kaynaklı zorluğu araştırmacıları sezgisel yöntemler üzerinde çalışmaya yöneltmiştir. Problemin boyutunun artması ile birlikte problemin kesin yöntemlerle çözümü güçlü bilgisayar sistemlerine ihtiyaç duymakta ve çözüm de makul bir sürede gerçekleşememektedir. Bu sebeple, araştırmacılar makul sürelerde kaliteli sonuçlar veren sezgisel yöntemler ile çalışmışlardır. KAP için önerilen bazı sezgisel yöntemler şöyledir: kurma yöntemi, sınırlı sayımlama yöntemi ve iyileştirme yöntemidir.

Gilmore [26] tarafından geliştirilen kurma yöntemi, *i* tesisini *j* yerine atayarak boş bir permütasyonu sırayla doldurarak işlemektedir. Devamında tüm alternatif permütasyonları denemektedir. Kurma yöntemine dayalı ilk sezgisellerden biri Buffa ve ark. [27] tarafından geliştirilen CRAFT algoritmasıdır. Diğer bir sezgisel olan sınırlı sayımlama yöntemi, başlangıçta verilen bilgiler eşliğinde dal-sınır yöntemine benzer şekilde çalışmakta ve bu bilgilerin kalitesine göre algoritmanın çalışmasının hemen ardından iyi sonuçlar verebilmektedir. İyileştirme yöntemi ise yerel arama algoritmaları sınıfında gösterilebilir [28]. Bu algoritma, temel olarak komşu çözümlere bakarak ikili ve mümkün değişikliklerle daha iyi çözümlere gitmektedir.

2.3.3. Metasezgisel yöntemler

Metasezgisel yöntemler, optimizasyon problemleri için geliştirilmiş kesin çözümü garanti etmeyen fakat kısa sürede en iyi çözüme yakınsayan tekniklerdir. Bu yöntemler, sezgisel yöntemlerin çoğunda olduğu gibi tek bir tip optimizasyon problemi için geliştirilmemiştir. Kesin çözüm veren algoritmaların makul bir sürede çözüm vermediği durumlarda sıklıkla kullanılırlar. Metasezgiseller farklı alanlarda birçok optimizasyon problemine uygulanabilmektedir:

- Mühendislik tasarımları, elektronikte topolojik ve yapısal optimizasyon, telekomünikasyon, aerodinamik, akışkanlar dinamiği,

- Biyoenfotmatikte yapay zeka ve veri madenciliği uygulamaları, hesaplamalı biyoloji ve finans,
- Sistem modelleme, fizik, kimya ve biyolojide benzetim, sinyal ve resim işleme,
- Rotalama problemleri, çizelgeleme ve üretim problemleri, lojistik ve taşıma problemleri, tedarik zinciri yönetimi vb. [29].

Bu algoritmalar her ne kadar bütünsel en iyiyi bulmayı garanti etmese de birçok alanda büyük boyutlu ve karmaşık problemlere yapılan uygulamalarda, güvenilirlik ve geçerliliğini göstermektedir.

KAP için kesin yöntemlerle problemin boyutu 15'e kadar en iyi çözüm bulunabilmektedir [29]. Problemin boyutunun büyümesiyle birlikte kesin çözüm yöntemlerinin kullanımını güçleşmektedir. Her ne kadar gelişen teknoloji daha güçlü bilgisayar olanakları sunsa da problemlerin birçoğunun çözümü polinom sürede gerçekleşememektedir. Bu sebeple literatürde KAP ile ilgili birçok metasezgisel algoritma geliştirilmiştir.

Tavlama benzetimi, kombinatorik optimizasyon problemlerinin çözümünde kullanılan, demirin tavlanmasındaki fiziksel durum ve istatistiksel bir mekanizmanın bir araya getirilmesiyle oluşan, iterasyon temelli metasezgisel bir algoritmadır. 1983 yılında Kirkpatrick ve ark. [30] tarafından sunulan algoritma, KAP literatüründe Burkard ve Rendl [31], Wilhelm ve Ward [32], Abreu ve ark. [33] ve daha birçok çalışmada ele alınmıştır.

Karınca kolonileri optimizasyonu, karıncaların yiyeceğe giden yollarını oluşturmalarından esinlenerek ortaya atılmıştır. Popülasyon temelli olan bu algoritma, Stützle ve Dorigo [34] ve Dorigo ve ark. [35] gibi çalışmalarda KAP için ele alınmıştır.

Glover [36], 1989 yılında tamsayılı programlama uygun problemleri çözmek üzere yasaklı arama algoritmasını geliştirmiştir. Bir tür yerel arama algoritması olan yasaklı arama, temel olarak komşu çözümlere bakarak bir yasak listesi oluşturmakta ve yerel en iyi noktalardan kaçma mekanizması ile bütünsel en iyi çözüme ulaşmak için tasarlanmıştır. KAP literatüründe büyük ilgi uyandıran bu metasezgisel algoritma, Skorin [37], Taillard [38], Misevicius ve

Ostereika'nın [39] çalışmalarında kendine yer bulmuş ve iyi performans gösterdiği bu çalışmalarda vurgulanmıştır.

Darwin'in evrim teorisinden yola çıkarak J.H. Holland [40] GA'yı geliştirmiştir. Popülasyon temelli evrimsel bir algoritma olan GA seçim, çaprazlama, mutasyon gibi operatörlerden oluşmaktadır. 1980'li yıllardan sonra optimizasyonla ilgili neredeyse her alanda kullanılan GA, KAP literatüründe de kendine geniş yer bulmuştur. Kochhar ve ark. [41] ve Drezner'in [42] yaptığı çalışmalar bunlardan bazılarıdır.

Bunlara ek olarak, ilk olarak Li ve ark.'nın [43] üzerinde çalıştığı GRASP algoritması birçok problemin çözümünde kullanılmıştır. Literatürde KAP için geliştirilen metasezgisel algoritmalar, paralel genetik algoritmalar, yapay sinir ağları, parçacık sürü optimizasyonu gibi daha birçok örnek gösterilebilmektedir.

2.4. Genetik Algoritmalar ve Karesel Atama Problemi

Bu çalışmada sunulan evrimsel yaklaşımda genetik algoritma operatörleri benzeri operatörler kullanılmıştır. Bu sebeple, çalışmanın bu alt bölümünde, KAP için geliştirilen benzeri algoritmalar ile ilgili son yıllarda yapılan çalışmalar incelenmektedir.

2000 yılında Ahuja ve ark. [44] çalışmalarında açgözlü bir GA sunmuşlardır. Bu yaklaşımda, başlangıç popülasyonu GRASP ile belirlenmekte sonrasında GA operatörleri rol almaktadır. Çaprazlama operatörüne göndermek üzere bireyler seçilmektedir. Farklı çaprazlama teknikleriyle yeni bireyler oluşturulmakta ve küçük olasılıklarla mutasyona uğratılmaktadır. Çalışmada klasik operatörlerin haricinde her tekrarda popülasyona yeni bireyler katılması ve bununla birlikte popülasyon içi çeşitliliğin artırılması için göç operatörü kullanılmıştır. Bu aşamadan sonra ise 2-opt yerel arama algoritmasından faydalanılmıştır.

Yuan ve ark. [45] 2002 yılında melez bir genetik algoritma sunmuşlardır. Çalışmada GA operatörlerinden mutasyon operatöründe yerel arama uygulanmıştır. Bu çalışmayı farklı kılan durum, amaç fonksiyonu değeri hesaplanırken normalleştirme yapılmış olmasıdır.

El-Baz [46], 2004 yılındaki çalışmasında problemi tesis yerleşimi şeklinde ele almıştır. Klasik GA operatörlerinden oluşan bir yöntem geliştirilmiş ve literatürde geçen gerçek hayat problemleri üzerinde uygulanmıştır.

Drezner [47], 2005 yılında geliştirdiği yöntemi, birleştirilmiş GA olarak isimlendirmiştir. Çalışmada geliştirilen yöntemin, hem hızlı hem de etkin çalışıyor olması vurgulanmıştır. Algoritmada farklı bir çaprazlama yöntemi kullanılmıştır. Bu yöntem, popülasyondan seçilen iki bireyin bir araya gelmesiyle, yeni bir birey elde etme şeklinde oluşturulmuştur. Elde edilen bireyin amaç fonksiyonu değeri çaprazlanan bireylerden daha iyiyse yeni birey sonraki nesle aktartılmıştır. Aksi taktirde, daha kötü amaç fonksiyonu değeri olan birey popülasyonda bulunmuyor ise yeni nesle aktarılmıştır.

Wu ve Ji [48], 2007 yılında yaptıkları çalışmada popülasyon içi çeşitliliği yüksek seviyede tutan bir GA sunmuşlardır. Böylelikle, algoritmaların birçoğunun genel sıkıntısı olan erken yakınsama sorununda önemli gelişmeler kaydedildiği belirtilmiştir. Algoritmanın test problemleri üzerinde denenmesinin yanı sıra, literatürde yapılan benzer çalışmalar ile karşılaştırmalar yapılarak yöntemin etkinliği ortaya konulmuştur.

Drezner [49], 2008 yılında yaptığı çalışmada birbirine benzer birçok melez GA ile hesapsal sonuçlar elde etmiştir. Çalışmada, yerel arama algoritması olarak yeni basit bir yasaklı arama yöntemi sunmuştur. Üzerinde çalışılan tüm melez yöntemler test problemleri ile birbirleriyle karşılaştırılmış ve farkları ortaya konmuştur.

Ramkumar ve ark. [50], 2009 yılında yapmış oldukları çalışmada literatüre yeni ve hızlı çalışan melez bir yaklaşım sunmuşlardır. Bu çalışmada sunulan yöntem, GA benzeri operatörlerden oluşmaktadır. Popülasyonda bulunan birey sayısı ikidir ve bu başlangıç bireyleri problem büyüklüğünde bir permütasyon oluşturacak şekilde rassal olarak oluşturulmaktadır. Sonrasında bu iki birey çaprazlanmakta ve mutasyon operatörüne gönderilmektedir. Mutasyondan sonra bireylere 2-opt yerel arama algoritması uygulanmakta ve süreç bu şekilde tekrar etmektedir. Algoritma MİB'lere uygun şekilde kodlanmış ve literatürde yer alan test problemlerinin büyük çoğunluğuna uygulanmış çözüm süreleri ve kaliteleri rapor edilmiştir.

2009 yılında yapılan bir başka çalışmada James ve ark. [51] paralel yasaklı arama algoritması sunmuşlardır. Özellikle MİB'nin kendi içinde bilgi paylaşımı sayesinde çözüm sürelerinde kısalmalar elde edildiği vurgulanmıştır. Paralel hesaplamanın faydalarından ve kombinatorik optimizasyon problemlerinde paralel hesaplama üzerine çalışmaların olumlu getirisinden bahsedilmiştir.

Misevicius [52], 2012 yılında yayınlanan çalışmasında, GA'dan farklı olarak iterasyon tabanlı yasaklı arama algoritması sunmuştur. Çalışmada orta ve büyük ölçekli literatür problemleri ele alınmıştır. Bu çalışmanın en önemli noktası literatürde bilinen zor problemlerden olan *tai50a*, *tai80a* ve *tai100a* için bilinen en iyi sonuçların geliştirilmiş olmasıdır. Ancak yeni elde edilen çözümler raporlanmamıştır.

3. GRAFİK İŞLEM BİRİMLERİ

Grafik işlem birimleri (Graphics Processing Units GPUs), ekran kartları üzerindeki işlemcilerdir. Bu işlemciler, MİB'e göre daha az güçte fakat daha fazla sayıdadır. Son yıllarda piyasaya sürülen ekran kartlarının üzerinde oldukça gelişmiş özellikleriyle bulunan GİB, genel olarak bilgisayar ortamında görselleştirme aracı olarak kullanılmaktadır. Öte yandan GİB yalnızca görselleştirme işlemlerinde değil, hesaplamalı bilimlerde de yükselen bir eğilimle ilgi görmektedir.

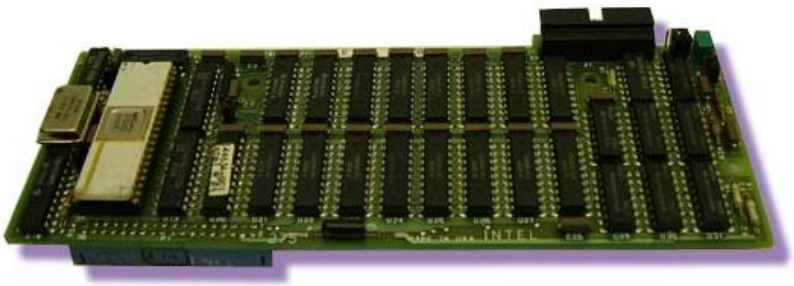
Optimizasyon problemlerinin çözüm sürelerinin kısalması motivasyonuyla, metasezgisel algoritmaların GİB üzerinde geliştirilmesi yaygınlaşmaktadır. Bununla birlikte, söz konusu teknolojinin yeni oluşu, geleneksel programlama bilgi ve becerilerinin doğrudan kullanılmasının önünde bir engeldir. Bu teknolojiyi kullanarak uygulama geliştirmek için GİB mimarisinin çok iyi şekilde anlaşılması gereklidir. Bu sebeplerden dolayı, literatürde kombinatorik optimizasyon problemlerinin GİB'de metasezgiseller ile çözümüne yönelik sınırlı sayıda çalışma bulunmaktadır.

Çalışmanın bu bölümünde GİB'in kısaca tarihçesi, mimarisi ve geliştirme ortamı anlatılmıştır. Ayrıca, özellikle kombinatorik optimizasyon problemlerinin çözülmesi bakış açısından GİB üzerinde yapılan çalışmalar ile ilgili literatür taraması sunulmuştur.

3.1. Tarihçesi

1960'lı yıllarda Boeing firmasının, kokpit tasarımının görselleştirmesi için geliştirmiş olduğu ekran kartları, bugün gelişmiş bilgisayar oyunlarından, üç boyutlu modellemeye kadar bir çok alanda önemli rol oynamaktadır. Bildiğimiz şekliyle ilk ekran kartı (Şekil 3.1), IBM tarafından 1981 yılında renksiz grafikleri destekler halde piyasaya sürülmüştür. Fakat ekran kartları üzerinde çalışan ilk işlemci, yine IBM tarafından 1984 yılında geliştirilmiştir [1]. Dolayısıyla bu geliştirme, günümüz GİB'i için büyük bir adım olmuştur. Özellikle bilgisayar oyunlarının yüksek talep görmesi, nVIDIA, ATI, Matrox gibi markaların ekran

kartlarının geliştirilmesi üzerine odaklanmalarını sağlamıştır. Başlangıçta tek bir GİB'e sahip olan ekran kartları üzerinde bugün yüzlercesi bulunabilmektedir.



Şekil 3.1. IBM tarafından çıkarılan ilk ekran kartı

GİB üzerinde hesapsal işlemlerin yapılması, her ne kadar 2003 yılına dayansa da 2006 yılında nVIDIA firmasının CUDA™ geliştirme ortamını tanıtması bu alanda milat olarak kabul edilebilir. O günden beri binlerce uygulama ve makale literatüre katılmış, üç yüz milyonu aşkın bilgisayar ise CUDA kullanılabilir hale gelmiştir [53]. Günümüzde sadece bilgisayarlarda değil, televizyonlarda, akıllı cep telefonlarında ve tabletlerde de GİB teknolojisin yaygın şekilde kullanıldığı görülmektedir.

3.2. Mimarisi

Bu alt bölümde, tez çalışmasında kullanılan nVIDIA GTX580 yonga kümesinin mimarisi üzerinde durulmuştur.

nVIDIA bugüne kadar yonga kümeleri için üç farklı mimari sunmuştur. İlki 2006 yılında sunulan G80, ikincisi yakın zamanlarda birçok ekran kartının mimarisini oluşturan GT200 ve sonuncusu 2011 yılında sunulan yeni nesil Fermi teknolojisidir. Üç mimari de GİB üzerinde programlama yapmaya olanak sağlamaktadır. CUDA, C, C++ ve Fortran gibi programlama dillerinde yazılan kodların GİB üzerinde paralel olarak işlenmesini sağlayan yazılımdır. Mimarinin daha iyi anlaşılabilmesi için bilinmesi gereken bazı terimler şöyledir:

CUDA Çekirdeği (CUDA Core): Yonga kümesi üzerinde yer alan işlemciler.

Akış İşlemcisi (Streaming Multiprocessor): CUDA çekirdeklerinin kümelendiği işlemci.

Cihaz Belleği (Device Ram): Yonga kümesinin bağlı olduğu bellek.

Paylaşılan Bellek (Shared Memory): Yonga üzerindeki bilgi paylaşımını sağlayan bellek.

Kernel: Yazılan programın GİB üzerinde işlenmesini sağlayan kod bloğu.

Thread: GİB'de işlenen en küçük iş parçacığı.

Blok: Threadlerden oluşan iş parçacığı topluluğu.

Grid: Bloklardan oluşan iş parçacığı topluluğu.

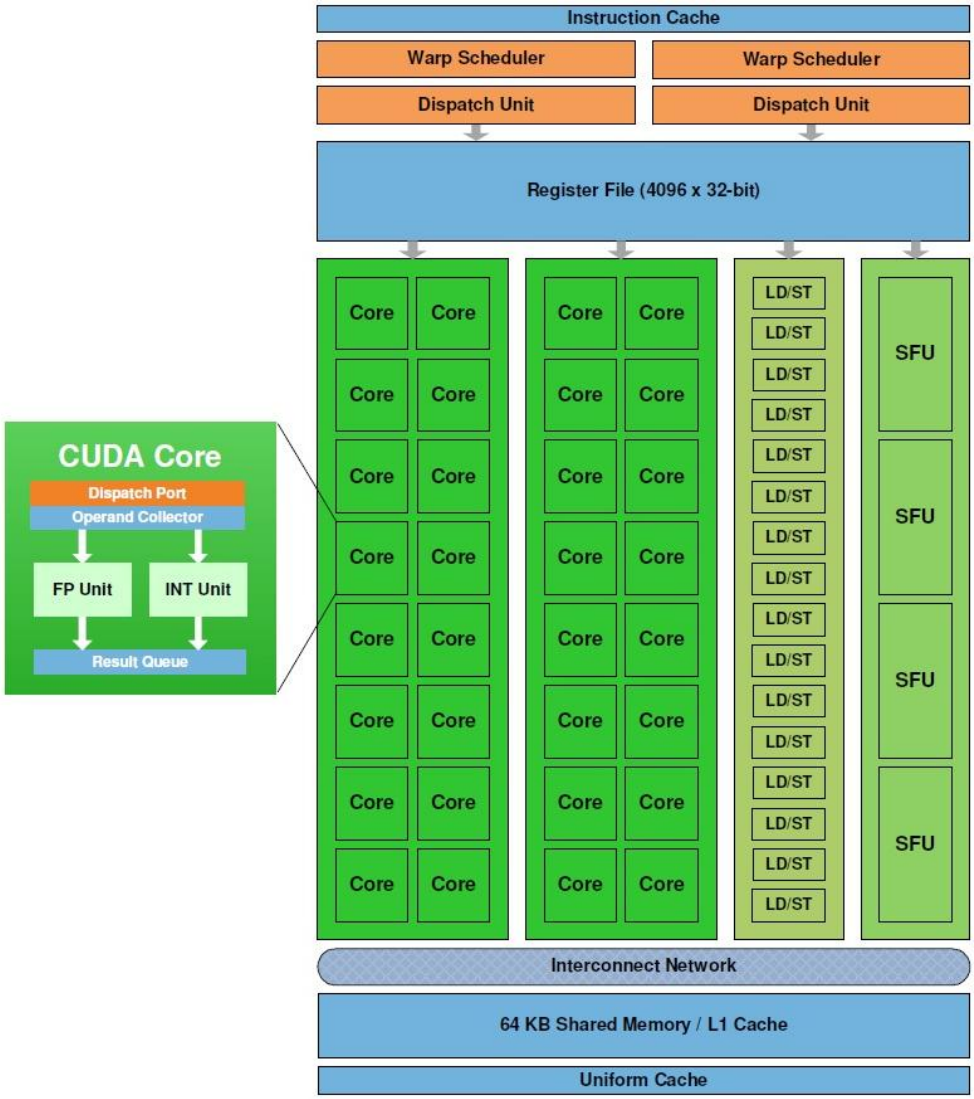
G80 mimarisinde 681 milyon transistör ve 128 CUDA çekirdeği bulunurken, Fermi mimarisinde yonga üzerinde 3 milyar transistör ve 512 CUDA çekirdeği bulunmaktadır. Tablo 3.1'de mimarilerin karşılaştırması görülmektedir.

Tablo 3.1.GİB mimari karşılaştırması

Mimari	**G80**	**GT200**	**Fermi**
Transistör	681 Milyon	1.4 Milyar	3.0 Milyar
CUDA Çekirdeği	128	240	512

Yukarıda da bahsedildiği gibi Fermi teknolojisinde 512 CUDA Çekirdeği bulunmaktadır. Bu çekirdekler her birinde 32'şer tane olmak üzere 16 adet akış işlemcisi içerisinde kümelenmiştir. Fermi, 6 GB'a kadar kullanımlarda bellek desteği vermektedir. 64K paylaşılan hafızayla, bloklar arası bilgi paylaşımına olanak sağlamaktadır [54].

Şekil 3.2'de akış işlemcilerinden birinin yapısı ve bir CUDA çekirdeğinin detayları bulunmaktadır. 32 adet çekirdek barından akış işlemcisinin, özel matematiksel işlemleri yapmak üzere kullanılan dört adet SFU (Special Function Unit) birimi bulunmaktadır. Her bir çekirdekte ise tam sayılı işlemler için INT (Integer Unit), ondalıklı işlemler için FP (Floating Point Unit) birimleri bulunmaktadır.

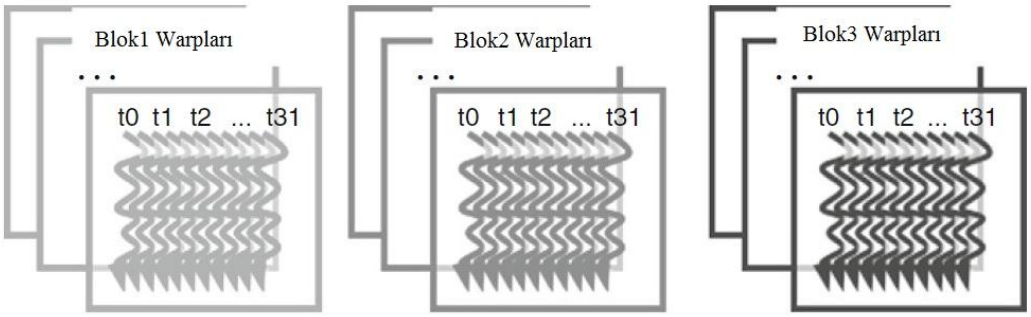


Şekil 3.2. Akış işlemcisi ve cuda çekirdeği [54]

CUDA, sıklıkla kullanılan birçok programlama dili ile Microsoft Visual Studio ortamında ya da diğer derleyici ortamlarında program yazılmasını mümkün kılmaktadır. Yazılan kodların bir kısmı hala MİB üzerinde işlenirken, bir kısmı da kernel sayesinde GİB üzerinde işlenmektedir. Bir kernel çalıştırılmadan önce, grid yapısına dair parametrelere ihtiyaç duymaktadır. Grid yapısı, iş parçacıklarının GİB üzerinde eş zamanlı yürütülmesi açısından büyük önem taşımaktadır. Fermi mimarisinde her akış işlemcisinde en fazla sekiz blok eş zamanlı olarak çalışabilmektedir. Eş zamanlı işlenebilecek en fazla thread sayısı ise 1536'dır. Buradan bir akış işlemcisinde CUDA çekirdeği başına 48 thread'i eş zamanlı işlenebildiği ortaya çıkmaktadır. Her blokta bulunabilecek thread sayısı ise 1024

ile sınırlandırılmıştır. Örneğin bir akış işlemcisinde 384'er thread içeren 4 blok ya da 192'şer thread içeren 8 blok işlenebilir. Ancak 96'şar thread içeren 16 bloklu bir yapı, akış işlemcisindeki en fazla blok sayısı aşıldığı için işlenemez. Aynı zamanda 1536 thread bulunduran tek bloklu bir yapıda blok başına en fazla thread sayısı aşıldığı için işlenemez.

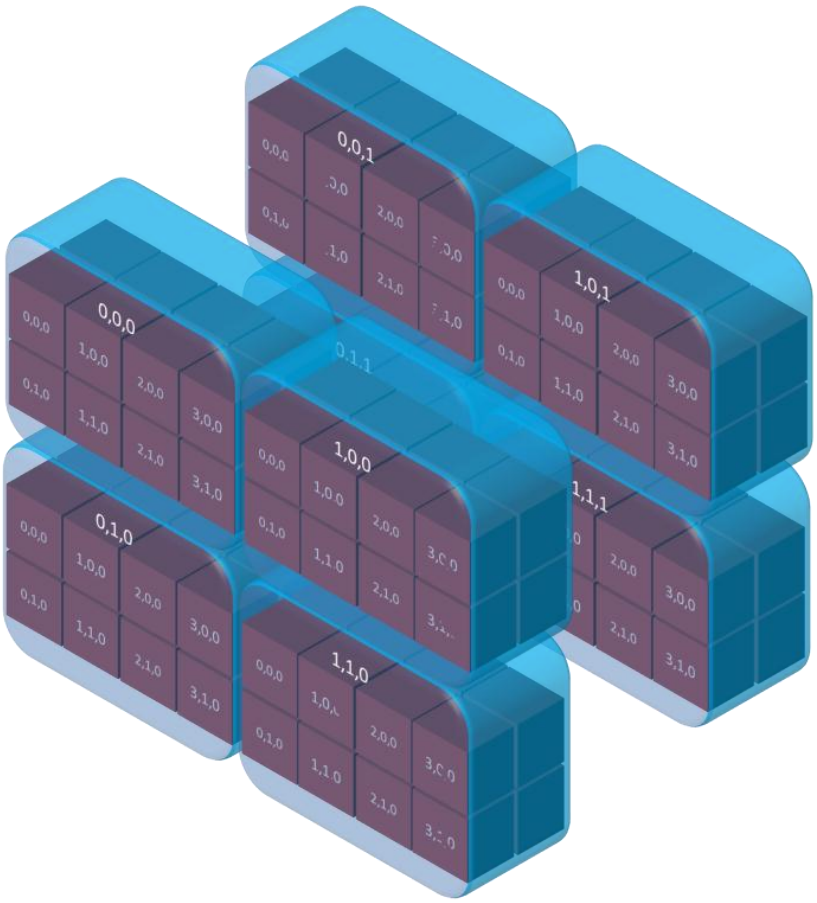
Bir grid yapısı tanımlandığında threadler Warp çizelgeleyeci (Warp Scheduler) tarafından 32'şerli gruplar olarak akış işlemcisine gönderilmektedir. Şekil 3.2 incelendiğinde Fermi mimarisinde her akış işlemcisinde ikişer adet Warp çizelgeleyici olduğu görülmektedir. GİB üzerinde programlama yaparken Warp çizelgeliyecelerin çalışma mantığını göz önünde bulundurmak, yazılan programın etkinliği için büyük önem taşımaktadır. Şekil 3.4'te örnek olarak 3 blok ve her blokta 32 thread içeren bir grid yapısında warpların nasıl çalıştığını gösterilmektedir. Çizelgeleyici threadleri CUDA çekirdeklerine dağıtmaktadır. Çekirdek 48 threade kadar, her thread için ayrılmış registerları kullanarak işlemleri yapmaktadır.



Şekil 3.3. Warp çizelgeleyici

Şekil 3.4'te bir kernel için örnek bir grid yapısı bulunmaktadır. Burada gridde x, y ve z eksenlerinde toplam 8 blok bulunmaktadır. Her blokta toplam 16 thread bulunmaktadır. Her akış işlemcisinde en fazla 8 blokta toplam 1536 thread işlenebilmekte olduğunu göz önünde bulundurarak, örnekte 8 blok ve 128 thread bulunduğundan yalnızca bir akış işlemcisinin kullanıldığı anlaşılmaktadır. Buradan örnek grid yapısının bir akış işlemcisinin %8.3'ünü, 512 CUDA çekirdeğine sahip bir grafik kartının ise %0.5'ini kullanmaktadır. Bu grid yapısı programda şu şekilde tanımlanmaktadır.

```
dim3   dimGrid(2,2,2)
dim3  dimBlock(4,2,2)
Kernel1<<<dimGridT,dimBlock>>>(dev_pointers)
```



Şekil 3.4. Grid yapısı örneği

Grid yapısı, en fazla (65535,65535,65535) blok ve en fazla her blok için (1024,1024,64) thread içerecek şekilde tanımlanabilmektedir. GİB işlem birimleri üzerinde programlama yaparken en çok dikkat edilmesi gereken durumlardan biri de hafıza kontrolüdür. Paylaşılan bellekte tanımlı değişkenlerin ömrü kernelin işlenmesine kadar sürmektedir ve bu hafıza kısmı oldukça hızlıdır. Ayrıca, bu bellek üzerinde tanımlanan değişkenlere tüm bloklar tarafından ulaşılabilmektedir. Tüm program boyunca bir defa tanımlanacak değişkenlerin sabit bellekte (constant memory) ya da ekran kartının kendi belleğinde tanımlanması, o değişkenin program çalıştığı sürece ömrünü korumasını sağlamaktadır.

GİB'in mimarisinin daha iyi anlaşılabilmesi için şu şekilde bir analojiye değinilebilir. Örneğin, bir okulda 16 sınıf, her sınıfta 32 öğrenci ve her öğrencinin 48 adet birbirine benzer basit problemi eş zamanda çözebilme yeteneği bulunsun. Bir öğrenciyi bir CUDA çekirdeğine benzetecek olursak, sınıf bir akış işlemcisini okul ise mimarinin tümünü temsil etmektedir.

3.3. Çalışma Ortamı

nVIDIA yongalı ekran kartları üzerinde çalışma yapabilmek üzere geliştirilen CUDA'nın 2012 Nisan ayı içerisinde 4.2 versiyonu çıkmıştır. GİB üzerinde çalışmalarda yaşanan zorlukları her geçen gün azaltan bu sistem C, C++ ve Fortran gibi programlama dilleriyle birlikte kullanılabilmekte ve Visual Studio ortamında programlar geliştirilebilmektedir. GİB için yazılan program kısmı CUDA C olarak bilinen C benzeri bir programlama dilidir.

GİB üzerinde çalışacak bir kerneli tasarlarken çok önemli iki nokta bulunmaktadır. Bunlardan biri döngüler, ikincisi ise koşul durumlarıdır. Kernel içerisinde döngüleri ve koşul durumları kullanmak, GİB üzerinde yapılan işlemin sürelerinde ciddi artışlara sebep olabilmektedir. Kernel içerisinde döngülerden kaçınmak için genelde izlenen yol, thread ve blokların indekslerini ve boyut bilgilerini kullanmaktan geçmektedir.

3.4. Örnek Hesaplamalı Uygulamalar

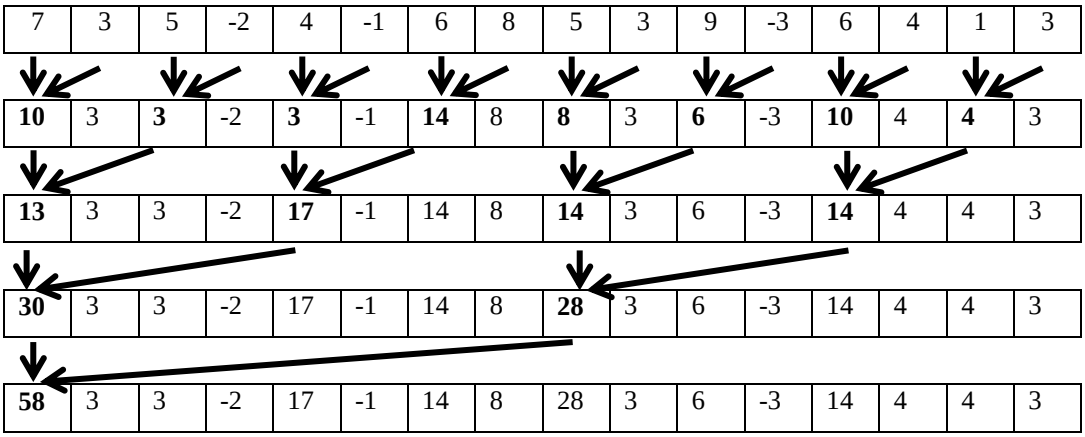
GİB üzerinde etkin bir program geliştirmenin bazı önemli noktaları şöyledir:

- Paralel yapıya uygun bir algoritma geliştirme,
- Büyük boyutlu dizilerle çalışma,
- Tüm GİB'i kullanabilme [55].

GİB üzerinde etkin bir çalışma, eş zamanlı işlemeye uygun yapıda tasarlanmış bir algoritmanın yanı sıra, parçalara bölünmeye elverişli ve büyük boyutlu veri yapılarından geçmektedir. GİB'nin gücünden tam anlamıyla faydalanmak için CUDA çekirdeklerinin tümünü meşgul etmek ve buna uygun program geliştirmek gerekmektedir. İzleyen bölümdeki örnekler bu durumu vurgulamaktadır.

3.4.1. GİB üzerinde paralel toplama

nVIDIA araştırmacıları, 2010 yılında oluşturdukları raporda [55] çok büyük boyuttaki bir dizideki tam sayıların toplanmasının GİB üzerinde yapılabilecek geliştirmeler üzerinde durmuşlar ve çarpıcı sonuçlar ortaya koymuşlardır. Diziyi bir ağaç benzeri toplayarak işe başlayan araştırmacılar, altı farklı geliştirme ile GİB üzerinde yapılan bir çalışmanın ne derece etkin olabileceğini vurgulamışlardır.

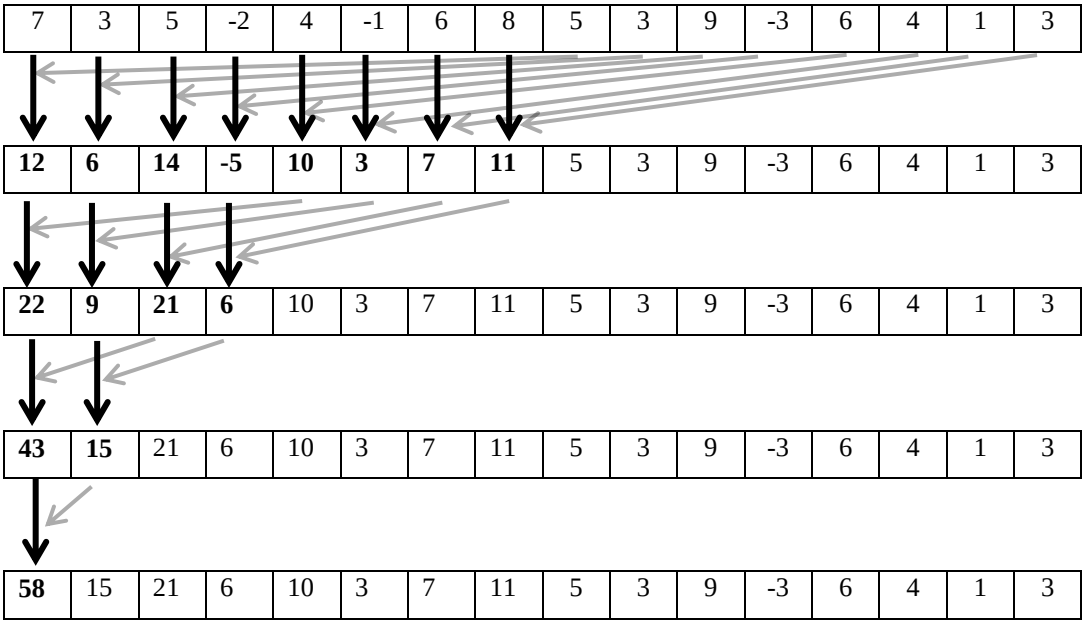


Şekil 3.5. Paralel toplama versiyon 1

Şekil 3.5'te çalışmanın ilk adımını özetleyen 16 boyutlu bir dizideki elemanların toplama işlemi gösterilmektedir. Bu işlem toplamda beş adımdan oluşmaktadır. İlk adımda sekiz paralel işlem, sonra dört, sonra iki ve son adımda toplama işleminin sonucu dizinin ilk elemanı olarak görülmektedir. Benzer şekilde çalışmanın ilk adımında 2^{22} elemandan oluşan tam sayılı bir dizinin toplamını GİB üzerinde 8.054 ms'lik bir sürede elde etmişlerdir. Toplanacak dizideki elemanların tümüne karşılık gelen bir threadIdx.x bulunmaktadır. Toplama işlemi, adımların sayısı kadar tekrar eden bir döngü ile yapılmaktadır. İkinin katları şeklinde artan bu döngüde eğer, threadIdx.x değerinin iki için modu sıfıra eşitse toplama işlemi gerçekleşmektedir.

İkinci aşamada, ilk aşamada önerilen mod alma işleminin warpların etkin bir şekilde oluşturulmasına engel olduğundan, hesaplama süresini artırdığı vurgulanmış ve yalnızca mod alma işlemi yerine bir başka yöntem getirilmiştir. Yine ikinin katları şeklinde artan döngüde dizideki elemanların indeksleri kullanılarak her adımda ortadan ikiye bölerek aynı indeksli değerlerin toplanarak

işlem yapılmaktadır. Böylelikle hesaplama süresi 3.456 ms'ye düşmüştür. Şekil 3.6'teki gibi gerçekleştirilen işlemin paylaşılan bellek kullanılarak yapılan versiyonunda ise hesaplama süresi 1.722 ms'dir.



Şekil 3.6. Paralel toplama versiyon 2

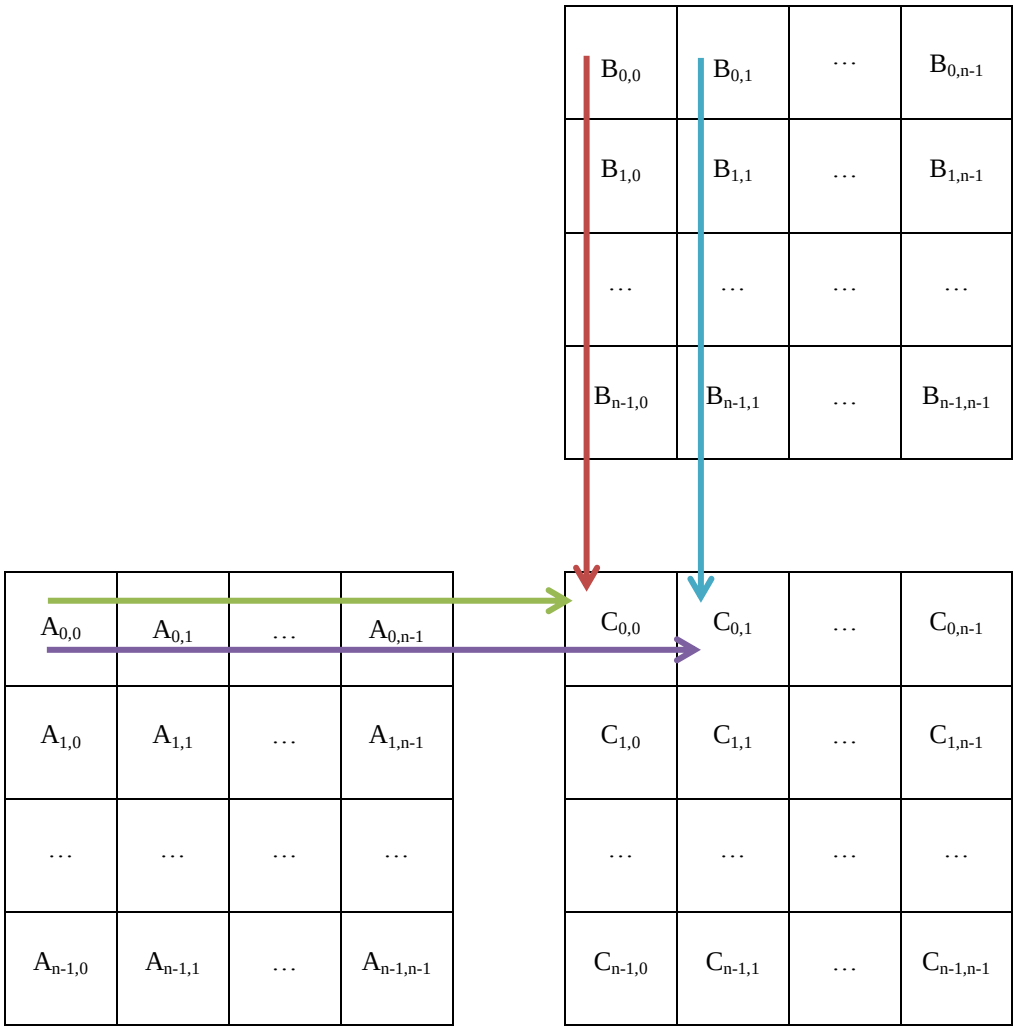
Tablo 3.2. Paralel toplama hesaplama süreleri ve hızlanma faktörleri

Versiyon	Süre(ms)	Adım hızlanma	Kümülatif hızlanma
1	8.054		
2	3.456	2.33x	2.33x
3	1.722	2.01x	4.68x
4	0.965	1.78x	8.34x
5	0.536	1.80x	15.01x
6	0.381	1.41x	21.16x
7	0.268	1.42x	30.04x

Bu üç aşama sonunda, ilk aşamaya göre 4.68 kat hesaplama süresinde hızlanma elde edilmiştir. Toplamda altı geliştirme ile yedi adımdan oluşan bu çalışmada elde edilen hesaplama süreleri ve hızlandırma katsayıları Tablo 3.2'de bulunmaktadır.

3.4.2. GİB üzerinde matris çarpma

İki kare matrisin çarpımı konusu, hesaplamalı bilimlerde GİB kullanımı için çoğunlukla üzerinde durulan uygulamalardan biridir. Kirk ve Hwu [56], 2010 yılında yayınladıkları kitapta GİB üzerinde matris çarpımını ayrıntılarıyla belirtmişlerdir. Bu çalışmaya benzer şekilde aynı boyutlarda iki kare matrisin, çarpılıp üçüncü bir matrise yazdırılması bu tez çalışmasının GİB'de işlenmesi için öncelikle ele alınmıştır. A_{n*n} ve B_{n*n} matrisi gibi iki kare matrise MİB üzerinde rasgele sayılar atanmakta ve GİB'e kopyalanmaktadır. Çarpım işlemi GİB üzerinde gerçekleştirilip C_{n*n} gibi bir matrise çarpım işlendikten sonra C matrisi MİB'ye yazdırılmak üzere kopyalanmaktadır.



Şekil 3.7. Paralel matris çarpımı

GİB üzerinde çarpımı gerçekleştirecek kernel için grid yapısı $(\frac{n}{16}, \frac{n}{16}, 0)$ blok ve her blokta (16,16,0) thread içerecek şekilde oluşturulmuştur. Söz konusu blok ve thread indeksleri kullanılarak tüm satır ve sütun çarpım işlemleri paralel olarak ele alınmaktadır. Şekil 3.7'da olduğu gibi C matrisinin her elemanının hesaplanması eş zamanlı olarak yürütülmektedir. Fakat her satır, sütun çarpımından doğan toplama işlemi sıralı olarak ele alınmaktadır.

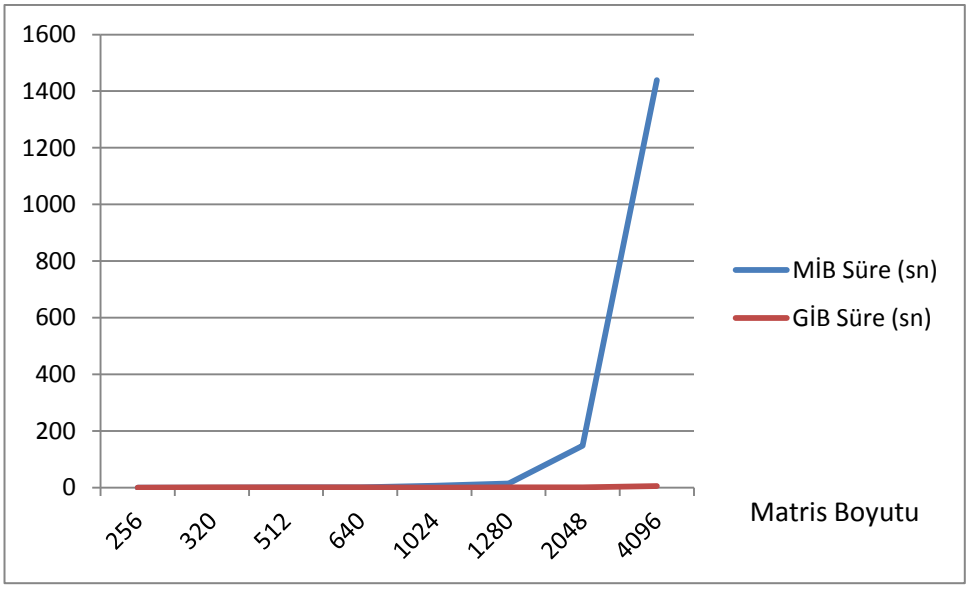
GİB'deki çarpım işlemi sonlandıktan sonra, MİB'de aynı işlem gerçekleştirilerek çarpım süreleri tutulmuştur. Matris çarpma denemelerinden elde edilen sonuçlar Tablo 3.3'te görülmektedir. 320*320 boyutlu matrislerin çarpımına kadar MİB sürelerinin kısmen daha önde olsa da 512*512 boyutlu matrislerin çarpımından başlamak üzere GİB'nin etkinliği göze çarpmaktadır. Öyle ki 4096*4096 boyutlu iki matrisin çarpımından oluşan işlem, MİB'de 1439 saniye, GİB'de ise yalnızca 5.1 saniye sürmektedir. Bu işlemi, GİB, MİB'e göre yaklaşık olarak 280 kat daha hızlı yapabilmektedir.

Tablo 3.3. GİB üzerinde matris çarpma ölçüm süreleri ve hızlanma faktörleri

Kare Matrislerin Boyutu	MİB Süre (sn)	GİB Süre (sn)	Hızlanma
(256*256)*(256*256)	0.066	0.297	-
(320*320)*(320*320)	0.124	0.356	-
(512*512)*(512*512)	0.704	0.327	2.15x
(640*640)*(640*640)	1.4	0.319	4.39x
(1024*1024)*(1024*1024)	6.9	0.305	22.62x
(1280*1280)*(1280*1280)	13.93	0.466	29.89x
(2048*2048)*(2048*2048)	147.75	0.906	163.08x
(4096*4096)*(4096*4096)	1439.3	5.1	282.22x

GİB'de yapılan paralel matris çarpma işlemi 512*512 boyutlu örnekler ile ele alındığında için gird yapısında *x* ve *y* eksenlerini içeren (32,32,0) toplam 1024 blok tasarlanmıştır. Alt bölüm 3.2'de tartışıldığı gibi bir akış işlemcisinde en fazla 8 blok işlenebilmektedir. Öte yandan, her blok için thread organizasyonu (16,16,0) şeklindedir. Buradan her blok için 256 thread tanımlandığı ortaya çıkmaktadır. Bir akış işlemcisinde en fazla 1536 threadin aynı anda işlenebileceği

kısıtı doğrultusunda, bu grid yapısında her akış işlemcisinde 6 blok çalışabilmektedir. 16 akış işlemcisini kullanarak 96 blokta bulunan 24576 thread ile eş zamanlı olarak işlem yapılmaktadır. Toplamda 1024 blok ve 262144 thread olduğunu düşünerek işlemin on bir parçada yapıldığını söyleyebiliriz.



Şekil 3.8.GİB üzerinde matris çarpma süreleri

Şekil 3.8'de farklı boyutlardaki kare matrislerin MİB ve GİB'deki çarpım sürelerinin karşılaştırması yer almaktadır. Matris boyutundaki artış ile birlikte MİB'deki çarpım süresinin hızla arttığı gözlenirken, GİB'de bu artış çok düşük seviyededir.

3.5. Metasezgisel Algoritmalar ile GİB üzerinde Optimizasyon

Bir önceki bölümde GİB'nin potansiyel gücünü gösteren bazı örnekler incelenmişti. Bu potansiyelden faydalanmanın, kombinatorik optimizasyon problemleri için de mümkün olduğu literatürde yapılan sınırlı sayıdaki çalışmadan anlaşılmaktadır. Özellikle GA ve türevleri, karınca kolonileri optimizasyonu gibi popülasyon temelli metasezgisel algoritmalar, yapıları gereği paralelleştirmeye ve dolayısıyla GİB üzerinde çalışmaya uygundur.

Tsutsui ve Fujimoto [57] 2009 yılındaki çalışmalarında KAP için GİB üzerinde paralel bir GA sunmuşlardır. Bilgisayarın ana belleğinde oluşturdukları

bireylerin tümünü cihaz belleğine kopyaladıktan sonra, paylaşılan belleğin alabileceği kadar bireyi paylaşılan hafızaya taşımaktadırlar. Paylaşılan belleğe kullanarak bireylere GA operatörlerini uygulayıp cihaz belleğine geri göndermektedirler. Bu işlem tüm bireyler bitene kadar sürmektedir. Bundan sonra ise, tüm bireyler cihaz belleğinden ana belleğe kopyalanmaktadır. Burada bireylerin parça parça cihaz belleğinden paylaşılan belleğe taşınmasının sebebi bu belleğin erişim hızından faydalanmaktır. Çalışmada alınan ölçümlerde GİB üzerinde geliştirilen algoritmanın MİB'e göre 3 ile 12 kat arası daha hızlı sonuçlar elde edilmiştir. Aynı yazarlar 2011 yılında yaptıkları çalışmada [58], KAP'ın çözümü için karınca kolonileri optimizasyonu ve 2-opt yerel arama algoritmasının bira araya getirilmesinden oluşan bir yöntem sunmuşlardır. Çalışmada 2-opt yerel arama algoritmasının, yöntemin bir tekrarında geçen sürenin büyük bir kısmını aldığı vurgulanmış ve CUDA ile bu durumun iyileştirilebileceği belirtilmiştir. Karıncaların başlangıç çözümlerinin oluşturulması, feromonların güncellenmesi ve 2-opt yerel arama algoritmasını GİB üzerinde gerçekleştirmişlerdir. Elde edilen sonuçlara göre, GİB üzerindeki çözüm süreleri MİB'e göre ortalama 24.6 kat daha hızlı olduğu vurgulanmıştır.

Bir diğer çalışmada ise Melab ve ark. [59], KAP için geliştirilen bir lokal arama algoritmasının GİB üzerinde uygulamasını gerçekleştirmişlerdir. Bilinen en iyi çözümlere %13 ile %27'lik uzaklıklarla alınan ölçümlerde 15.1 kata kadar hızlanma elde etmişlerdir. Luong ve ark. [60], evrimsel algoritmaların GİB'de paralelleştirilmesi üzerine yaptıkları çalışmada literatürde yer almıştır. KAP'ın ele alındığı bu çalışma da 14.6 kata kadar GİB'de hızlanma elde edilmiştir. Fakat, MİB için çözüm süreleri günler, GİB için ise saatler şeklindedir.

Czapinski [61], literatüre son eklenen çalışmalardan olan makalesinde CUDA ortamında KAP için çoklu başlangıçlı yasaklı arama algoritması geliştirmiştir. Algoritmanın CUDA uyarlaması üzerine önemli bilgilere değinilmiş ve GİB üzerinde kodlama yaparken hafıza yönetiminin ne kadar önemli bir nokta olduğu vurgulanmıştır. Yasaklı aramada en çok zaman alan işlemin komşu çözümleri kontrol etmek olduğu belirtilmiştir. Komşu çözümlerin GİB'de her thread için ayrı ayrı ele alınmıştır. Komşu çözümlere bakılırken, amaç fonksiyonunu yeniden hesaplamak yerine, sadece değişen noktaları ele almanın

işlem süresinde anlamlı kısalmalar yaratacağına değinilmiştir. Akış ve uzaklık matrisini paylaşılan belleğin alan kısıtı nedeniyle cihaz belleğinde tutulmuştur. Çalışmada, KAP için geliştirilen literatür problemlerinden bazıları ele alınmış ve %1'lik çözüm uzaklıklarıyla, 70 kata kadar daha kısa sürede sonuçlar elde edilmiştir.

KAP literatürü dışında, diğer kombinatorik optimizasyon problemlerinin çözümü içinde GİB üzerinde metasezgisel algoritmaların ele alındığı çalışmalar bulunmaktadır. Cecilia ve ark. [62] gezgin satıcı problemi için karınca kolonileri optimizasyonu geliştirmiş ve 20 kata kadar daha hızlı çözümler elde etmişlerdir. Çalışmada karıncaların bazıları kraliçe bazıları ise işçi olarak belirlenmiştir. İşçi karıncalar, kraliçe karıncaya sonraki şehrin seçiminde yardımcı olmaktadır. Ayrıca paralelleştirmeye uygun yeni bir rulet tekerleği seçiş yöntemi önermişlerdir. Grid yapısında her blok bir kraliçe karıncaya, her thread ise bir şehre veya işçi karıncaya karşılık gelmektedir. Çalışmanın bir başka yönü ise GİB üzerinde feromon güncellemesinde kullanılan operatörün atomik işlemlerden oluşmasıdır. Aynı problem için yapılan bir başka çalışmada, Delevacq ve ark. [63] karınca kolonileri optimizasyonu ile 23.6'ya kadar çıkan hızlanma elde etmişlerdir. Karıncaları GİB üzerinde eş zamanlı olarak ele almalarının yanı sıra 3-opt yerel arama algoritmasının paralelleştirilmesi üzerinde durmuşlardır. Shulz [64], araç rotalama problemi için yerel arama algoritmalarının GİB uygulamaları üzerine bir çalışma sunmuştur. Hızlanma faktörlerine ait bilgi verilmeyen çalışmada, özellikle GİB üzerinde işlem yaparken darboğazlardan birisi olan hafıza işlemlerinin azaltılabileceği üzerinde durulmuştur. Aynı problem üzerine çalışan Groer ve ark. [65] yerel arama tekniklerini GİB üzerinde uygulamışlardır. Yöntemlerinin diğer kombinatorik optimizasyon problemlerine de uyarlanabileceğini vurgulayan yazarlar hızlanma faktörleriyle ilgili detayları aktarmamışlardır.

Bir diğer makalede, Huang ve ark. [66] GA ile bulanık çizelgeleme problemleri üzerine çalışmışlardır. GA'da kromozomlar oluşturulurken rassal anahtar yöntemi kullanılmıştır. Bu yöntem ile oluşturulan kromozomlar ile GİB üzerinde etkili çalışılabileceği vurgulanmıştır. Bir başka çalışma, özel bir çizelgeleme problemi türü olan n işin m makineye çizelgelenmesi problemi

üzerine yapılmıştır. Czapinski ve Barnes [5], çalışmalarında problemin çözümü için GİB üzerinde çalışan bir yasaklı arama algoritması sunmuşlardır. Özellikle amaç fonksiyonu hesaplamalarının GİB'de paralel olarak yapılması ve komşu seçiminin eş zamanlı aranmasının algoritmanın etkinliğini artırdığı belirtilmiştir. Çalışmada, GİB üzerinde çalıştırılan programda MİB'e göre 89 kata kadar hızlanma elde edilmiştir.

Tablo 3.4. GİB'de metasezgiseller ile kombinatorik optimizasyon problemleri

Yazarlar	Problem	Yöntem	Hızlanma	Yonga Kümesi
Tsutsui ve Fujimoto [57]	KAP	GA	3x-12x arasında	GTX285
Tsutsui ve Fujimoto [58]	KAP	Karınca Kolonileri + Yerel Arama	Ortalama 24x	GTX480
Melab ve ark. [59]	KAP	Evrimsel Algoritma + Yerel Arama	1x-13x arasında	GTX480
Luong ve ark. [60]	KAP	Evrimsel Algoritma	5x-12x arasında	GTX280
Czapinski [61]	KAP	Çoklu Başlangıçlı Yasaklı Arama	70x'e kadar	GTX480
Cecilia ve ark. [62]	Gezgin Satıcı	Karınca Kolonosi	20x'e kadar	Tesla C2050
Delevacq ve ve ark. [63]	Gezgin Satıcı	Karınca Kolonosi	23x'e kadar	Tesla C2050
Shulz [64]	Araç Rotalama	Yerel Arama	-	GTX480
Huang ve ark. [66]	Çizelgeleme	GA	Ortalama 19x	GTX285
Czapinski ve Barnes [5]	Çizelgeleme	Yasaklı Arama	89x'e kadar	Tesla C1060
Groer ve ark. [65]	Araç Rotalama	Yerel Arama	-	-

Tablo 3.4. GİB'de metasezgiseller ile kombinatorik optimizasyon problemleriTablo 3.4'te GİB üzerinde kombinatorik optimizasyon problemlerinin metasezgisel yöntemler ile çözümü üzerine yapılan söz konusu çalışmalardan bilgiler bulunmaktadır. Kullanılan teknolojinin bu alanda kullanılmasının oldukça yeni olması sebebiyle literatürde sınırlı çalışma ile karşılaşılmıştır. Tabloda sırasıyla bu bölümde sözü geçen çalışmaların yazarları, çalışılan problem türü,

kullanılan metasezgisel yöntem veya yöntemler, elde edilen hızlanma faktörleri ve kullanılan GİB'nin yonga kümesi bulunmaktadır. Yonga kümesi sütununda bulunan değerler nVIDIA'e ait ekran kartları üzerinde bulunan yonga kümelerinin isimleridir. Bunlardan GTX480 ve Tesla C2050 Fermi mimarisinde diğerleri ise GT200 mimarisinde geliştirilmiş yonga kümeleridir.

4. KAP İÇİN BİR EVRİMSEL ALGORİTMA

Tez Çalışmasında, melez bir evrimsel yaklaşım önerilmiştir. GA operatörlerinin yanı sıra, farklı bir mutasyon operatörü ve 2-opt yerel arama algoritmasından esinlenilerek oluşturulan yeni bir operatör kullanılmıştır. Bu şekilde geliştirilen sıralı algoritma, paralel hesaplamaya uygun hale getirilmiş ve GİB üzerinde CUDA ile programlaması yapılmıştır. İzleyen alt bölümde, evrimsel algoritmalar ile genel bilgiler verildikten sonra önerilen algoritmada kullanılan operatörler tanıtılmış, sonrasında sıralı ve paralel algoritmadan bahsedilmiştir.

Evrimsel algoritmalar

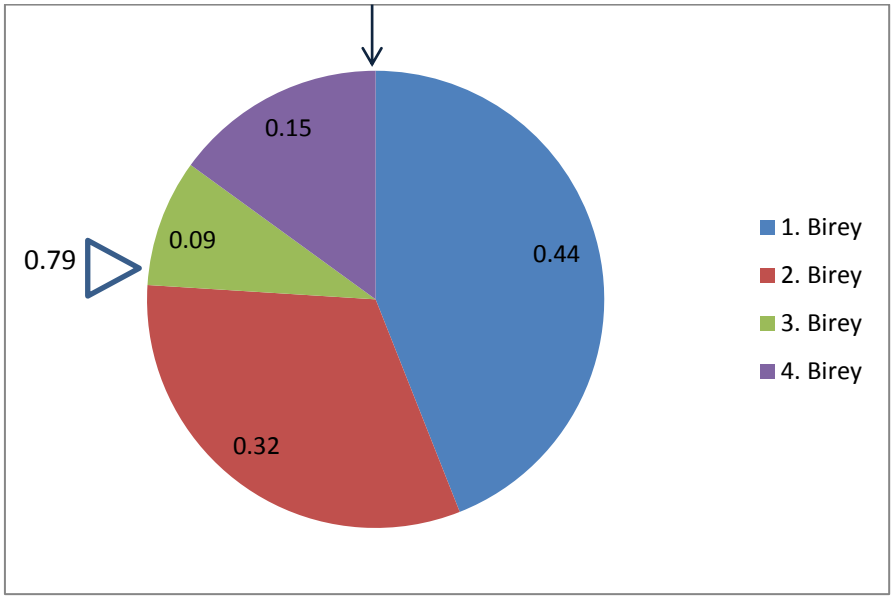
Evrimsel algoritmalar, nesilden nesle iyi özelliklerin taşınarak, kaliteli bilginin korunduğu ve her tekrarda çözüme yakınsamanın olduğu yöntemlerdir. Evrimsel bir algoritma olan GA'nın Holland [40] tarafından ortaya atılmasından sonra operatörler üzerinde çok farklı seçenekler ortaya çıkmıştır.

GA tek noktadan arama yapan algoritmaların aksine popülasyondaki birey sayısı kadar çözüm üzerinde çalışır. Yaygın kullanımda popülasyonda bulunan ve her biri KAP'ın bir çözümüne karşı gelen bireyler için permütasyon gösterim şekli kullanılır. Permütasyon dışında bireylerin 0-1 değerleri ile çözümün gösterildiği çalışmalar da mevcuttur. GA'da her bir bireye kromozom ismi verilmektedir. Kromozom yapısında bilgi taşıyan her bir bileşense gen olarak adlandırılmaktadır.

GA bir optimizasyon problemi için ele alındığında ilk adım kromozomların bir araya geldiği başlangıç popülasyonunu oluşturmaktır. Başlangıç popülasyonu tamamen rasgele oluşturulabileceği gibi, probleme özgü her hangi bir sezgisel yöntem ile de oluşturulabilmektedir. Başlangıç popülasyonunun oluşturulmasından sonra her bir kromozomun uygunluk değeri hesaplanır. Kromozomların uygunluk değerleri, sonraki nesle aktarılmada kullanılmaktadır.

Algoritmanın sonraki adımı *yeniden üretim* veya *seçim* olarak bilinen operatördür. Seçim operatörü, temel olarak uyum gücü yüksek kromozomlardan, sonraki nesli oluşturacak bireylerin belirlenmesidir. Rulet tekeri ve turnuva

seçimi, en çok bilinen seçim yöntemleridir. Rulet tekeri yönteminde tüm bireylerin uygunluk değerleri ve bunların toplamı tutulur. Sonrasında, tüm bireylerin uygunluk değeri, toplama bölünerek her birey için [0,1] aralığında seçilme olasılıkları elde edilir. Bu seçilme olasılıkları kümülatif olarak tutulur. [0,1] aralığında rasgele bir sayı belirlenir ve o sayının denk geldiği kümülatif aralıkta bulunan birey sonraki nesli oluşturmak üzere seçilir. Şekil 4.1'deki grafikte örnek olarak dört bireyin uygunluk değerlerinden elde edilen olasılıklar bulunmaktadır. Ok ile gösterilen yerden itibaren ruletin saat yönünde çalıştığı varsayılsın ve seçilen rasgele değer 0.79 olsun. Bu sayı, olasılıklar kümülatif olarak toplandığında, 3 numaralı bireye düşen aralıkta yani 0.77 ile 0.85 arasında bulunmaktadır. Bu durumda, rulet tekerinden 3 numaralı bireyin seçildiği anlaşılmaktadır. Seçim operatörleri, popülasyon içi çeşitliliğin korunması açısından büyük önem taşımaktadır. Çeşitliliğin sağlanamaması durumunda, erken yakınsama problemiyle karşılaşılması kaçınılmazdır. Rulet tekeri yöntemi, güçlü bireylerin fazla sayıda seçilmesine sebep olduğundan, popülasyon içi çeşitliliğin azalmasına sebep olmaktadır.



Şekil 4.1. Rulet tekeri seçim yöntemi

Turnuva seçim yönteminde ise, iki ya da daha çok birey uygunluk değerlerine göre karşılaştırılır. Daha nitelikli bilgiye sahip, uyum gücü daha iyi

olan birey turnuvayı kazanmakta ve sonraki nesli oluşturmak üzere seçilmektedir. Turnuvaya katılıp kazanan bireylerin tekrardan turnuvaya katılması engellenebilir ya da katılmasına izin verilebilir. Ayrıca, turnuva yapılırken, uyum gücü daha düşük olan bireye de belirli seçilme olasılıkları belirlenebilir.

Seçim operatörünün belirlediği kromozomlar yeni nesli oluşturmak için hazırdır. Seçilen bireyleri bir araya getirerek, kromozomlardaki iyi bilgilerin yeni nesle taşınmasını sağlayan operatör, *çaprazlama* operatörü olarak bilinmektedir. Bu operatörde öncelikle, bireylerin çaprazlama işlemi görme olasılığı ve çaprazlama türü belirlenmektedir. Çaprazlama olasılığı, seçilen bireylerden ne kadarının çaprazlanarak yeni nesli oluşturacağını ifade etmektedir. En basit şekliyle, seçilen iki birey anne ve bana olarak belirlenmektedir. Yeni nesli oluşturacak çocuklar, anne ve babadan çaprazlama yöntemine göre genlerin alınmasıyla meydana gelmektedir. Tek nokta çaprazlama, iki nokta çaprazlama ve pozisyona dayalı çaprazlama bilinen çaprazlama türlerinden bazılarıdır. Bunlardan pozisyona dayalı çaprazlama önerilen evrimsel algoritmanın detayları açıklanırken anlatılmaktadır.

Şekil 4.2'de tek nokta çaprazlamaya yönelik 12 boyutlu KAP için hazırlanmış permütasyon tabanlı kromozomlardan oluşan bir örnek bulunmaktadır. Bu çaprazlama türünde her hangi bir nokta rasgele olarak seçilir. Seçilen çaprazlama noktasına kadar olan genler Birey 1'den Yeni Birey 1'e, Birey 2'den ise Yeni Birey 2'ye taşınmaktadır. Kalan boş genler ise Yeni Birey 1 için Birey 2'den, Yeni Birey 2 için ise Birey 1'den sırasıyla tamamlanmaktadır.

							↓					
Birey 1	2	8	12	1	3	5	6	11	9	4	7	10
Birey 2	4	9	5	7	10	1	3	2	6	8	11	12

Yeni Birey 1	**2**	**8**	**12**	**1**	**3**	**5**	**6**	4	9	7	10	11
Yeni Birey 2	**4**	**9**	**5**	**7**	**10**	**1**	**3**	2	8	12	6	11

Şekil 4.2. Tek nokta çaprazlama

Şekil 4.3'te ise iki nokta çaprazlamanın gösterildiği bir örnek bulunmaktadır. Tek nokta çaprazlamadan farklı olarak, bu teknikte iki çaprazlama

noktası belirlenmektedir. Belirlenen iki çaprazlama noktası arasında kalan genler Birey 1'den Yeni Birey 1'e, Birey 2'den ise Yeni Birey 2'ye aktarılmaktadır. Kalan boş genler ise benzer şekilde Yeni Birey 1 için Birey 2'den, Yeni Birey 2 için ise Birey 1'den sırasıyla tamamlanmaktadır.

			↓						↓			
Birey 1	2	8	12	1	3	5	6	11	9	4	7	10
Birey 2	4	9	5	7	10	1	3	2	6	8	11	12

Yeni Birey 1	4	7	10	**1**	**3**	**5**	**6**	**11**	**9**	2	8	12
Yeni Birey 2	8	12	5	**7**	**10**	**1**	**3**	**2**	**6**	11	9	4

Şekil 4.3. İki nokta çaprazlama

GA'da çaprazlama işlemi sonlandıktan sonra yeni nesil oluşmaktadır. Yeni nesil için sıradaki operatör *mutasyon* operatörüdür. Bu operatörde, her bir yeni bireyin genlerinde küçük olasılıklarla değişiklikler yapılır. Bir bireyin mutasyona gireceği belirlendikten sonra iki gen rasgele olarak seçilir ve değiştirilir. Bunun dışında ikiden fazla değişimin göz önüne alındığı mutasyon operatörleri de bulunmaktadır.

Evrimsel algoritmalar popülasyon temelli oldukları için keşfetme yetenekleri oldukça fazladır. Farklı bireyle çözüm uzayında iyi noktaları yakalayabilmektedir. Ancak keşfedilen noktalarda, derinlemesine arama özelliği kısıtlıdır. Bu yüzden GA genellikle melezleştirilerek yerel arama algoritmalarıyla birlikte kullanılır. Böylelikle algoritmaya keşfetme özelliğinin yanı sıra derinlemesine arama özelliği kazandırılmaktadır.

4.1. Önerilen Evrimsel Algoritmanın Operatörleri

Popülasyon tabanlı evrimsel bir metasezgisel algoritma olan GA'nın KAP'ın çözümü için literatürde sıklıkla adının geçmekte olduğundan önceki bölümlerde bahsedilmişti. Bununla birlikte, GA birçok çalışmada yerel arama algoritmalarıyla birlikte kullanılmaktadır. Yapılan çalışmaları birbirinden farklı

kılan unsur ise, GA'nın seçim, çaprazlama ve mutasyon gibi operatörlerinde çeşitli tekniklerin kullanılmasıdır. Çalışmada önerilen sıralı algoritmada 1000 bireyden oluşan başlangıç popülasyonu tamamen rasgele olarak oluşturulmuştur. Bireylerin ilk kez amaç fonksiyonlarının değerlendirilmesinden sonra, sırasıyla, seçim, çaprazlama, mutasyon, yeni arama ve elitizm operatörleri yer almaktadır. Yapılan tekrarlarla tıpkı evrim teorisindeki gibi, yeni nesillerin daha da iyileşmesi amaçlanmıştır. Burada yeni nesillerin iyileşmesi olarak kastedilen problemin çözümüne yakınsamaktır.

4.1.1. Seçim operatörü

Önerilen yaklaşımda kullanılan seçim operatörü, temel olarak bir turnuva seçim yöntemidir. Çaprazlama operatörüne göndermek üzere, yeni nesli oluşturacak bireyler bu adımda seçilmektedir. Popülasyondan rasgele seçilen, herhangi iki birey amaç fonksiyonu değerleri birbirleriyle karşılaştırılmaktadır. Amaç fonksiyonu değeri daha düşük olan birey 0.85 olasılıkla yeni nesli oluşturmak üzere çaprazlama operatörüne gönderilmektedir. Öte yandan, amaç fonksiyonu değeri daha kötü olan birey de 0.15 olasılıkla turnuvayı kazanabilmekte ve yeni nesli oluşturmak için kullanılabilmektedir. Burada söz konusu olasılıklara deneysel çalışmalar sonucu ulaşılmış olup, bu tür bir uygulamaya popülasyon içi çeşitliliği artırmak ve erken yakınsama probleminden kurtulmak üzere gidilmiştir.

4.1.2. Çaprazlama operatörü

Çaprazlama operatörü, seçim operatöründen gelen bireylerin yeni nesli oluşturduğu adımdır. Bu bireylerin %80'i çaprazlanmakta, kalanlar ise seçim operatöründen geldikleri gibi yeni nesle aynen aktarılmaktadır. Bu adımda, tek nokta, iki nokta ve pozisyona dayalı çaprazlama türleri karşılaştırılmış ve pozisyona dayalı çaprazlamanın daha iyi sonuçlar vermesi sebebiyle kullanılmasına karar verilmiştir. Popülasyondaki sırası art arda olan iki birey ele alınmaktadır. Devamında bir bireydeki gen sayısından daha küçük bir tam k sayısı

rassal olarak seçilmektedir. Bu *k* sayısı, çaprazlanacak ardışık bireylerden kaç genin yeni nesle aktarılacağını ifade etmektedir. Çaprazlama sonrası iki birey elde edilmektedir. İlk birey elde edilirken, *k* kadar gen, çaprazlanacak ardışık bireylerden ilkinden, yerleri rasgele olacak şekilde seçilmekte, kalan genler ise diğer bireyden tamamlanmaktadır. İkinci birey elde edilirken ise, çaprazlanacak ardışık bireylerden ikincisinden *k* kadar gen, yerleri ilk oluşturulan yeni bireydeki gibi olacak şekilde elde edilmekte, eksik genler diğer bireyden tamamlanmaktadır. Örneğin, 12 boyutlu bir KAP için iki birey çaprazlanacak olsun ve *k* sayısı 7 olsun. Ayrıca bireylerden alınacak rasgele 7 genin yerleri oklardaki gibi belirtilsin. Bu koşullarda oluşacak yeni bireyler Şekil 4.4'te görülmektedir.

	↓	↓			↓		↓	↓	↓		↓	
Birey 1	2	8	12	1	3	5	6	11	9	4	7	10
Birey 2	4	9	5	7	10	1	3	2	6	8	11	12

Yeni Birey 1	**2**	**8**	4	5	**3**	10	**6**	**11**	**9**	1	**7**	12
Yeni Birey 2	**4**	**9**	8	12	**10**	1	**3**	**2**	**6**	5	**11**	7

Şekil 4.4. Pozisyona dayalı çaprazlama

4.1.3. Mutasyon operatörü

Çaprazlama operatöründen sonra, bireylerin tümü mutasyon operatörüne gelmektedir. Burada, öncelikle bireyden iki gen rasgele seçilmektedir ve bu iki genin yerlerinin değişikliğinde ortaya çıkan amaç fonksiyonu değeri ile değişiklikten önceki amaç fonksiyonu arasındaki fark negatif ise değişiklik kabul edilmektedir. Eğer söz konusu fark pozitif ise, bir başka deyişle, değişiklik daha kötü bir çözüm veriyorsa, 0.1'lik bir olasılıkla kabul edilmekte; 0.9 olasılıkla değişiklik reddedilmektedir. Popülasyon içi çeşitliliğin, iyileşerek artmasını ve yerel minimum noktalardan kaçmayı hedefleyen bu strateji iki kez üst üste tekrar etmektedir.

4.1.4. Yeni arama operatörü

Yeni arama operatörü, temel olarak bir bireydeki ikili değişikliklere bakarak, o bireyin iyileştirilmesi için kullanılmıştır. Bu operatörde, 2-opt yerel arama algoritması benzeri bir sistem oluşturulmuştur. Boyutu *n* olan bir simetrik KAP 'ta $\frac{n(n-1)}{2}$ adet alternatif değişiklik her birey için ele alınmaktadır. Kontrol edilen her hangi bir değişiklik, bireyin amaç fonksiyonu değerini azaltması durumunda kabul edilmekte, aksi takdirde değişiklik yapılmamaktadır. Değişiklik yapılsın ya da yapılmasın algoritma aramaya aynı sıradan devam etmektedir. Operatörde ikili değişikliklerin olumlu olup olmadığı araştırılırken amaç fonksiyonun yeniden hesaplanmasına gerek duyulmamaktadır. Bunun yerine sadece, değişecek genlerden ortaya çıkan durum ele alınmaktadır. Permütasyon tabanlı simetrik bir KAP çözümünde değiştirilmek üzere önerilen tesislerin permütasyondaki sırası *a* ve *b* olsun. Ayrıca *π(a) a* sırasındaki ve *π(b) b* sırasındaki tesisi belirtsin. Bu durumda delta fonksiyonu aşağıdaki gibidir. Delta fonksiyonunda elde edilen değerin negatif olması *a* ve *b* sırasındaki tesislerin değişimin çözümü iyileştireceğini gösterir.

$$\delta = \sum_{t=1}^{n} \sum_{a \neq t} \sum_{b \neq t} (d_{at} - d_{bt})(f_{\pi(b)\pi(t)} - f_{\pi(a)\pi(t)})$$

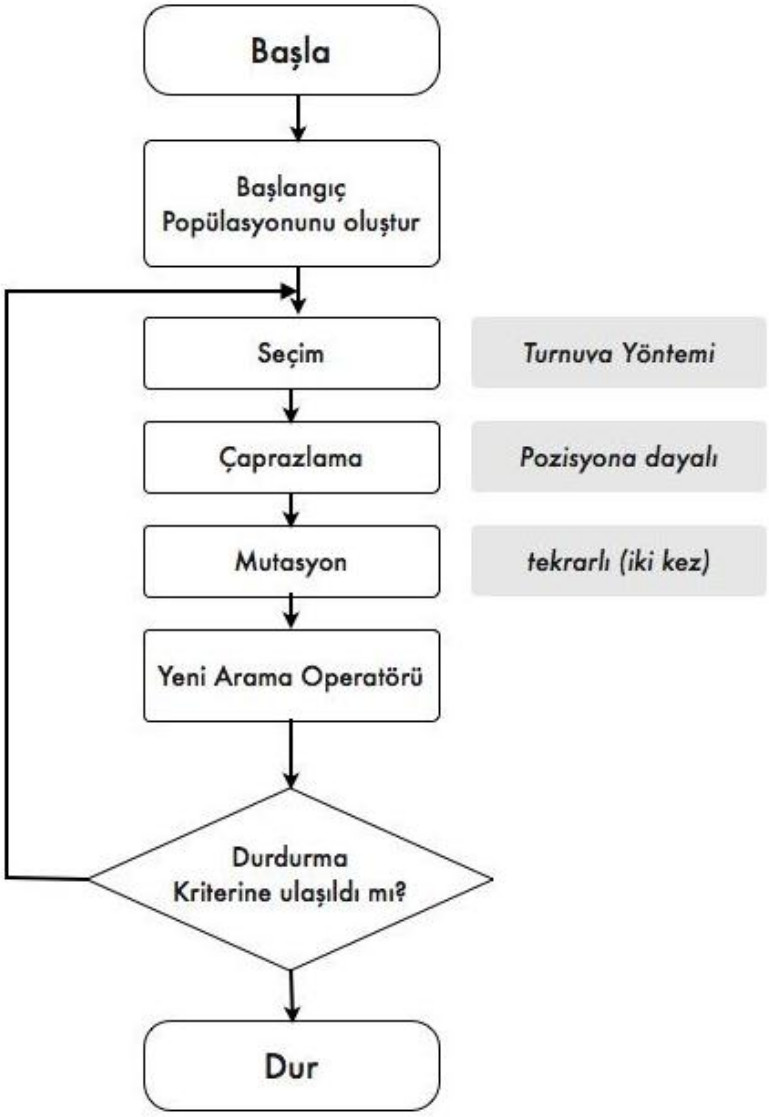
Simetrik problemler için gerçek ikili değişim farkı δ değerinin iki katı olur.

4.1.5. Elitizm operatörü

Elitizm operatörü algoritmanın her tekrarının son adımı olarak bulunmaktadır. Burada, popülasyon içerisindeki en iyi bireyin saklanması sağlanmaktadır. Böylelikle, bütünsel en iyi çözüme yakınsama amacıyla, en iyi birey sürekli popülasyonda tutulmaktadır.

4.2. Önerilen Sıralı Yaklaşımın Akış Diyagramı

Önerilen sıralı yaklaşım önceki bölümde anlatılan operatörlerden oluşmaktadır. Sıralı algoritmanın akış diyagramı Şekil 4.5'teki gibidir.



Şekil 4.5. Sıralı algoritmanın akış diyagramı

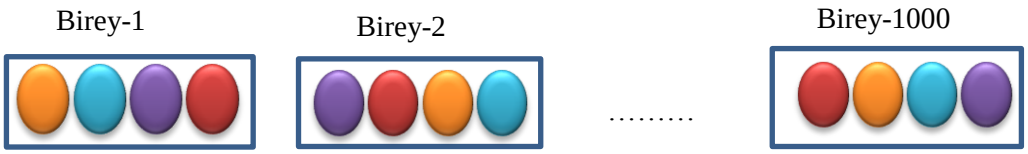
Önceden de belirtildiği gibi seçim operatöründe özelleştirilmiş bir turnuva seçim yöntemi, çaprazlama operatöründe pozisyona dayalı çaprazlama yöntemi kullanılmıştır. Mutasyon operatöründe ise iki kez üst üste tekrar eden çözümü iyileştirmeyi amaçlayan fakat kötü çözümlerinde kabul edildiği bir yöntem kullanılmıştır. Yeni arama operatörü ise tüm ikili değişikliklere bakılarak sürekli çözümler iyileştirilmek üzere ele alınmıştır.

4.3. Önerilen Yaklaşımın Paralelleştirilmesi

KAP literatüründe bulunan çalışmalarda genellikle MİB üzerinde etkin fakat çözüm süresinin dikkate alınmadığı çalışmalar mevcuttur. GİB üzerinde yapılan çalışmalarda ise çözüm kalitesinden çok, belirli çözüme uzaklıklar ile yalnızca süre karşılaştırmaları yapılmaktadır. Bu çalışmanın özgün yanı, literatüre KAP çözümü için sıralı ve paralel olmak üzere birer algoritma sunulmasının yanı sıra, kalitesi yüksek etkin çözümler ile birlikte GİB'yi kullanarak çözüme makul sürelerde ulaşılmasıdır.

Önerilen sıralı yaklaşımın bazı operatörleri eş zamanlı olarak işlenmeye uygun yapıdadır. Bu çalışmada, tüm bireylerinin amaç fonksiyonunun değerlendirilmesi ve yerel arama operatörüne paralel bir yapı düzenlenmiştir. Söz konusu iki kısım, algoritma çalışırken her bir tekrarda en çok zaman alan kod parçalarıdır. Bu kısımların, paralel yapıya dönüştürüldükten sonra, GİB üzerinde kodlanması ve böylelikle eşzamanlı olarak işlenmesi ile çözüm sürelerinde kayda değer azalma beklentisi, bu çalışmanın temel amaçlarından biridir.

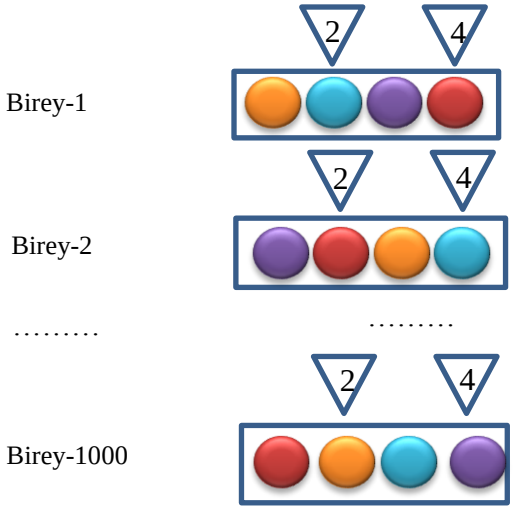
Paralel yapı kazandırılan iki operatörden biri amaç fonksiyonlarının değerlendirilmesidir. Her tekrarda, tüm bireylerin amaç fonksiyonları Şekil 4.6'daki gibi eş zamanlı olarak hesaplanmaktadır. Bir sonraki alt bölümde bu operatörde yapılan eş zamanlı işlemler açıklanmıştır.



Şekil 4.6. Eş zamanlı amaç fonksiyonu hesaplama

Benzer şekilde yerel arama operatörü de GİB üzerinde eş zamanlı işlenmek üzere ele alınmaktadır. Örneğin, Şekil 4.7'deki gibi tüm bireylerde ikinci ve dördüncü genlerin yer değiştirmesi ele alınsın. Söz konusu değişikliğin kontrolü popülasyonda bulunan 1000 birey için eş zamanlı olarak GİB üzerinde yürütülmektedir. Tüm bireyler için aynı değişiklik kontrol edilirken, her bireye ait bir delta fonksiyonu hesaplanmaktadır. Her birey için eğer delta fonksiyonun

değeri negatif ise ele alınan sıralardaki genler yer değiştirmektedir. Aksi halde değişiklik yapılmamaktadır.

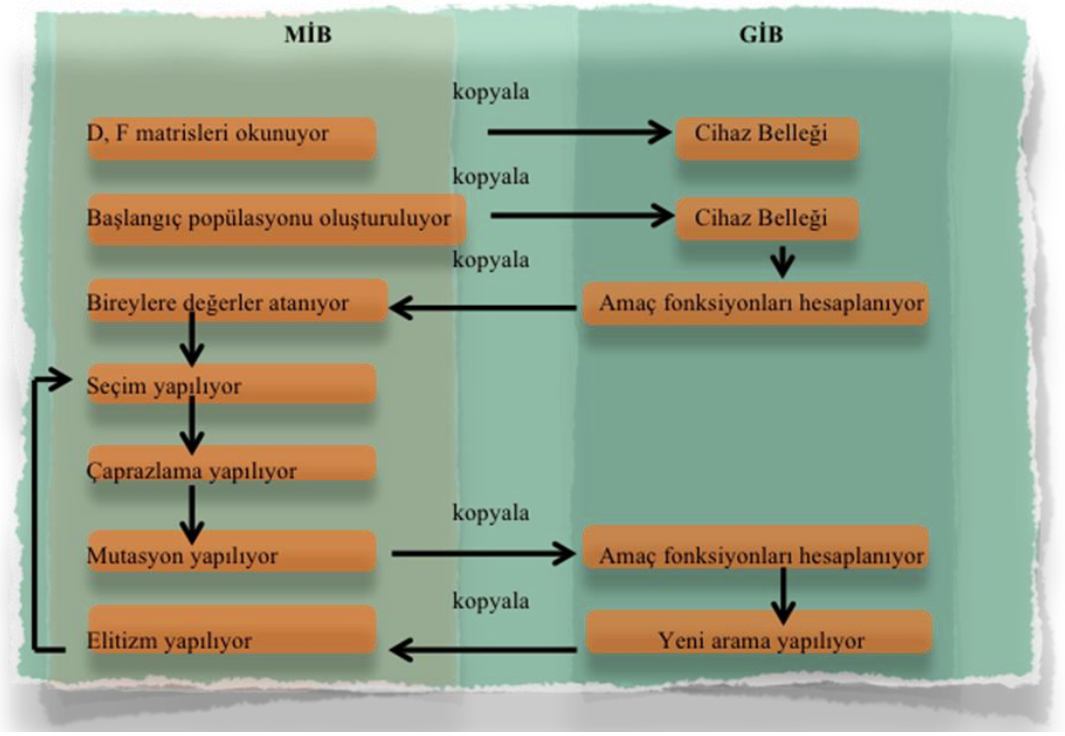


Şekil 4.7. Eş zamanlı arama

GİB'de kodlanmak üzere hazırlanan algoritmanın bu iki kısmı için farklı kerneller oluşturulmuştur. Bir sonraki alt bölümde bu kernellerin yapısı ve programın GİB üzerinde işleyişi açıklanmıştır.

4.4. Paralel algoritmanın GİB'de kodlanması

Geliştirilen paralel yaklaşımın bir kısmı GİB üzerinde çalışırken, bir kısmı ise MİB üzerinde çalışmaktadır. Probleme ait girdi verileri olan uzaklık D ve akış F matrisleri okunduktan sonra cihaz belleğine kopyalanmaktadır. Sonrasında MİB tarafında başlangıç popülasyonu oluşturulmaktadır. Başlangıç popülasyonu cihaz belleğine kopyalanmakta ve burada paralel olarak her bireyin amaç fonksiyonu değeri hesaplanmaktadır. Devamında bireylerin amaç fonksiyonları MİB'nin kullanımındaki ana belleğe aktarılmakta ve burada sırasıyla seçim, çaprazlama ve mutasyon işlemleri yapılmaktadır. Mutasyon işleminden sonra tüm bireyler GİB'de yerel arama operatöründen geçmek üzere cihaz belleğine taşınmaktadır. Burada eş zamanlı yerel arama işlemi gerçekleştirilmekte ve sonrasında bireyler tekrardan ana belleğe kopyalanmaktadır. Bu adımdan sonra MİB'de elitizm işlemi gerçekleştirilme ve döngü seçim işlemiyle



Şekil 4.8. Paralel algoritmanın akış diyagramı

Boyutu n olan simetrik problemlerin amaç fonksiyonu eş zamanlı hesaplanırken $\frac{n(n-1)}{2}$ adet çarpım işlemi yapılmaktadır. Bu yüzden GİB'de tüm bireylerin çarpım işlemlerinin yer aldığı $\frac{n(n-1)}{2}$ boyutunda A vektörü bulunmaktadır. Söz konusu her çarpım A vektöründe bir elemana karşılık gelmektedir. A vektöründeki her $\frac{n(n-1)}{2}$ elemanın toplamının iki katı bir bireyinin amaç fonksiyonu değerini vermektedir.

GİB üzerinde amaç fonksiyonu ve yeni arama operatörünün nasıl çalıştığını daha iyi açıklamak üzere izleyen örnek oluşturulmuştur.

Açıklayıcı Örnek:

Dört tesisin dört aday yere atanacağı simetrik bir problemi ele alalım. Bu problemde T_1, T_2, T_3, T_4 tesisleri arası akış ve Y_1, Y_2, Y_3, Y_4 yerleri arası uzaklık matrisleri F ve D Şekil 4.9'da verilmiştir.

F	T_1	T_2	T_3	T_4
T_1		10	20	30
T_2	10		40	50
T_3	20	40		60
T_4	30	50	60	

D	Y1	Y2	Y3	Y4
Y1		3	5	4
Y2	3		1	2
Y3	5	1		4
Y4	4	2	4	

Şekil 4.9. Akış ve uzaklık matrisleri

Problemin çözümü için tasarlanan evrimsel algoritmada popülasyon büyüklüğünün 3 olduğunu ve popülasyondaki bireylere karşı gelen permütasyonların Π_1, Π_2, Π_3, sırasıyla 1-2-3-4, 4-1-3-2, 1-3-4-2 şeklinde elde edildiğini kabul edelim. Her bireydeki gen sırası yerleri, her genin özelliği de tesisleri temsil etmektedir. Bu durumda üç numaralı bireyde tesis-yer atamaları T1→ Y1, T3→Y2, T4→Y3 ve T2→Y4 şeklinde gerçekleşmiştir. Bu verilere göre GİB üzerinde amaç fonksiyonu hesaplanırken oluşturulan *A* vektörü aşağıdaki şekilde hesaplanır.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
S	d_{12}	d_{13}	d_{14}	d_{23}	d_{24}	d_{34}	d_{12}	d_{13}	d_{14}	d_{23}	d_{24}	d_{34}	d_{12}	d_{13}	d_{14}	d_{23}	d_{24}	d_{34}
R	f_{12}	f_{13}	f_{14}	f_{23}	f_{24}	f_{34}	f_{41}	f_{43}	f_{42}	f_{13}	f_{12}	f_{32}	f_{13}	f_{14}	f_{12}	f_{34}	f_{32}	f_{42}
A	$s_1.r_1$	$s_2.r_2$	$s_3.r_3$	$s_4.r_4$	$s_5.r_5$	$s_6.r_6$	$s_7.r_7$	$s_8.r_8$	$s_9.r_9$	$s_{10}.r_{10}$	$s_{11}.r_{11}$	$s_{12}.r_{12}$	$s_{13}.r_{13}$	$s_{14}.r_{14}$	$s_{15}.r_{15}$	$s_{16}.r_{16}$	$s_{17}.r_{17}$	$s_{18}.r_{18}$
Amaç	$2(A_1+A_2+A_3+A_4+A_5+A_6)$						$2(A_7+A_8+A_9+A_{10}+A_{11}+A_{12})$						$2(A_{13}+A_{14}+A_{15}+A_{16}+A_{17}+A_{18})$					

Şekil 4.10. *A* vektörünün hesaplanması

Bu problemde bir bireyin amaç fonksiyonu hesaplanmasında $\frac{4(4-1)}{2} = 6$ adet çarpım işlemi bulunmaktadır. Bu çarpım işlemlerinin her biri bir threade karşılık gelmek üzere GİB'de eş zamanlı olarak gerçekleştirilmekte ve her birey için *A* vektöründe tutulmaktadır. Şekil 4.10'daki gibi s_i ve r_i elemanlarının eş zamanlı çarpımı ile elde edilen *A* vektörünün altışarlı elemanlarının paralel toplama yöntemiyle toplanıp iki katının alınmasıyla bir bireyin amaç fonksiyonu değeri elde edilmektedir.

Bu örnek için, eğer elimizde 18'den fazla thread olursa, çarpma işlemlerinin tamamını bir seferde gerçekleştirebiliriz. Ancak gerek popülasyondaki birey sayısının gerekse problem boyutlarının artması, *A* vektörünün büyüklüğünün elimizdeki thread sayısından daha fazla çıkmasına yol açar. Bu yüzden, *A* vektörü bir seferde değil parça parça hesaplanmaktadır.

Yeni arama operatöründe ise popülasyondaki tüm bireylerde aynı sıradaki genlerin değişimi eş zamanlı olarak kontrol edilmektedir. Her bireydeki değişimi ele alan delta fonksiyonunun değeri, bireyin indeksiyle ilişkili delta vektörüne atılmaktadır. Aynı açıklayıcı örnek için bu durumu göz önüne alalım.

Birey	Ele alınan genler	Tesisler	Delta
Π_1	π_1 π_2	1-2	120
	π1 π3	1-3	
	π_1 π_4	1-4	
	π_2 π_3	2-3	
	π_2 π_4	2-4	
	π_3 π_4	3-4	
Π_2	π_1 π_2	4-1	-240
	π_1 π_3	4-3	
	π_1 π_4	4-2	
	π_2 π_3	1-3	
	π_2 π_4	1-2	
	π_3 π_4	3-2	
Π_3	π_1 π_2	1-3	180
	π_1 π_3	1-4	
	π_1 π_4	1-2	
	π_2 π_3	3-4	
	π_2 π_4	3-2	
	π_3 π_4	4-2	

Şekil 4.11. GİB'de yeni arama operatörü

Ele alınan örnekte popülasyonda bulunan 3 bireyin amaç fonksiyonu değerleri sırasıyla 630,790 ve 590'dır. GİB üzerinde çalışan yeni arama operatörü tüm bireyler için aynı sıradaki genlerin değişimden doğacak delta fonksiyonunu eş zamanlı olarak hesaplamaktadır. Örnekte bulunan bireylerde ilk değişim kontrolü π_1 π_2 değişimi olarak ele alınmaktadır. İlk birey için 1-2, ikinci birey için 4-1, üçüncü birey için ise 1-3 tesislerinin değişikliği kontrol edilmektedir. Bu eş zamanlı kontrolden birey 1 için elde edilen delta değeri 120 diğer bireyler için ise Şekil 4.11'deki gibi, -240 ve 180'dir. Bu durumda eş zamanlı olarak hesaplanan delta değerlerinden ikinci bireye ait olan sıfırdan küçük olduğu için kabul edilmekte, diğerleri ise reddedilmektedir. Diğer tüm değişim kontrolleri benzer şekilde çalışmaktadır.

5. HESAPSAL SONUÇLAR

Çalışmanın bu bölümünde, önerilen sıralı algoritmanın MİB'de, paralel algoritmanın ise GİB'de kodlanması sonucunda elde edilen sonuçlar tartışılmıştır. İki program için de literatürde yer alan KAP ile ilgili simetrik test problemleri üzerinde denemeler yapılmıştır. Çalışmanın özgün yanı olan kısa sürelerde etkin çözümler veren evrimsel bir algoritma özelliğini vurgulayacak şekilde karşılaştırmalara yer verilmiştir. Çözüm süreleri karşılaştırılırken çözüm kaliteleri de göz önünde bulundurulmuştur.

5.1. Bilgisayar Konfigürasyonu

Tez çalışmasının deneysel çalışmalarında kullanmak üzere Anadolu Üniversitesi Bilimsel Araştırmalar Komisyonu'nun katkılarıyla yüksek kapasiteli bir iş istasyonu edinilmiştir. Bilgisayarda 32 GB bellek ve 6 çekirdekli Intel I7 MİB ile GİB üzerinde programlamaya uygun nVIDIA GTX 580 yongalı ekran kartı bulunmaktadır. Tüm programlama işlemleri Microsoft Visual Studio 2010 ortamında gerçekleştirilmiştir. MİB birimlerinde kodlanan sıralı yöntem için C++ programlama dili kullanılmıştır. GİB birimlerinde kodlanan paralel yöntem için ise C++ ve CUDA-C programlama dilleri kullanılmıştır. GİB üzerinde programlama aracı olarak Visual Studio 2010 ortamına entegre CUDA 4.0 kullanılmıştır.

5.2. Kullanılan Parametreler

Çalışmada önerilen sıralı ve paralel yöntemlerin karşılaştırılabilmesi için, denemelerde başarılı sonuçların elde edildiği parametre değerleri kullanılmıştır. Özellikle MİB ve GİB üzerinde çalışan algoritmaların çözüm sürelerinin karşılaştırılması için elde edilen sonuçların kalitesi arasında anlamlı bir fark olmaması gereklidir. Popülasyonda 1000 birey bulunmaktadır ve bireyler 0.8 çaprazlama oranı ile yeni nesli oluşturmaktadır. Turnuva seçim aşamasında güçlü

olan, yani amaç fonksiyonu değeri daha düşük olan bireyin kazanma olasılığı 0.85'tir. Tüm bireyler mutasyon operatörüne girmekte, değişiklik olumlu ise yapılmakta, olumsuz ise 0.1 olasılıkla yapılmaktadır. Yeni arama operatörü ise tüm bireylere uygulanmaktadır. Ayrıca, programın durdurma politikası problem türüne göre değişmektedir. Optimal değeri bilinen küçük boyutlu problemlerde, o değere ulaşılıncaya dek program çalıştırılmaktadır. Büyük problemlerde ise çözüme belirli uzaklığa kadar program çalıştırılmaktadır. Ayrıca CUDA grid yapısı parametreleri amaç fonksiyonu hesaplamaları için problem boyutuyla ilişkili olarak *n* blokta *n* thread, yeni arama operatörü için ise 125 blok ve her blokta 8 thread kullanılmıştır.

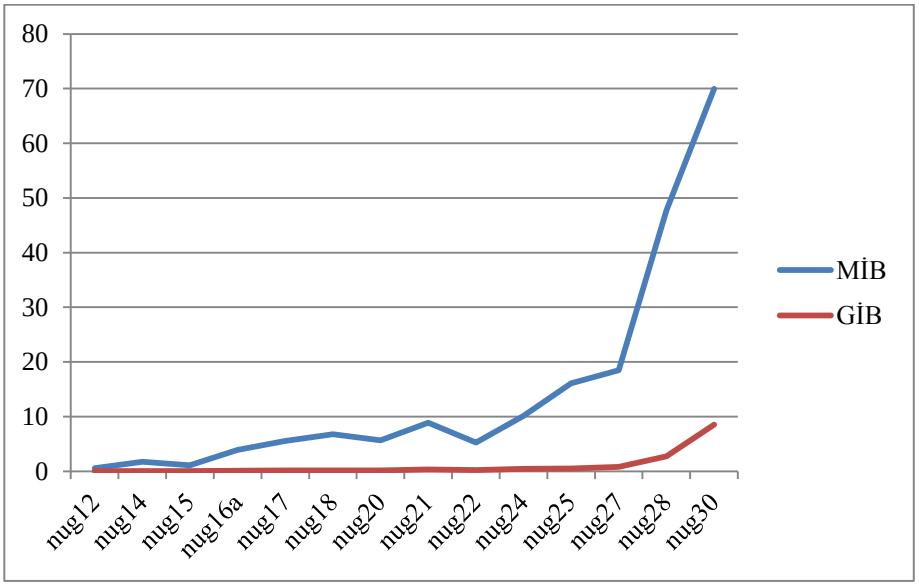
5.3. Karşılaştırmalar

Çalışmanın bu bölümünde, MİB ve GİB çözüm süreleri ve çözüm kaliteleri üzerine karşılaştırmalar yapılmaktadır. Test problemlerine dair bilinen en iyi sonuçlar ve optimal değerler Misevicius'un [52] çalışmasından ve QAPLIB [67] internet sitesinden alınmıştır. Tüm problemler için hem MİB hem GİB'de yirmişer kez program çalıştırılmıştır. Tablolarda yer alan BK sütunu, probleme ait bilinen en iyi değeri ya da optimal değeri, MİB(sn) sütunu saniye olarak MİB'de ortalama ölçüm süresini, GİB(sn) GİB'de ortalama ölçüm süresini, GAP MİB sütunu MİB'de GAP GİB sütunu ise GİB'de çözüme olan yüzde ortalama uzaklığı ($\frac{Ort.Elde\ Edilen\ Değer-BK}{BK}$), HİT sütunu bilinen en iyi çözüme ulaşılma sayısını ve son sütunda GİB ile MİB çözüm süreleri arasındaki hızlanma faktörü ($\frac{MİB(sn)}{GİB(sn)}$) görülmektedir.

Tablo 5.1. Nugent test problemlerine ait ölçümler

Problem	BK	MİB(sn)	GİB(sn)	GAP MİB	GAP GİB	HİT	Hızlanma
Nug12	578	0.585	0.035	0	0	20/20	16.71x
Nug14	1014	1.747	0.064	0	0	20/20	27.30x
Nug15	1150	1.068	0.046	0	0	20/20	23.22x
Nug16a	1610	3.917	0.083	0	0	20/20	47.46x
Nug17	1732	5.522	0.154	0	0	20/20	35.79x
Nug18	1930	6.756	0.174	0	0	20/20	38.74x
Nug20	2570	5.637	0.154	0	0	20/20	36.58x
Nug21	2438	8.904	0.309	0	0	20/20	28.83x
Nug22	3596	5.247	0.189	0	0	20/20	27.80x
Nug24	3488	10.132	0.446	0	0	20/20	22.70x
Nug25	3744	16.086	0.509	0	0	20/20	31.58x
Nug27	5234	18.476	0.789	0	0	20/20	23.42x
Nug28	5166	47.685	2.734	0	0	20/20	17.44x
Nug30	6124	69.989	8.549	0.0005	0.0004	15/20	8.19x
Ortalama		**14.411**	**1.017**	**0.0000**	**0.0000**		**27.55x**

Tablo 5.1'de Nugent test problemlerine ait ölçümler yer almaktadır. Nug30 problemi hariç diğer tüm problemlerde, program her çalıştırıldığında bilinen en iyi sonuçlara ulaşılmıştır. Nug30 problemi için ise bu oran %75'tir (15/20). Bu problem için çözüme olan ortalama yüzde uzaklık MİB için 0.0005 (%0.05), GİB için 0.0004 (%0.04) gibi oldukça düşük değerlerdir.



Şekil 5.1. Nugent test problemleri ölçüm süreleri

Nugent test problemleri kümesi için 8.19 ile 47.46 kat arasında ortalama 27.55 kat GİB'de MİB'ye göre daha hızlı çözümler elde edilmektedir. GİB'de Nug12'den Nug27'ye kadar olan problemler 1 saniyenin altında %100(20/20) başarı ile çözülmüştür. Her tekrarda bilinen en iyi çözüme ulaşılan bir başka problem olan Nug28'e MİB'de ortalama 47.7 saniye gibi bir sürede çözüme ulaşılırken GİB'de ise bu süre yalnızca 2.7 saniyedir. Şekil 5.1'de MİB ve GİB'de çözüm sürelerinin artışı izlenmektedir.

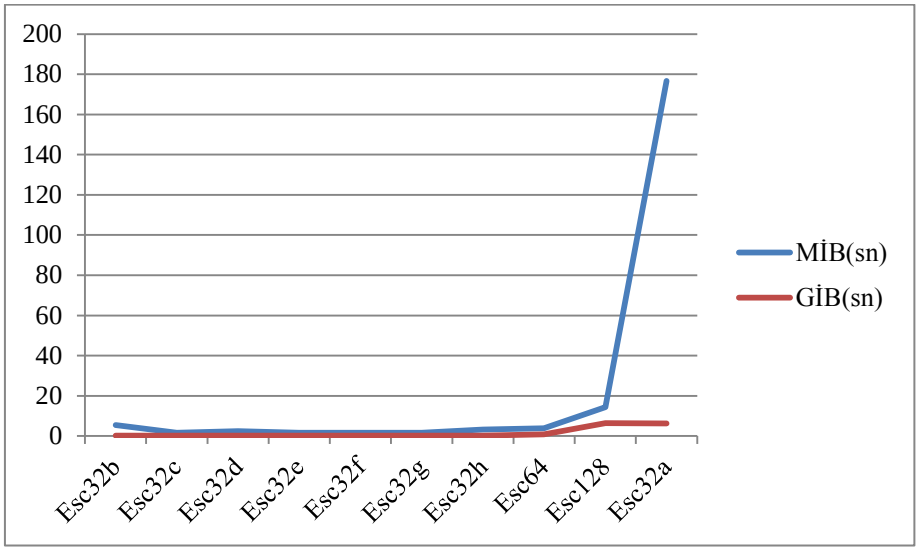
Tablo 5.2. Escherman test problemlerine ait ölçümler

Problem	BK	MİB(sn)	GİB(sn)	GAP MİB	GAP GİB	HİT	Hızlanma
Esc32b	168	5.493	0.229	0	0	20/20	23.99x
Esc32c	642	1.553	0.154	0	0	20/20	10.08x
Esc32d	200	2.360	0.155	0	0	20/20	15.23x
Esc32e	2	1.558	0.153	0	0	20/20	10.18x
Esc32f	2	1.554	0.150	0	0	20/20	10.36x
Esc32g	6	1.553	0.152	0	0	20/20	10.22x
Esc32h	438	3.216	0.158	0	0	20/20	20.35x
Esc64a	116	3.837	0.872	0	0	20/20	4.40x
Esc128	64	14.440	6.322	0	0	20/20	2.28x
Esc32a	130	176.651	6.195	0.0006	0.0007	19/20	28.52x
Ortalama		**21.221**	**1.454**	**0.00006**	**0.00007**		**13.56x**

Literatürde yer alan diğer bir test problemi kümesi ise Escherman problemleridir. 16 boyutludan 128 boyutluya kadar olan bu problemlerin tümünde bilinen en iyi sonuçlara ulaşılmıştır. Esc16a, Esc16b, Esc16c, Esc16d, Esc16e, Esc16f, Esc16g, Esc16h, Esc16i ve Esc16j isimli problemlerde her çözüm denemesinde ilk tekrarda optimal sonuçlara ulaşılmaktadır. Bu nedenle, bu problemlerin karşılaştırmaları tablolara yansıtılmamıştır. Bahsi geçen bu problemler hariç, diğer tüm Escherman problemleri için karşılaştırmalı sonuçlar

Tablo 5.2'de yer almaktadır. Bu test problemleri kümesinde, tüm örnekler için bilinen en iyi değerlere ulaşılmıştır. 32 boyutlu Esc32a hariç, diğer tüm problemlerde en iyi çözüme hem MİB'deki hem de CUDA ile yazılan GİB'deki programda her tekrarda (20/20) erişilmiştir. Esc32a problemi ise bu seride bilinen en zor problemdir. Bu problemde ise başarı %95 (19/20) gibi oldukça yüksek bir orandır. Ayrıca aynı problem için MİB'de ortalama 176.7 saniyede sonuca

ulaşılırken bu süre GİB için yalnızca 6.2 saniyedir. Esc32a için bu sürelerden elde edilen hızlanma faktörü 28.52 kattır. Bu test problemleri kümesi için elde edilen hızlanma faktörleri 2.28 kat ile 28.52 kat arasında değişmektedir. Tüm problemler göz önüne alındığında ortalama hızlanma faktörü 13.56 kat olarak karşımıza çıkmaktadır. Şekil 5.2'de KAP literatüründe önemli bir yeri olan Escherman test problemlerine ait çalışmada elde edilen MİB ve GİB çözüm sürelerinin karşılaştırması görülmektedir.



Şekil 5.2. Escherman test problemleri ölçüm süreleri

Kesin yöntemler kullanılarak yapılan son çalışmalardan birinde Fischetti ve ark [68], Esc32a problemini ele almamışlar fakat Esc32c problemini 9643.82 saniyede, Esc32d problemini ise 2973.26 saniyede çözmüşlerdir. Yine son çalışmalardan birinde Misevicius [52] metasezgisel yaklaşımıyla bu problemleri sırasıyla 2.1 ve 2.0 saniyede çözmüştür. Bu problemler için tez çalışmasında %100 başarı ile elde edilen çözüm süreleri ise sırasıyla MİB'de 2.36 saniye ve 1.558 saniyedir. Bu çözüm süreleri GİB'de ise sırasıyla 0.154 saniye ve 0.155 saniyedir.

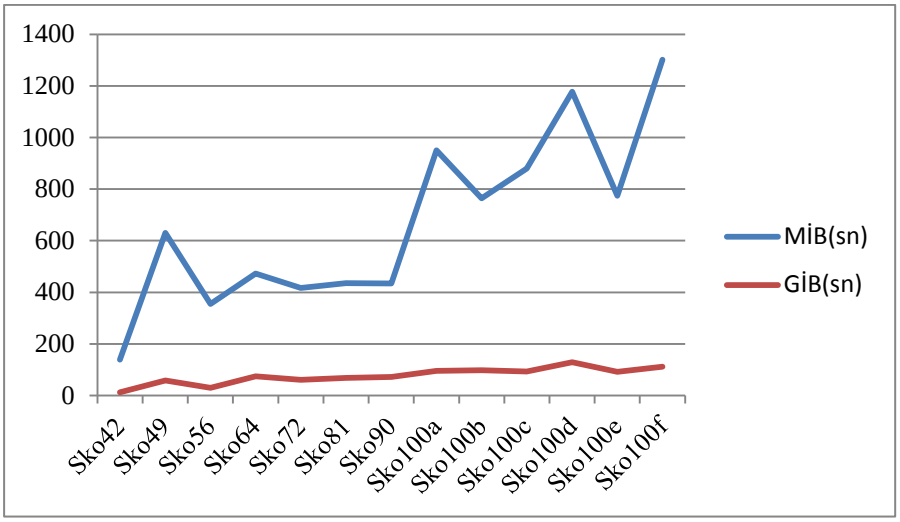
KAP literatüründe yer alan bir başka simetrik problem kümesi ise Skorin problemleridir. 42 ile 100 arasında boyutları değişen bu problemlerin bazılarında bilinen en iyi değerlere ulaşılırken, bazılarında bilinen en iyi değerlere 0.005'ten (%0.5) daha küçük uzaklıklara ulaşılmıştır. Tablo 5.3'te Skorin problemlerine ait

ölçümler bulunmaktadır. Ölçümlerde, çözüme olan uzaklıklar 0.005'in altına indiğinde program durdurulmuş ve karşılaştırılmalar yapılmıştır. MİB ve GİB çözüm sürelerinin karşılaştırılması için çözüme olan uzaklıkların ortalamalarındaki farkın istatistiksel olarak anlamlı olup olmadığının araştırılması için *t* testi yapılmıştır. %99 (p=0.716) güven düzeyiyle MİB ve GİB'den alınan ölçümlerin çözüme olan ortalama uzaklıkları arasında anlamlı bir fark olmadığı ortaya konmuştur.

Tablo 5.3. Skorin test problemlerine ait ölçümler

Problem	BK	MİB(sn)	GİB(sn)	GAP MİB	GAP GİB	HİT	Hızlanma
Sko42	15812	139.040	12.090	0.0001	0.0001	19/20	11.50x
Sko49	23386	630.635	57.635	0.0012	0.0012	1/20	10.94x
Sko56	34458	354.380	29.965	0.0011	0.0013	0/20	11.83x
Sko64	48498	473.065	73.845	0.0006	0.0003	7/20	6.41x
Sko72	66256	416.802	60.861	0.0039	0.0043	0/20	6.85x
Sko81	90998	434.909	67.809	0.0017	0.0018	0/20	6.41x
Sko90	115534	433.990	72.158	0.0038	0.0037	0/20	6.01x
Sko100a	152002	950.996	94.941	0.0029	0.0032	0/20	10.02x
Sko100b	153890	763.755	98.125	0.0019	0.0027	0/20	7.78x
Sko100c	147862	879.414	93.312	0.0030	0.0031	0/20	9.42x
Sko100d	149576	1177.933	129.301	0.0019	0.0022	0/20	9.11x
Sko100e	149150	773.763	92.227	0.0017	0.0019	0/20	8.39x
Sko100f	149036	1301.036	111.872	0.0028	0.0031	0/20	11.63x
Ortalama		**671.516**	**76.472**	**0.002**	**0.002**		**8.95x**

Elde edilen hızlanma faktörleri, ortalama 8.95 kattır . Problemlerin zorluk derecesinin fazla ve boyutlarının büyük oluşu çözüm sürelerini artırsa da Sko42 problemi için GİB'de yaklaşık 12 saniyede %95 başarı ile bilinen en iyi değer elde edilmiştir. Sko100 problemlerinde ise 0.0032 (%0.32) ve daha küçük çözüme uzaklıklarla GİB'de MİB'ye göre ortalama 9.39 kat daha hızlı çözümler elde edilmiştir. Şekil 5.3'te Skorin test problemlerinin ölçüm sürelerinin karşılaştırılması yer almaktadır.

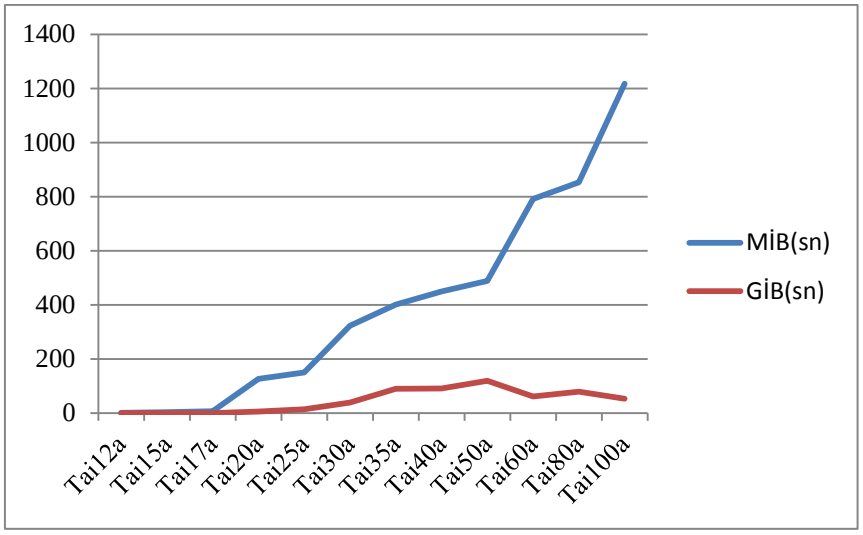


Şekil 5.3. Skorin test problemleri ölçüm süreleri

KAP literatüründe sıklıkla en zor problemler olarak yorumlanan Taillard problemleri çalışmada ele alınan bir diğer test problemleri kümesidir. Öyle ki problemlerden bazıları için bilinen en iyi sonuçlar Misevicius'unki [52] gibi çalışmalarda hala geliştirilmektedir ve geliştirmeye açıktır. Tez çalışmasında Taillard problemlerinin bazıları içinin bilinen en iyi sonuçlara ulaşılırken, bazıları için çözüme 0.02'den daha az (%2) uzaklıklar elde edilmiştir.

Tablo 5.4. Taillard test problemlerine ait ölçümler

Problem	BK	MİB(sn)	GİB(sn)	GAP MİB	GAP GİB	HİT	Hızlanma
Tai12a	224416	0.467	0.028	0	0	20/20	16.83x
Tai15a	388214	2.967	0.164	0	0	20/20	18.15x
Tai17a	491812	6.406	0.125	0	0	20/20	51.27x
Tai20a	703482	126.707	5.637	0	0	20/20	22.48x
Tai25a	1167256	149.951	13.999	0.0036	0.0034	5/20	10.71x
Tai30a	1818146	322.371	38.664	0.0031	0.0032	5/20	8.34x
Tai35a	2422002	400.977	89.797	0.0100	0.0091	1/20	4.47x
Tai40a	3139370	449.663	91.118	0.0130	0.0130	0/20	4.93x
Tai50a	4938796	487.824	119.537	0.0199	0.0197	0/20	4.08x
Tai60a	7208572	791.636	61.517	0.0199	0.0199	0/20	12.87x
Tai80a	13515450	853.236	78.646	0.0199	0.0199	0/20	10.85x
Tai100a	21054656	1217.771	52.467	0.0199	0.0199	0/20	23.21x
Ortalama		**400.831**	**45.975**	**0.009**	**0.009**		**15.68x**



Şekil 5.4. Taillard test problemleri ölçüm süreleri

Skorin test problemlerinde olduğu gibi MİB ve GİB karşılaştırılması yapabilmek için çözüme olan uzaklıkların ortalamalarındaki farkın istatistiksel olarak anlamlı olup olmadığı araştırılmıştır. %99 (p=0.976) güven düzeyiyle MİB ve GİB'de çözüme olan ortalama uzaklıkları arasında anlamlı bir fark olmadığı yapılan t testiyle anlaşılmıştır. Bu test problemleri için GİB'de MİB'ye göre en düşük 4.08 kat en yüksek 51.27 kat, ortalama ise 15.68 kat daha hızlı çözümler elde edilmiştir. Problemlerden Tai12a, Tai15a, Tai17a, Tai20a, Tai25a, Tai30a ve Tai35a'nın bilinen en iyi değerlerine ulaşılırken, kalan problemlerde 0.02(%2) ve daha az çözüme ortalama uzaklık elde edilmiştir. Tablo 5.4'te Taillard test problemlerine ait MİB ve GİB çözüm süreleri, çözüme olan ortalama uzaklıklar ile hızlanma faktörleri bulunmaktadır. Şekil 5.4'te ise çözüm sürelerinin karşılaştırılması yer almaktadır.

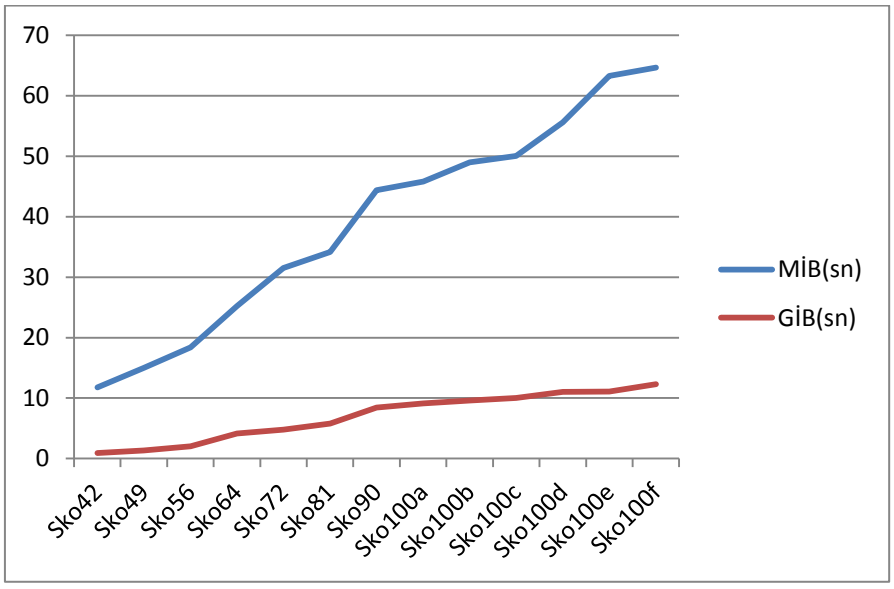
5.4. Farklı Denemelerle Karşılaştırmalar

Bir önceki bölümde ele alınan test problemleri kümelerinden Skorin problemleri için 0.005 (%0.5) ve daha az çözüme olan uzaklıklar için karşılaştırmalar yapılmıştır. Bu bölümde ise öncelikle aynı test problemleri kümesi için 0.01 (%1) ve daha az çözüme olan uzaklıklar için MİB ve GİB'de elde edilen çözüm süreleri ele alınmaktadır.

Tablo 5.5. Skorin test problemleri %1'lik ölçümler

Problem	BK	MİB(sn)	GİB(sn)	Hızlanma
Sko42	15812	11.760	0.909	12.94x
Sko49	23386	14.988	1.353	11.08x
Sko56	34458	18.390	2.039	9.02x
Sko64	48498	25.226	4.155	6.07x
Sko72	66256	31.499	4.812	6.55x
Sko81	90998	34.163	5.784	5.91x
Sko90	115534	44.369	8.418	5.27x
Sko100a	152002	45.809	9.149	5.01x
Sko100b	153890	48.976	9.609	5.10x
Sko100c	147862	50.023	10.001	5.00x
Sko100d	149576	55.568	11.033	5.04x
Sko100e	149150	63.232	11.104	5.69x
Sko100f	149036	64.606	12.303	5.26x
Ortalama		**39.124**	**6.975**	**6.76x**

MİB ve GİB'den alınan ölçümler her bir problem için 0.01 (%1) çözüme olan uzaklığa düşene kadar programların çalıştırılması ile elde edilmiştir. Buna göre GİB'deki çözüm süreleri MİB'ye göre ortalama 6.76 kat daha hızlı ulaşılmıştır. Bu oran Sko42 probleminde 12.94 iken, Sko100b probleminde 5.1 olarak gözlemlenmiştir. Tablo 5.5'te tüm ölçüm süreleri, Şekil 5.5'te ise çözüm sürelerindeki artış izlenmektedir.



Şekil 5.5. Skorin test problemleri %1'lik ölçüm süreleri

Tez çalışmasındaki konu ile örtüşen literatüre son zamanlarda kazandırılan makalede [61] GİB üzerinde CUDA ile KAP için uygulamalar yapılmış ve algoritmanın başarısı için Skorin problemleri incelenmiştir. Bu problemlerden Sko100d problemini ele alacak olursak, söz konusu makalede önerilen yöntem ile GİB'de 0.066'lık (%6.6) çözüme uzaklığa 247 saniyede ulaşılmakta 0.001'den (%0.1) daha az çözüme uzaklığa 742 saniyede ulaşılmıştır. Tez çalışmasında ise aynı problem için GİB'de 0.01'lik (%1) dilime ortalama 12 saniyede, 0.0027'lik (%0.27) dilime ise 130 saniyede ulaşılmıştır.

Farklı bir denemede ise CUDA grid parametrelerinin, GİB'de çalışan program üzerindeki etkisi ele alınmıştır. Esc64a, Tai80a, Sko90 ve Sko100a gibi farklı boyutlardaki problemler için bir tekrarda tüm popülasyon için amaç fonksiyonunu değerlendirilmesi ve lokal aramada geçen toplam süreler üzerinde durulmuştur. Bölüm 4.3'te bahsedildiği gibi GİB üzerinde yeni arama operatörü eş zamanlı olarak ele alınmıştır. Çarpımları popülasyondaki birey sayısına eşit olacak şekilde farklı grid parametreleri belirlenmiştir. Tablo 5.6'da satırlarda problem büyüklüklerine göre, sütunlarda ise farklı grid parametrelerine göre ölçüm süreleri saniye olarak yer almaktadır. Tabloda 250/4, 125/8 vb. gösterimi gridin blok sayısı / her bloktaki thread sayısını göstermektedir. Örneğin 250/4 gösteriminde 250 blok ve her blokta 4 thread bulunmaktadır. Alınan farklı ölçümlere göre grid parametrelerinin çözüm süresi üzerinde etkili olduğu anlaşılmaktadır. Tüm problem boyutlarında en kısa sürede alınan ölçümler 125/8 grid parametrelerinden elde edilmiştir.

Tablo 5.6. Farklı grid parametreleri ile alınan ölçümler

Blok/Thread n	**250/4**	**125/8**	**100/10**	**50/20**	**25/40**	**8/125**
64	0.577	0.386	0.402	0.496	0.602	0.612
80	1.129	0.747	0.785	0.967	1.187	1.216
90	1.598	1.052	1.108	1.367	1.676	1.722
100	2.195	1.428	1.509	1.870	2.297	2.368

Bir başka deneme ise yeni arama operatörü yerine 2-opt yerel arama algoritmasının kullanılması karşılaştırılmıştır. 2-opt yerel arama algoritmasında her birey için tüm ikili değişiklikler kontrol edilmektedir. Eğer çözümü iyileştiren ikili değişimler söz konusu ise bunlardan en iyi olan değişiklik gerçekleştirilmektedir. Nug30 problemi için yeni arama operatörü ile 20 denemede 15 kez bilinen en iyi değere ulaşılırken, algoritmada yeni arama operatörü yerine 2-opt yerel arama algoritması uygulandığında 20 denemede hiçbir kez bilinen en iyi değere ulaşılamamıştır. 20 denemede bu problem için yeni operatörü içeren algoritmada çözüme olan ortalama uzaklık 0.0005 (%0.05) iken 2-opt içeren algoritmada bu değer 0.023'tür (%2.3). Buradan yeni arama operatörünün etkinliği anlaşılmaktadır.

6. SONUÇ VE ÖNERİLER

Çalışmada kombinatorik optimizasyon problemlerinden karesel atama problemi hakkında genel bilgi verilerek problemin çözüm yöntemlerinden bahsedilmiştir. Özellikle metasezgisel yöntemlerden evrimsel algoritmalar üzerinde durulmuştur. Ayrıca ekran kartları üzerinde yer alan grafik işlem birimleri ve hesaplamalı bilimlerde grafik işlem birimlerinin kullanımı hakkında açıklamalara yer verilmiştir. Yoğun işlem gerektiren hesaplamalarda grafik işlem birimleri kullanımının artan bir eğilimle yaygınlaştığı vurgulanmıştır. Literatürde karesel atama probleminin çözümü için birçok metasezgisel çalışma mevcut iken, grafik işlem birimleri üzerinde yapılan metasezgisel çalışmalara çok sınırlı sayıda rastlanmıştır. Bu çalışmanın literatüre katkısı, KAP'ın çözümü için başarılı sonuçlar elde eden sıralı ve paralel olmak üzere birer algoritma sunulmasının yanı sıra, GİB'yi kullanarak kısa sürelerde çözümlere ulaşılmasıdır.

Kesin yöntemlerle, literatürde karesel atama problemi için yer alan test problemlerine makul sürelerde çözüm bulunamadığı açıklanmıştır. Çalışmada, problemin çözümü için başarılı ve tutarlı sonuçlar veren popülasyon temelli evrimsel bir algoritma önerilmiştir. Algoritmanın operatörlerine ait parametreler, popülasyon içi çeşitliliği artırmak ve erken yakınsama probleminden uzaklaşmak için özel olarak belirlenmiştir. Evrimsel algoritmaya keşfetme yeteneğinin yanı sıra, derinlemesine arama yeteneği de kazandırılması için bir yeni arama operatörü eklenmiştir.

Önerilen algoritma, merkezi işlem birimlerinde ölçüm almak için C++ programlama diliyle kodlanmıştır. Algoritmada, en çok zaman alan kısımları, popülasyonda bulunan bireylerin amaç fonksiyonun hesaplanması ve yeni arama operatörüne paralel bir yapı kazandırılmıştır. Problemin çözüm sürelerinde anlamlı azalma beklentisiyle, algoritmanın paralelleştirilen operatörleri CUDA 4.0 kullanılarak grafik işlem birimleri üzerinde kodlanmıştır.

Algoritmanın etkinliğinin gösterilmesinin yanı sıra, iki yöntem arasında sonuç olarak bir fark olmadığının istatistiksel testler ile anlaşılması ve grafik işlem birimlerinde elde edilen çözüm sürelerinin, merkezi işlem birimlerine göre daha kısa olduğunun gösterilmesi için karşılaştırmalar yapılmıştır. Algoritma

çalışmada ele alınan 59 test probleminin 43'nde bilinen en iyi değerlere ulaşmıştır. Çözüm sürelerinin ölçümü sonucunda grafik işlem birimlerinde elde edilen çözümler, merkezi işlem birimlerine göre 51 kata kadar, ortalama 17 kat daha hızlı olduğu gözlemlenmiştir.

Elde edilen hızlanma katsayılarını, grafik işlem birimleri üzerinde program yazmanın daha kolay bir yapı alacağı öngörüsü ile algoritmanın diğer operatörlerine de paralel bir yapı kazandırılarak, daha yüksek seviyelere taşınması mümkün olabilir. Ayrıca önerilen sıralı ve paralel yöntemin esnekliği sayesinde karesel atama problemi dışında yer alan diğer kombinatorik optimizasyon problemlerinin çözümünde de kullanılabilir.

7. KAYNAKLAR

[1] T. S. Crow, *Evolution of the Graphical Processing Unit,* University of Nevada, 2004.

[2] T. C. Koopmans ve M. J. Beckmann, "Assignment problems and the location of economic activities," *Econometrica,* **25**, 53-76, 1957.

[3] S. Sahni ve T. Gonzalez, "P-complete approximation problems," *Journal of the Association of Computing Machinery,* **23**, 555-565, 1976.

[4] V. Boyer, D. E. Baz ve M. Elkihel, "Solving knapsack problems on GPU," *Computers & Operations Research,* **39**, 42-47, 2012.

[5] M. Czapiński ve S. Barnes, "Tabu Search with two approaches to parallel flowshop evaluation on CUDA platform," *J. Parallel Distrib. Comput.,* **71**, 802-811, 2011.

[6] J. Li, X. Lv ve L. Liu, "A Parallel Genetic Algorithm with GPU Accelerated for Large-scale MDVRP in Emergency Logistics," *IEEE International Conference on Computational Science and Engineering*, 2011.

[7] G. Hasle ve C. Schulz, "Neighborhood Evaluation on GPU for the DVRP," SINTEF ICT, Oslo, 2011.

[8] C. Schulz, G. Hasle, O. Kloster, A. Riise ve M. Smedsrud, "Parallel Local search for the CVRP on the GPU," SINTEF ICT, Oslo, 2011.

[9] S. Sanci ve V. İsler, "A Parallel Algorithm for UAV Flight Route Planning on GPU," *International Journal of Parallel Programming,* **39**, 809-837, 2011.

[10] N. Fujimoto ve S. Tsutsui, "A Highly-Parallel TSP Solver for a GPU Computing Platform," *Numerical Methods and Applications,* **6046**, 264-271, 2011.

[11] A. Schrijver, "On the history of combinatorial optimization," *Handbooks in OR & MS,* **12**, 1-68, 2005.

[12] R. E. Burkard, E. Çela, P. M. Pardalos ve L. S. Pisoulis, "The Quadratic Assignment Problem," Spezialforschungsbereich F 003 "Optimierung und

Kontrolle", Graz, 2007.

[13] E. Lawler, "The quadratic assignment problem," *Management Science,* **9**, 586-599, 1963.

[14] L. Kaufmann ve F. Broeckx, "An algorithm for the quadratic assignment problem using Benders' decomposition," *European Journal of Operational Research,* **2**, 204-211, 1978.

[15] A. Frieze ve J. Yadegar, "On the quadratic assignment problem," *Discrete Applied Mathematics,* **5**, 89-98, 1983.

[16] W. Adams ve T. Johnson, "Improved linear programming-based lower bounds for the quadratic assignment problem," *DIMACS Series on Discrete Mathematicsand Theoretical Computer Science,* **16**, 43-75, 1994.

[17] A. Barvinok, "Computational complexity of orbits in representations of symmetric groups," *Advances in Soviet Mathematics,* **9**, 161-182, 1992.

[18] M. Jünger ve V. Kaibel, "A basic study of the QAP polytope," Universitat zu Koln, Köln,Germany, 1996.

[19] E. Çela, The quadratic assignment problem theory and algorithms, Dordrecht: Kluver Academic Publishers, 1998.

[20] P. Hahn ve T. Grant, "Lower bounds for the quadratic assignment problem based upon a dual formulation," *Operations Research,* **46**, no. 6, 912-922, 1998.

[21] P. Pardalos, K. Ramakrishnan, M. Resende ve Y. Li, "Implementation of a variable reduction based lower bound in a branch and bound algorithm for the quadratic assignment problem," *Journal on Optimization,* **7**, 280-294, 1997.

[22] M. S. Bazaraa ve H. D. Sherali, "Benders' partitioning scheme applied to a new formulation of the quadratic assignment problem," *Naval Research Logistics Quarterly,* **27**, 29-41, 1980.

[23] E. Balas ve J. Mazzola, "Quadratic 0-1 programming by a new linearization," *ORSA/TIMS National Meeting,* Washington D.C., 1980.

[24] Ö. Üstün, *Kareli atama probleminin çözümü için subgradient yöntemi,*

Eskişehir, 2001.

[25] R. Gasimov ve Ö. Üstün, "Solving the quadratic assignment problem using F-MSG algorithm," *Journal of Industrial and Management Optimization,* **3**, 173-191, 2007.

[26] P. Gilmore, "Optimal and suboptimal algorithms for the quadratic assignment problem," *Journal on Applied Mathematics,* **10**, 305-313, 1962.

[27] E. Buffa, G. Armour ve T. Vollmann, "Allocating facilities with CRAFT," *Harvard Business Review,* **42**, 136-158, 1962.

[28] P. Pardalos, F. Rendl ve H. Wolkowicz, "The Quadratic Assignment Problem: A Survey and Recent Developments," *Series in Discrete Mathematics and Theoretical Computer Science,* **16**, 1-42, 1994.

[29] E.-G. Talbi, Metaheuristics, New Jersey: Wiley, 2009.

[30] S. Kirkpatrick, C. Gelat ve M. Vecchi, "Optimization by Simulated Annealing," *Science,* **220**, 671-680, 1983.

[31] R. Burkard ve F. Rendl, "A thermodynamically motivated simulation procedure for combinatorial optimization problems," *European Journal of Operational Research,* **17**, 169-174, 1984.

[32] M. Wilhelm ve T. Ward, "Solving quadratic assignment problems by 'simulated annealing'," *IIE Transactions,* **19**, 107-119, 1987.

[33] N. Abreu, T. Querido ve P. Boaventura-Netto, "A simulated annealing for the quadratic assignment problem," *Rairo-Operations Research,* **33**, 249-273, 1999.

[34] T. Stützle ve M. Dorigo, "ACO Algorithms for the Quadratic Assignment Problem," New Ideas in Optimization,McGraw-Hill, 1999.

[35] M. Dorigo, V. Maniezzo ve A. Colorni, "The Ant System: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems,* **26**, 1-13, 1996.

[36] F. Glover, "Tabu Search - Part I," *ORSA Journal on Computing,* **1**, 190-206, 1989.

[37] J. Skorin-Kapov, "Tabu Search Applied to the Quadratic Assignment

Problem," *ORSA Journal on Computing,* **2**, 33-45, 1990.

[38] E. D. Taillard, "Robust taboo search for the quadratic assignment problem," *Parallel Computing,* **17**, 443-455, 1991.

[39] A. Misevičius ve A. Ostreika, "Defining Tabu tenure for the Quadratic Assignment Problem," *Information Technology and Control,* **36**, 2007.

[40] J. H. Holland, "Genetic Algorithms," *Scientific American,* **267**, 66-72, 1992.

[41] J. Kochhar, B. Foster ve S. Heragu, "A genetic algorithm for the unequal area facility layout problem," *Computers & Operations Research,* **25**, 583-594, 1998.

[42] Z. Drezner, "A New Genetic Algorithm for the Quadratic Assignment Problem," *Informs Journal on Computing,* **15**, 320-330, 2003.

[43] L. Yong, M. Pardolos ve G. Mauricio, "A Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment Problem," *DIMACS Series in Discrete Mathematics and Theoretical Computer Science,* 1994.

[44] R. K. Ahuja, J. B. Orlin ve A. Tiwari, "A greedy genetic algorithm for the quadratic assignment problem," *Computers & Operations Research ,* **27**, 917-934, 2000.

[45] M.-H. L. Yuan Yu ve S. Omatu, "Extensive Testing of a Hybrid Genetic Algorithm for Solving Quadratic Assignment Problems," *Computational Optimization and Applications,* **23**, 47-64, 2002.

[46] M. El-Baz, "A genetic algorithm for facility layout problems of different manufacturing environments," *Computers & Industrial Engineering,* **47**, 233-246, 2004.

[47] Z. Drezner, "Compounded genetic algorithms for the quadratic assignment problem," *Operations Research Letters,* **33**, 475-480, 2005.

[48] Y. Wu ve P. Ji, "Solving the Quadratic Assignment Problems by a Genetic Algorithm with a New Replacement Strategy," *International Journal of Computational Intelligence,* **4**, 225-229, 2007.

[49] Z. Drezner, "Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem," *Computers & Operations*

Research, **35,** 717-736, 2008.

[50] A. Ramkumar, S. Ponnambalam ve N. Jawahar, "A new iterated fast local search heuristic for solving QAP formulation in facility layout design," *Robotics and Computer-Integrated Manufacturing,* **25,** 620-629, 2009.

[51] T. James, C. Rego ve F. Glover, "A cooperative parallel tabu search algorithm for the quadratic assignment problem," *European Journal of Operational Research,* **195,** 810-826, 2009.

[52] A. Misevicius, "An implementation of the iterated tabu search algorithm for the quadratic assignment problem," *OR Spectrum,* 34, 665-690, 2012.

[53] "Developer Centers: CUDA Zone," nVIDIA, [Çevrimiçi]. Available: http://developer.nvidia.com/cuda/what-cuda. [1 Ekim 2012 tarihinde erişilmiştir].

[54] nVIDIA, "NVIDIA's Next Generation CUDA Computer Architecture Fermi," nVIDIA, 2011.

[55] M. Harris, "Optimizing Parallel Reduction in CUDA," nVIDIA, 2010.

[56] D. B. Kirk ve W.-m. W. Hwu, Programming Massively Parallel Processors, Massachusetts,USA: Elsevier Inc., 2010.

[57] S. Tsutsui ve N. Fujimoto, "Solving Quadratic Assignment Problems by Genetic Algorithms with GPU Computation: A Case Study," *GECCO,* Montreal, 2009.

[58] S. Tsutsui ve N. Fujimoto, "Fast QAP Solving by ACO with 2-opt Local Search on a GPU," *IEEE Section Congres,* San Francisco, 2011.

[59] N. Melab, T.-V. Luong, K. Boufaras ve E.-G. Talbi, "Towards ParadisEO-MO-GPU: a Framework for GPU-based Local Search Metaheuristics," *11th International Work-Conference on Artificial Neural Networks,* Malaga, 2011.

[60] T. V. Luong, N. Melab ve E.-G. Talbi, "Parallel Hybrid Evolutionary Algorithms on GPU," *IEEE World Congress on Computational Intelligence,* Barcelona, 2010.

[61] M. Czapinski, "An effective Parallel Multistart Tabu Search for Quadratic Assignment Problem CUDA platform," *J. Parallel Distrib. Comput.*

doi:10.1016/j.jpdc.2012.07.014 In Press.

[62] J. M. Cecilia, J. M. Garcia, A. Nisbet, M. Amos ve M. Ujaldon, "Enhancing data parallelism for Ant Colony Optimization on GPUs," *J. Parallel Distrib. Comput. doi:10.1016/j.jpdc.2012.01.002 In Press.*

[63] A. Delevacq, P. Delisle, M. Gravel ve M. Krajecki, "Parallel Ant Colony Optimization on Graphics Processing Units," *J. Parallel Distrib. Comput. doi:10.1016/j.jpdc.2012.01.003 In Press.*

[64] C. Shulz, "Efficient local search on the GPU—Investigations on the vehicle routing problem," *J. Parallel Distrib. Comput. doi:10.1016/j.jpdc.2012.02.020 In Press.*

[65] C. Groer, B. Golden ve E. Wasil, "A Parallel Algorithm for the Vehicle Routing Problem," *INFORMS Journal on Computing,* **23**, 315-330, 2011.

[66] "Modified genetic algorithms for solving fuzzy flow shop scheduling problems and their implementation with CUDA," *Expert Systems with Applications,* **39**, 4999-5005, 2012.

[67] R. Burkard, E. Çela, S. Karisch ve F. Rendl, "QAPLIB," [Çevrimiçi]. Available: http://www.opt.math.tu-graz.ac.at/qaplib/inst.html. [15 Ekim 2012 tarihinde erişilmiştir].

[68] M. Fischetti, M. Monaci ve D. Salvagnin, "Three Ideas for the Quadratic Assignment Problem," *Operations Research,* **60**, 954-964, 2012.