BENZETİLMİŞ TAVLAMA ALGORİTMASININ GRAFİK İŞLEMCİ ÜNİTESİ KULLANILARAK PARALELLEŞTİRİLMESİ

2017
DOKTORA TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ

Emrullah SONUÇ

BENZETİLMİŞ TAVLAMA ALGORİTMASININ GRAFİK İŞLEMCİ ÜNİTESİ KULLANILARAK PARALELLEŞTİRİLMESİ

Emrullah SONUÇ

Karabük Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalında
Doktora Tezi
Olarak Hazırlanmıştır

KARABÜK
Haziran 2017

Emrullah SONUÇ tarafından hazırlanan "BENZETİLMİŞ TAVLAMA ALGORİTMASININ GRAFİK İŞLEMCİ ÜNİTESİ KULLANILARAK PARALELLEŞTİRİLMESİ" başlıklı bu tezin Doktora Tezi olarak uygun olduğunu onaylarım.

Yrd. Doç. Dr. Şafak BAYIR .....
Tez Danışmanı, Bilgisayar Mühendisliği Anabilim Dalı

Yrd. Doç. Dr. Baha ŞEN .....
Tez Danışmanı, Yıldırım Beyazıt Üniversitesi

Bu çalışma, jürimiz tarafından oy birliği ile Bilgisayar Mühendisliği Anabilim Dalında Doktora tezi olarak kabul edilmiştir. 30/06/2017

Ünvanı, Adı SOYADI (Kurumu)	İmzası
Başkan : Prof. Dr. Mehmet AKBABA (KBÜ)	.....
Üye : Prof. Dr. Abdullah ÇAVUŞOĞLU (YÖK)	.....
Üye : Prof. Dr. Fatih Vehbi ÇELEBİ (AYBÜ)	.....
Üye : Yrd. Doç. Dr. Baha ŞEN (AYBÜ)	.....
Üye : Yrd. Doç. Dr. İlker TÜRKER (KBÜ)	.....
Üye : Yrd. Doç. Dr. Özden İŞBİLİR (KBÜ)	.....
Üye : Yrd. Doç. Dr. Şafak BAYIR (KBÜ)	.....

...../...../2017

KBÜ Fen Bilimleri Enstitüsü Yönetim Kurulu, bu tez ile, Doktora derecesini onamıştır.

Prof. Dr. Nevin AYTEMİZ .....
Fen Bilimleri Enstitüsü Müdürü

"Bu tezdeki tüm bilgilerin akademik kurallara ve etik ilkelere uygun olarak elde edildiğini ve sunulduğunu; ayrıca bu kuralların ve ilkelerin gerektirdiği şekilde, bu çalışmadan kaynaklanmayan bütün atıfları yaptığımı beyan ederim."

Emrullah SONUÇ

ÖZET

Doktora Tezi

BENZETİLMİŞ TAVLAMA ALGORİTMASININ GRAFİK İŞLEMCİ ÜNİTESİ KULLANILARAK PARALELLEŞTİRİLMESİ

Emrullah SONUÇ

Karabük Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı:
Yrd. Doç. Dr. Şafak BAYIR

Haziran 2017, 73 sayfa

Bilgisayar bilimlerinde, yapay zeka ve optimizasyon problemlerinde klasik çözüm yöntemlerinin yetersiz veya yavaş kalması sezgisel yöntemlerin ortaya çıkmasına neden olmuştur. Benzetilmiş Tavlama (BT) algoritması sezgisel algoritmalar arasında önemli bir yere sahiptir. Geçmişten günümüze birçok problem üzerinde uygulanmış ve uygulanmaya devam etmektedir. BT algoritmasının çözüme bir noktadan başlaması ve daha sonraki adımlarda yeni çözümü elde etmek için önceki çözüme ihtiyaç duyması çözüm uzayının dar bir alanda aranmasına neden olmaktadır. Çözüm uzayını genişletmek için algoritmayı birden fazla çalıştırmak veya algoritmanın iterasyon sayısını artırmak gerekmektedir. Ancak bu durum algoritmanın çalışma süresini artıracağından ortaya zaman probleminin çıkmasına neden olmaktadır.

Grafik donanımının performansındaki hızlı artış ve bu donanımın üzerindeki programlanabilirliğin gelişmesi, grafik donanımını hesaplama gerektiren çeşitli uygulama alanlarında ilgi çekici bir platform haline getirmiştir. Günümüzde birçok araştırmacı ve geliştirici, genel-amaçlı hesaplama kullanımı için grafik donanımının gücünü kullanmayı tercih etmektedir. Son yıllarda GPGPU olarak bilinen bu araştırma çabalarına ilginin hızla arttığı görülmektedir.

Bu çalışmada BT algoritmasının GPU üzerinde paralel bir şekilde çalışması sağlanmıştır. Paralel algoritmanın daha etkin bir şekilde çalışmasını sağlayan çoklu-başlangıç tekniği ile problem üzerinde elde edilecek sonuçların kalitesini artırmak amaçlanmış, aynı zamanda BT algoritmasının çalışma süresi sorununa bir çözüm getirilmesi amaçlanmıştır. Geliştirilen yöntem, Karesel Atama Problemi, 0/1 Sırt Çantası Problemi ve Silah-Hedef Atama Problemi üzerinde test edilmiştir.

Anahtar Sözcükler : Benzetilmiş Tavlama (BT), paralel programlama, optimizasyon, GPGPU, CUDA.

Bilim Kodu : 924.1.014

ABSTRACT

Ph. D. Thesis

PARALLELIZATION OF SIMULATED ANNEALING ALGORITHM ON GRAPHICS PROCESSING UNIT

Emrullah SONUÇ

Karabük University
Graduate School of Natural and Applied Sciences
Department of Computer Engineering

Thesis Advisor:
Assist. Prof. Dr. Şafak BAYIR
June 2017, 73 pages

In computer science, heuristic methods were arised because of the weakness and slowness of classical solution methods in artificial intelligence and optimization problems. Simulated Annealing (SA) algorithm is one of the heuristic methods. SA algorithm has been applied to many optimization problems from past to present day and continues to be applied. Since SA algorithm starts with one point to solve a problem and requires a previous solution to start a new solution at next iteration, search space is in a limited range. For searching in a large search space, it is necessary to run the algorithm more than once or to increase the number of iterations of the algorithm. On the other hand, runtime of the algorithm will also increase and this will be a problem.

The rapid increase in the performance of graphics hardware, coupled with recent improvements in its programmability, have made graphics hardware a compelling

platform for computationally demanding tasks in a wide variety of application domains. Researchers and developers have become interested in harnessing this power for general-purpose computing. In recent years, interest in these research efforts, known as GPGPU has increased rapidly.

In this study, SA algorithm is run in parallel on the GPU. It is aimed to increase the quality of the results obtained by the multistart technique which enables the parallel method to work more efficiently. Also it is aimed to provide a solution to the problem of runtime of SA algorithm. The developed method has been tested on Quadratic Assignment Problem, 0/1 Knapsack Problem and Weapon-Target Assignment Problem.

Key Word : Simulated Annealing (SA), parallel programming, optimization, GPGPU, CUDA.

Science Code : 924.1.014

TEŞEKKÜR

Âlemlerin Rabbi olan Allah'a (cc) sonsuz hamdler olsun.

Bu tez çalışmasının planlanmasında, araştırılmasında, yürütülmesinde ve tamamlanmasında ilgi ve desteğini esirgemeyen danışman hocalarım Yrd. Doç. Dr. Şafak BAYIR'a ve Yrd. Doç. Dr. Baha ŞEN'e sonsuz teşekkürlerimi sunarım.

Tez izleme sürecinde değerlendirmeleri ile destek olan değerli hocalarım Prof. Dr. Mehmet AKBABA'ya ve Yrd. Doç. Dr. Özden İŞBİLİR'e teşekkür ederim.

Sevgili eşime ve aileme manevi desteklerini her zaman hissettiğim için, sevgili kızıma ve oğluma varlıklarıyla hayatıma neşe kattıkları için tüm kalbimle teşekkür ederim.

Bu çalışma Karabük Üniversitesi tarafından Bilimsel Araştırma Projesi olarak KBU-BAP-15/2-DR-027 proje numarasıyla desteklenmiştir.

İÇİNDEKİLER

Sayfa

Sayfa

Sayfa

ŞEKİLLER DİZİNİ

Sayfa

ÇİZELGELER DİZİNİ

Sayfa

SİMGELER VE KISALTMALAR DİZİNİ

KISALTMALAR

API : Application Programming Interface (Uygulama Programlama Arayüzü)
BT : Benzetilmiş Tavlama
CAD : Computer Aided Design (Bilgisayar Destekli Çizim)
CPU : Central Processing Unit (Merkezi İşlemci Ünitesi)
CUDA : Compute Unified Device Architecture (Birleşik Hesaplama Aygıt Mimarisi)
FPGA : Field Programmable Gate Array (Alanda Programlamlanabilir Kapı Dizileri)
GPGPU : General-Purpose Computing on Graphics Processing Units (Grafik İşlemcisinde Genel Amaçlı Hesaplama)
GPU : Graphics Processing Unit (Grafik İşlemci Ünitesi)
KP : Knapsack Problem (Sırt Çantası Problemi)
MPI : Message Passing Interface (Mesaj Geçirme Arayüzü)
NP : Non-deterministic Polynomial-time (Deterministik-olmayan Polinominal-zaman)
OpenCL : Open Computing Language (Açık Hesaplama Dili)
OpenMP : Open Multi-Processing (Açık Çoklu-İşleme)
PSA : Parallel Simulated Annealing (Paralel Benzetilmiş Tavlama)
RPD : Relative Percentage Deviation (Bağıl Yüzde Sapma)
QAP : Quadratic Assignment Problem (Karesel Atama Problemi)
QAPLIB : Quadratic Assignment Problem Library (Karesel Atama Problemi Kütüphanesi)
SA : Simulated Annealing (Benzetilmiş Tavlama)
SD : Standard Deviation (Standart Sapma)
SIMD : Single Instruction Multiple Data (Tek Komut Çoklu Veri)
SM : Streaming Multiprocessor (Eşzamanlı Çoklu İşlemci)
TSP : Traveling Salesman Problem (Gezgin Satıcı Problemi)

VLSI : Very Large Scale Integration (Çok Büyük Boyutta İntegrasyon)

WTA : Weapon-Target Assignment Problem (Silah-Hedef Atama Problemi)

BÖLÜM 1

GİRİŞ

Bu bölümde ilk olarak tezin arkaplanı ve motivasyon kaynağından bahsedilecek, devamında problemin tanımı yapılarak tezin amaçları detaylandırılacaktır. Önerilen çözüm sunulduktan sonra ana katkılar listelenerek tezin ana hatları ile bölüm sonlandırılacaktır.

1.1. ARKAPLAN VE MOTİVASYON

Bilişim sektörünün geldiği nokta ve buna bağlı olarak ihtiyaçların her geçen gün hızla artması birçok problemi de beraberinde getirmektedir. Bilgi teknolojileri sektörü de bu konuda ilk sırada yer alan sektörlerin başında gelmektedir. Günümüzde gerek kurumsal, gerekse akademik çalışmalarda kullanılan veri boyutlarının artması performans konusunu ilk sıraya yerleştirmiştir. Artık geliştirilen bir sistemin kararlı ve hatasız çalışmasının yanında en kısa sürede en iyi sonucu elde etmek de isteklerin başında gelmektedir. Hız konusu; üretim sektörü, mühendislik hesapları, askeri simülasyon projeleri ve hava tahmin hesaplamaları için hayati öneme sahiptir. Paralel hesaplamanın kullanıldığı birçok alan vardır. Bunlara örnek olarak;

- Hava Tahmini,
- Fizik-Nükleer Hesaplamalar,
- Uzay Kara-deliklerinin Modellenmesi,
- Askeri Simülasyonlar,
- Kimya-Moleküler,
- Elektrik-Elektronik Devre Tasarımı,
- Veri Madenciliği,
- Kombinatoryal Optimizasyon

alanları verilebilir [1].

Paralel hesaplamada yapılacak bir iş, birden fazla iş bölümü tarafından gerçekleştirilebildiği gibi birden çok iş bölümlerine ayrılarak da gerçekleştirilebilir. Her iş bölümünde bulunan sonuçlar bir yerde toplanır ve bu şekilde işlerin daha kısa zamanda, daha hızlı ve verimli bir şekilde yapılmasına imkan sağlanır [2].

Paralel hesaplamanın genel kullanım amaçları şunlardır [3]:

- Zaman ve/veya Para Tasarrufu:
    - Teorik olarak, bir işe daha fazla kaynağın atanması, yapılacak işlemin çalışma süresini kısaltır.
    - Paralel bilgisayarlar maliyet açısından hesaplı bileşenler kullanılarak tasarlanabilir.
- Kompleks Problemlerin Çözümü:
    - Birçok problem büyük ve/veya karmaşıktır; bunları tek bir bilgisayarda özellikle de sınırlı belleğe sahip bir bilgisayarda çözmeye çalışmak pratik bir yol değildir ve imkânsızdır.
- Aynı Anda Kullanım (Concurrency) Sağlama:
    - Tek bir hesaplama kaynağı, aynı anda yalnızca tek bir iş yapabilir. Birden fazla hesaplama kaynağı aynı anda birçok işi yapabilir.
- Yerel Olmayan Kaynakların Avantajlarından Yararlanma:
    - Yerel hesaplama kaynakları yetersiz olan durumlarda geniş alan ağı veya internet üzerinde bulunan hesaplama kaynaklarından yararlanılabilir.
- Paralel Mimariyi Kullanarak İyileştirme Elde Etme:
    - Dizüstü bilgisayarlar dahil olmak üzere modern bilgisayarlar mimari olarak birden fazla işlemci/çekirdeğe sahiptirler.
    - Paralel yazılımlar çoklu iş parçacıklarının veya çekirdeklerin kullanıldığı paralel donanımlar için tasarlanır.
    - Çoğu durumda, seri çalışan bilgisayar programları potansiyel bilgi işlem gücünü yeterince kullanmadıkları için israf olarak nitelenmektedir.

1.2. PROBLEMİN TANIMLANMASI

Bilgisayar bilimlerinde, yapay zeka ve optimizasyon problemlerinde klasik çözüm yöntemlerinin yetersiz veya yavaş kalması sezgisel yöntemlerin ortaya çıkmasına neden olmuştur. Sezgisel ifadesi bir problem için olası çözümlerin bir tanesini bulan algoritmalar için kullanılır ve bu algoritmalar en iyi çözümü elde etme garantisi sunmazlar. Sezgisel algoritmaların asıl amacı, en iyi çözüme ya da en iyi çözüme yakın bir çözüme kolay ve hızlı bir şekilde erişmektir. Bu algoritmalar en iyi çözümü bulsa bile probleme ait en iyi çözümün bulunan çözüm olduğu çoğu zaman kanıtlanamadığı için sezgisel olarak adlandırılırlar.

Sezgisel algoritmalar, bir problemi çözmek için rastgele bir çözüm üreterek yöntemi başlatırlar. Daha sonra her bir adımda algoritmaya has tekniklerle komşu çözüm ararlar. Bulunan komşu çözümü yine kendine has tekniklerle kabul veya ret ederler. Bu amaç doğrultusunda algoritmaya ait sonlandırma kriterleri sağlanana kadar en iyi çözümü ararlar. Algoritma işlemini tamamlandığında ise bulunan en iyi çözümü çözüm olarak sunar. Bu bağlamda bakıldığında, sezgisel algoritmalar genel olarak bir sonraki çözümü bulmak için bir önceki çözüme ihtiyaç duyarlar.

Benzetilmiş Tavlama (Simulated Annealing, SA) algoritması sezgisel algoritmalardan bir tanesidir. SA diğer sezgisel algoritmalarda olduğu gibi en iyi çözüme yakın bir çözümü makul bir sürede elde etmeyi amaçlamaktadır. SA algoritmasının en önemli avantajlarından bir tanesi uygulanacak probleme kolay bir şekilde adapte edilebilmesidir. Bunun yanı sıra SA algoritmasının uygulanan problemler üzerinde etkili sonuçlar elde ettiği literatürdeki çalışmalarla ortaya konmuştur. SA algoritması ile ilgili detaylı ve geniş bilgi Bölüm 3'de sunulacaktır.

SA algoritmasının çözüme bir noktadan başlaması ve daha sonraki adımlarda yeni çözümü elde etmek için önceki çözüme ihtiyaç duyması çözüm uzayının dar bir alanda aranmasına neden olmaktadır. Çözüm uzayını genişletmek için algoritmayı birden fazla çalıştırmak veya algoritmanın iterasyon sayısını artırmak daha iyi çözümler elde etmeye imkan tanır. Ancak bu durum algoritmanın çalışma süresini artıracağından ortaya zaman probleminin çıkmasına neden olur. Birden çok çözüm

noktasıyla algoritmayı başlatmak ve zaman probleminin üstesinden gelmek için algoritmanın paralelleştirilmesi bir çözüm olarak sunulabilir.

1.3. ÖNERİLEN ÇÖZÜM

Grafik donanımının performansındaki hızlı artış ve bu donanımın üzerindeki programlanabilirliğin gelişmesi, grafik donanımını hesaplama gerektiren çeşitli uygulama alanlarında ilgi çekici bir platform haline getirmiştir. Grafik donanımları günümüzün en güçlü hesaplama platformlarından birisidir. Grafik İşlemci Ünitesi (Graphics Processing Unit, GPU), çevre birimlerini kendi başına modern, güçlü ve programlanabilir işlemcilere taşımıştır. Günümüzde birçok araştırmacı ve geliştirici, genel-amaçlı hesaplama kullanımı için grafik donanımının gücünü kullanmayı tercih etmektedir. Son yıllarda Grafik İşlemcisinde Genel Amaçlı Hesaplama'nın (General-Purpose Computing on Graphics Processing Units, GPGPU) ortaya çıkmasıyla bu araştırma çabalarına ilginin hızla arttığı görülmektedir [4].

Günümüzde GPU'ların çok sayıda iş parçacığına (thread) sahip olması paralel hesaplama alanında kullanım açısından büyük bir fayda sağlamaktadır. Merkezi İşlemci Ünitesi'ne (Central Processing Unit, CPU) nazaran daha fazla iş parçacığının olması, paralel hesaplama çalışmalarını CPU tarafından GPU tarafına yönlendirmiştir. Çalışmada kullanılan SA algoritması, GPU üzerinde paralelleştirilerek birkaç farklı probleme uygulanmıştır. Bununla birlikte Bölüm 1.2'de bahsedilen problemlere çözüm üretilmesi amaçlanmıştır. Bu aşamada sonuçlar, seri algoritma ile yapılan çalışmalar ile elde edilen sonuçlar ile karşılaştırılarak sonuçların kalitesi ve çalışma sürelerinin sunulması hedeflenmiştir. Bu bağlamda tez çalışmasında elde edilen katkıların sunulması amaçlanmıştır.

1.4. ORGANİZASYON VE YOL HARİTASI

Bu tez 6 bölümden oluşmakta ve her bir bölümde hangi konulara değinildiği aşağıda listelenmiştir.

Bölüm 1: Giriş bölümünde ilk olarak tezin arkaplanı ve motivasyonuna odaklanılmış, devamında problem tanımlanarak bu bağlamda probleme çözüm olarak sunulan tezin amaçlarından bahsedilmiştir.

Bölüm 2: Bu bölümde literatürde SA algoritmasının paralelleştirilmesiyle ilgili yapılan çalışmalar hakkında bilgiler verilmiştir.

Bölüm 3: Bu bölümde SA algoritması hakkında bilgiler verilmiştir. Algoritmanın çalışma prensibi anlatılmış, diğer algoritmalarla performans karşılaştırılması yapılmış, avantajları ve dezavantajları kapsamında algoritma hakkında değerlendirmeler sunulmuştur.

Bölüm 4: Bu bölümde SA algoritmasının paralelleştirildiği platform olan GPU hakkında bilgi verilmiştir. GPU üzerinde programlama platformu olarak kullanılan Birleşik Hesaplama Aygıt Mimarisi (Compute Unified Device Architecture, CUDA) platformunun programlama modeli ve bellek hiyerarşisi detaylı olarak sunulmuştur.

Bölüm 5: Bu bölümde deneysel çalışmalardan bahsedilmiştir. Paralel çalışan SA algoritmasının uygulandığı poblemler üzerinde elde edilen sonuçlar sunulmuş ve bu sonuçlar kapsamında değerlendirmelerde bulunulmuştur. Bu çalışmalar sonucu elde edilen nitel sonuçlar çizelgeler halinde verilmiştir.

Bölüm 6: Tezin bu son bölümünde, yapılan çalışmaların önemi, literatüre katkısı ve genel değerlendirmeler sunulmuştur.

BÖLÜM 2

LİTERATÜR ÇALIŞMALARINA GENEL BAKIŞ

Literatürde SA ile ilgili yapılan paralelleştirme çalışmalarına bakıldığında GPU teknolojisinin gelişmesiyle birlikte eğilim CPU'dan GPU'ya doğru yönelmiştir. Yapılan birçok çalışmada, CPU ve GPU arasında performans değerlendirmeleri yapılmış, kriter olarak süre ve sonuçların doğruluğu değerlendirilerek analizlerde bulunulmuştur. Yine kullanılan iş parçacığı sayısı da bir başka kriter olarak ortaya çıkmaktadır. Algoritmanın CPU üzerinde çalışırken işletim sistemi tarafından iş parçacıklarının meşgul edilmesi gibi sebepler CPU'yu bu konuda dezavantajlı hale getirmektedir. Diğer yandan GPU teknolojisi kullanılarak yapılan paralelleştirmelerde CPU'dan da faydalanılması söz konusu olduğundan problemin çözümüne yönelik farklı yaklaşımlar sergilenebilmektedir. Ayrıca GPU teknolojisinin gelişmesinden kaynaklı ortaya konulan yeni çalışmalar ve yaklaşımlar ile sonuçların her geçen gün daha da iyileştirildiği görülmektedir.

Bu bölümde literatürde konu ile ilgili yapılmış çalışmalar sunulmuştur. CPU üzerinde yapılan çalışmalar son yıllarda hızlı bir şekilde azalma eğilimine girdiği için liteatür araştırmasındaki ağırlık GPU üzerindeki çalışmalara verilmiştir. Sırasıyla öncelikle CPU üzerinde yapılan paralelleştirme çalışmalarından bahsedilmiş daha sonra GPU üzerinde yapılan paralelleştirme çalışmaları sunulmuştur.

Czech vd. [5] zaman kısıtlı araç yönlendirme problemi üzerinde çalışmışlar, problemi paralel SA algoritması ile çözmeyi amaçlamışlardır. CPU üzerinde yapılan paralelleştirme için Mesaj Geçirme Arayüzü (Message Passing Interface, MPI) kütüphanesinden yararlanmışlar, iş parçacıkları arasında iletişim sağlayarak en iyi çözümlerin bir sonraki adımda kullanılmasını amaçlamışlardır. Probleme ait literatürde bulunan örnekler üzerinde yöntemlerini uygulamışlar ve bu örnekler için başarılı sonuçlar elde ettiklerini belirtmişlerdir. Çalışmanın sonunda paralel SA

algoritmasının iki kriterli optimizasyon problemlerine uygulanabileceğini vurgulamışlardır.

Onbaşıoğlu ve Özdamar [6] global optimizasyon problemlerini çözmek için SA algoritmasının CPU üzerinde paralel çalışan farklı versiyonlarını geliştirmişlerdir. Bu yaklaşımlar iş parçacıkları arasındaki iletişim farklılıklardan meydana gelmektedir. Herhangi bir iletişim sağlanmadığı asenkron yaklaşımlar, çözümlerin ara sıra ya da belli durumlara göre iletildiği senkron yaklaşımlar sunulmuştur. Geliştirilen yöntemleri büyük ölçekli test fonksiyonları üzerinde denemişler, paralel SA algoritmasının seri SA algoritmasına göre daha iyi performans sergilediğini belirtmişlerdir. SA algoritmasında sıcaklık parametresinin ana (master) iş parçacığı tarafından kontrol edildiği ve tek bir Markov zincirinin tüm işçi (worker) iş parçacıkları tarafından takip edildiği Değiştirilmiş Yüksek Bağlı Eşzamanlı Yaklaşım (Modified Highly Coupled Synchronous Approach) adı verilen yöntemin başarılı olduğunu sonuç olarak sunmuşlardır.

Debudaj-Grabysz ve Rabenseifner [7] yaptıkları çalışmada SA algoritmasını paralelleştirirken sadece MPI kütüphanesini kullanmamışlar daha iyi performans elde etmek için MPI ve Açık Çoklu-İşleme (Open Multi-Processing, OpenMP) teknolojilerinin ikisini kullanan bir yöntem sunmuşlardır. Geliştirilen bu yeni hibrit yöntem, paralel SA algoritmasının diğer versiyonlarıyla karşılaştırılmıştır. Belirli bir zaman dilimi için verilen mevcut iş parçacıklarının çözüm uzayında optimal sonucu aradığı düşünüldüğünde geliştirilen bu hibrit yöntemin, diğer paralel yöntemlere kıyasla daha iyi bir değer ürettiği çalışmada vurgulanmıştır. Yöntemin avantajlarını göstermek için genel olarak kabul edilen bir karşılaştırma problemi olan araç yönlendirme problemi üzerinde çalışmışlar ve hibrit yöntemin sağladığı faydaları belirtmişlerdir.

Chang vd. [8] yüksek boyutlu uzaktan algılama görüntüleri için paralel SA bant seçimi algoritması (PSABS) geliştirmişlerdir. Geliştirilen yaklaşım, başlangıçta yüksek korelasyonlu hiperspektral bantları, dalga boyları açısından orijinal düzene bakılmaksızın daha küçük bir modül grubuna gruplamak üzere tasarlanmış SA bant seçimi (SABS) şemasına dayanmaktadır. SABS, SA algoritmasına dayanan

korelasyonlu hiperspektral bantların kümelerini seçer ve boyutsallığı azaltmak için farklı sınıfların doğal ayrılabilirliğini kullanır. Çalışmada önerilen PSABS ise, paralel hesaplama tekniğini kullanarak hesaplama performansını artırmak amacıyla geliştirilmiştir. Geliştirilen yöntem ile birden fazla Markov zincirinin (MMC) aynı anda izlenmesinin ve her paralel düğümde bir dizi SABS modülünün oluşturulmasının sağlanması amaçlanmıştır. CPU üzerinde paralelleştirilen yöntem için MPI ve OpenMP kullanılmıştır. Örnek problem setleri üzerinde test edilen PSABS algoritmasında, MMC tekniklerinin, hesaplama performansını belirgin bir şekilde artırdığı ve orijinal SABS algoritmasına kıyasla daha iyi sonuçların elde edildiği ortaya konulmuştur.

Bajrami vd. [9] yaptıkları çalışmada SA algoritmasını CPU ve GPU üzerinde çalıştırarak performans değerlendirmesi yapmışlardır. Performans kriteri olarak algoritmanın çalışma süresini referans almışlardır. Başlangıç nokta sayısı ve iş parçacığı sayısı olarak iki ayrı senaryo uygulamışlardır. Yapılan testlerin sonuçlarında GPU üzerinde CUDA kullanılarak çalıştırılan programın daha performanslı olduğu görülmüştür. Ayrıca CPU'nun işletim sistemi tarafından da meşgul edilmesinden ötürü çalışma yürütülürken iş parçacıklarının değiştiği gözlemlenmiştir.

Han vd. [10] SA algoritmasını GPU üzerinde paralelleştirerek tümleşik devre yerleşim planı (Integrated Circuit Floorplanning) için örnek bir uygulama yapmışlardır. Çalışmada, SA algoritmasının Tek İşlem Çoklu Veri (Single Instruction Multiple Data, SIMD) tarzı yapıya GPU üzerinde uyumlu olduğu belirtilmiştir. Verilen bir yerleşim planı için eş zamanlı taşımalar da değerlendirilerek yerleşim planı çözüm uzayının araştırılmasında farklı yaklaşımlar önermişler ve GPU üzerinde algoritmanın optimizasyonu için birkaç teknik göstermişlerdir. Yapılan karşılaştırmalar neticesinde uygulanan tekniklerin seri algoritmaya nazaran 6 kattan 160 kata kadar daha hızlı olduğu testlerin sonucu olarak sunulmuştur.

Fobel vd. [11] Alanda Programlanabilir Kapı Dizileri (Field Programmable Gate Array, FPGA) yerleşimi probleminin çözümü için SA algoritmasının geniş bir kullanım alanının olduğundan bahsetmişlerdir. GPU üzerinde paralelleştirme

yaparak, algoritmayı Star+ adı verilen bir tel-uzunluk modeli ile uygulamışlardır. Sonuçlara bakıldığında karşılaştırmalı değerlendirmelerde (benchmark) 5 kat ile 82 kat arasında performans artışı olduğu görülmüştür.

Li ve Liu [12] çalışmalarında protein yapısı tahmini için SA algoritmasını kullanmışlardır. Daha doğru ve etkili bir sonuç için algoritmayı GPU üzerinde çalıştırmışlardır. Aradaki farkı ortaya koymak adına öncelikle yöntemlerini CPU'da çalıştırmışlar ve çalışmalarının sonucunda GPU'da elde edilen değerlerin daha uygun değerler olduğu görülmüştür.

Fabry-Asztalos vd. [13] çalışmalarında, moleküler mesafe problemi için SA algoritmasını kullanmışlar ve GPU üzerinde CUDA ile paralelleştirmişlerdir. Çıkan sonuçlarda Karesel Ortalama Hata'yı (Root Mean Square Error, RMSE) kriter alarak doğru sonuçları elde etmek için SA algoritmasından yararlanmışlardır. Kullandıkları graf yapılarına göre performansta değişiklik gözlemlenmiştir. Problemin çözümü için hesaplama yükü çok olmasına karşılık SA algoritmasının çözüm doğruluğu açısından fayda sağladığı tespit edilmiştir.

Fobel vd. [14] yaptıkları çalışmada, FPGA mimarilerinin katlanarak büyüdüğünü belirtmişler ve buna bağlı olarak tasarımın güçlüğüne dikkat çekmişler ve GPU üzerinde CUDA ile bir algoritma geliştirmişlerdir. FPGA yerleşimi amacıyla geliştirilen algoritma VPR adlı uygulama karşılaştırıldığında, geliştirilen algoritmanın 19 kat daha hızlı olduğu sonucuna ulaşmışlardır.

Kataoka vd. [15] VLSI (Very Large Scale Integration) için CAD (Computer-Aided Design) algoritmasının kullanımı konusunda GPU üzerinde yapılan çalışmaların yeterli olmadığını belirtmişler ve bu amaçla bir yöntem geliştirmişlerdir. SA algoritmasını paralelleştirerek VLSI yerleşiminde önemli bir problem olan dikdörtgen paketlemeyi (rectangle packing) temel problem alarak bu bağlamda çalışmalarını yürütmüşlerdir. Karşılaştırmalı testlere bakıldığında etkili sonuçlar elde edildiği görülmüştür.

Ferreiro vd. [16] yaptıkları çalışmada son dönemde geliştirilen GPU'lar için optimize edilmiş bir SA algoritması önermişlerdir. Programlamayı NVIDIA GPU'larına yönelik birleşik cihaz mimarisi CUDA aracı kullanarak yürütmüşlerdir. Algoritmanın versiyonlarını GPU'ya adapte ederek analizlerde bulunmuşlardır. Seri bir SA yaklaşımı ile işe başlayarak daha sonra asenkron paralel versiyonunu uygulamaya koymuşlar daha sonra özelleştirilmiş ve senkronize çalışan bir versiyon geliştirmişlerdir. Uygulamanın performans özelliklerini göstermek için geniş ölçekli bir değerlendirme ölçümleri yapmışlardır. Yapılan testlerde, normalleştirilmiş Schwefel fonksiyonunun minimize edilerek uygulanan örneğinde; seri, asenkron ve senkron versiyonların davranışları karşılaştırılmış, senkron versiyonun yakınsama, doğruluk ve hesaplama maliyeti açısından daha avantajlı olduğu görülmüştür.

Stivala vd. [17] yaptıkları çalışmada SA algoritmasını kullanarak, hız ve doğruluk bakımından yaygın olarak kullanılan bazı yöntemlerle karşılaştırma yapmak amacıyla, tablo tabanlı protein yapısı ve altyapı araması gerçekleştirmişlerdir. Çalışmada, protein yapısı araması konusunda GPU üzerinde gerçekleştirilen ilk çalışmayı yaptıklarını vurgulamışlar ve CPU uygulamasına göre 34 kat hız elde etmişlerdir.

Literatürde SA algoritması ile ilgili parallelleştirme çalışmalarına bakıldığında GPU üzerindeki çalışmaların sayısının yetersiz olduğu görülmektedir. GPU üzerindeki programlamanın ve organizasyonun CPU'ya nazaran daha zor olması bunun nedenleri arasında gösterilebilir. Yapılan çalışmaların büyük bir kısmı değerlendirme açısından yalnızca hızlanmayı kriter olarak almıştır. Hem süre hem de sonuçların kalitesinin değerlendirildiği çalışmaların kısıtlı ve yetersiz olduğu görülmektedir. Yapılan bu çalışma ile SA algoritmasının hem süre hem de uygulanacak olan problemin sonuçları açısından performanslı bir yöntem olması amaçlanmıştır.

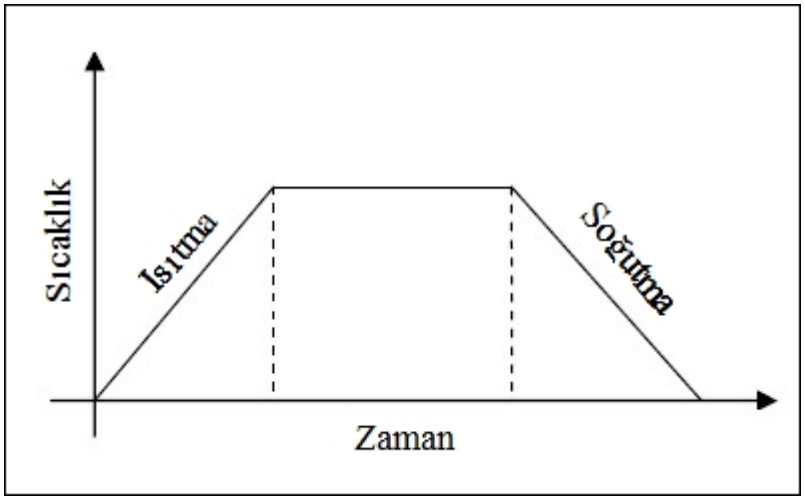
BÖLÜM 3

BENZETİLMİŞ TAVLAMA ALGORİTMASI

Bu bölümde SA algoritmasının tanımından ve tarihçesinden kısaca bahsedilerek çalışma prensibi anlatılacaktır. SA algoritmasına ait parametrelerin tanımları yapılacak, algoritmanın avantaj ve dezavantajlarından bahsedilecektir. Gezgin Satıcı Probleminin (Traveling Salesman Problem, TSP) SA algoritması kullanılarak çözümüne yönelik örnek bir uygulama ile bölüm sonlandırılacaktır.

3.1. GİRİŞ VE ARKAPLAN

Benzetilmiş Tavlama (Simulated Annealing) algoritması ismini demir tavlamak anlamına gelen tavlama (annealing) kelimesinden almıştır. Tavlama, sistemin iç yapısındaki fiziksel veya kimyasal özelliklerini değiştirmek için uygulanan ısıtma ve soğutma işlemi olarak tanımlanır. Tavlama işleminin temel amacı, sistemin sıcaklığını arttırmak ve daha sonra yavaş yavaş soğutmak suretiyle mevcut veya rastgele oluşturulmuş durumlardan sistemin amaçlanan yapıya kavuşmasını sağlamaktır. Dolayısıyla, tavlama işlemi, fiziksel bir sistemin ısıtılması ve soğutulması süreçleri olarak iki adımdan oluşmaktadır. Öncelikle, fiziksel sistemin enerjisini yükseltmek için ısıtılma işlemi gerçekleştirilir, böylece atomların sistem içerisinde serbestçe dağılması sağlanarak dengesiz bir yapı elde edilir. Daha sonra sistemin istenilen yapısını elde etmek amacıyla sistem yavaş bir şekilde soğutulur. Soğutulma işlemi sürecinde atomların farklı enerji seviyelerine geçmesi sağlanarak sistemin yapısı üzerinde değişiklik sağlanır. Bu süreç, sistemin daha düzgün ve kararlı bir fiziksel yapıya dönüşmesine imkan tanır. Soğutma işlemi, fiziksel sistemin minimum enerji seviyesine ulaşana kadar devam eder. Bu enerji seviyesi ancak başlangıç veya maksimum sıcaklık yeterince yüksek olduğunda ve fiziksel sistem çok yavaş soğutulduğunda elde edilir. Katı bir cisim üzerinde gerçekleştirilen ısıl işlem süreci Şekil 3.1'de gösterilmiştir.



Şekil 3.1. Katı bir cisim için ısıl işlem süreci.

Genel olarak bir sezgisel algoritma, rastgele bir çözüm üretip arama uzayı alanındaki komşu çözümleri sistematik olarak araştırarak en iyi çözümü elde etmeyi amaçlar. SA algoritması, yukarıda bahsedilen tavlama işleminden esinlenerek geliştirilen yerel bir arama algoritmasıdır.

Metropolis vd. [18] 1953 yılında SA algoritmasının temellerini atmışlardır. Çalışmalarında, bir sistemin sabit sıcaklıkta termal denge düzeyine ulaşması için farklı enerji seviyeleri arasındaki geçişini incelemişlerdir. SA algoritmasının anahtar parçası olarak kabul edilen ve enerji seviyeleri arasındaki geçiş ihtimali için kullanılan Boltzmann dağılımını sunmuşlardır. Boltzmann dağılımına göre, Kirkpatrick vd. [19] 1983 yılında, Cerny vd. [20] 1985 yılında optimizasyon problemlerini çözmek için birbirlerinden bağımsız olarak SA tekniğini arama algoritması olarak kullanmışlardır.

3.2. METROPOLİS ALGORİTMASI VE BOLTZMANN DAĞILIMI

Metropolis algoritması ve Boltzmann dağılımı SA algoritmasının sezgisel bir algoritma olarak kullanılmasında önemli yer tutarlar. Metropolis vd. [18] yaptıkları çalışmada bir sistemin sabit sıcaklıktaki termal dengeye ulaşması için farklı enerji seviyeleri arasındaki değişimi incelemişlerdir. Bunun üzerine bir sistemin farklı enerji seviyeleri arasındaki termal denge geçişini referans alarak bir algoritma geliştirmişlerdir. Bu çalışmaların sonucunda Boltzmann dağılımı ortaya çıkmıştır.

SA algoritmasında arama uzayında gerçekleştirilen rastgele hareketler Boltzmann dağılımına bağlı olarak gerçekleştirilir. Termal denge durumunda, bir sistemin *i* durumundaki konfigürasyonuna ait E_i enerjisi ile olasılığı (P_i) Boltzmann dağılımı ile aşağıdaki gibi gösterilir:

$$P_i = \frac{exp\ (-\frac{E_i}{kT})}{S} \tag{3.1}$$

S olası tüm konfigürasyonları içermektedir:

$$S = \sum_{i=1}^{N} exp\ (-E_i/kT) \tag{3.2}$$

Burada *T* tavlama sürecinde sistemin o anki sıcaklığı, *k* Boltzmann sabiti ve *N* ise toplam konfigürasyon sayısını ifade etmektedir ve olasılıklar toplamı aşağıdaki gibidir:

$$\sum_{i=1}^{N} p_i = 1 \tag{3.3}$$

3.3. ALGORİTMANIN ÇALIŞMA PRENSİBİ

Optimizasyon problemi çözümünde SA algoritmasının kullanılması, katı bir cismin tavlama sürecindeki düşük enerji seviyesini bulması arasındaki benzerliğe bağlı olarak geliştirilmiştir. Tavlama işleminde amaç, optimizasyon işleminde hedef fonksiyon değerini tavlama sürecindeki enerji seviyesi için yapıldığı gibi en aza indirgemektir. Fiziksel sistemin atomlarının konfigürasyonu (dağılımı), optimizasyon problemine ait değişkenlere karşılık gelir. Sistemin enerji durumu, optimizasyon probleminin olası çözümleri ile aynıdır. Fiziksel sistemin mevcut ve aday (alternatif) durumları, optimizasyon probleminde mevcut ve aday (alternatif) çözümlerle çakışmaktadır. Tavlama işlemindeki sıcaklık, hedef çözüme yakınsama seviyesini belirlemek için optimizasyon işleminde bir kontrol ve sonlandırma parametresi olarak kullanılır. Son olarak, yerel ve global minimum enerji durumları sırasıyla,

optimizasyon problemindeki yerel ve global optimum çözümlere karşılık gelir. Çizelge 3.1 tavlama süreci ile SA algoritması arasındaki ilişkiyi göstermektedir [21].

Çizelge 3.1. Tavlama süreci ve SA algoritması arasındaki ilişki.

Tavlama Süreci	**SA Algoritması**
Enerji Durumu	Olası Çözüm
Enerji Seviyesi	Amaç Fonksiyonu
Atomlarının Konfigürasyonu	Problem Değişkenlerinin Kümesi
Mevcut Durum	Mevcut Çözüm
Aday Durum	Aday Çözüm
Yerel MinimumEnerji	Yerel Minimum Çözüm
Global MinimumEnerji	Global Minimum Çözüm

SA algoritmasının sözde kodu (pseudocode) aşağıdaki gibidir:

```
s ← s0; e ← E(s)                    // İlk durum, enerji
sbest ← s; ebest ← e                // Başlangıcı en iyi çözüm ata
k ← 0                               // İterasyon sayısı
while k < kmax and e > emax         // İterasyon bitene devam et
        T ← temperature(k/kmax)     // Sıcaklığı azalt
        snew ← neighbor(s)          // Yeni bir komşu çözüm seç
        enew ← E(snew)              // Enerjiyi hesapla
        if P(e, enew, T) > random() then   // Yeni durum kabul edilebilir mi?
            s ← snew; e ← enew      // Evet ise mevcut durumu değiştir
        if e < ebest then           // Enerji daha düşükse en iyi çözüm yap
            sbest ← snew; ebest ← enew   // Mevcut durumu en iyisiyle değiştir
        k ← k + 1                   // İterasyon sayısını artır
return sbest                        // En iyi sonuçla bitir
```

Sözde kodda görüldüğü üzere bir iterasyon sayısının varlığı bir sonraki adım için mevcut adımın tamamlanmasını gerektirmektedir. Seri algoritmada her adımda tek bir çözüm bulunur ve mevcut ile karşılaştırılır. Günümüzde kullanılan paralelleştirme yöntemleri kullanılarak her adımın iş parçacıklarına dağıtılmasıyla birden fazla

çözüm üretilip bunlar için en uygun olanı seçebilmek mümkündür. Bu sayede seri algoritma için gereken süre içerisinde paralel bir yöntemde birden fazla çözüm üretilerek daha iyi sonuçların bulunabilme ihtimali de artmış olmaktadır.

SA algoritması için kullanılan parametrelerin doğru belirlenebilmesi problemin çözümünde önemli rol oynamaktadır. SA algoritmasında genel olarak 4 ana parametre vardır. Bunlar:

- Başlangıç Sıcaklığı,
- Soğutma Katsayısı,
- Hedef Sıcaklık ve
- İterasyon Sayısıdır.

3.3.1. Başlangıç Sıcaklığı

SA algoritmasında başlangıç sıcaklığının belirlenmesi önemli bir olgudur. Başlangıç sıcaklığının gereğinden fazla bir değere çıkartılması gereksiz işlem sürelerine neden olurken; düşük sıcaklıklarda başlanması ise algoritmanın yerel minimumlarda takılma riskini arttırmaktadır. O yüzden eğer başlangıç sıcaklığının değeri hakkında kesin bir sınır belirlenemiyorsa en iyi çözüme ulaşabilmek adına mümkün olduğunca yüksek değerlerin kullanılması daha uygun olacaktır.

3.3.2. Soğutma Katsayısı

Soğutma katsayısı α ile temsil edilmektedir. Genellikle 0 ile 1 arasında herhangi bir değer alarak mevcut sıcaklık ile çarpılarak düşürülür. Sıcaklığın düşürüldüğü farklı yöntemler de mevcuttur. Soğutma katsayısı 0'a yaklaştıkça sistem daha hızlı soğumakta, 1'e yaklaştıkça ise daha yavaş soğumaktadır.

Sıcaklık azaltma fonksiyonu farklı biçimlerde kurulabilir:

- Aritmetik ($T_k = T_k - 1 - C$)
- Geometrik ($T_k = T_k - 1 * \alpha$)

- Ters fonksiyon ($T_k = C / (1 + k)$)
- Logaritmik ($T_k = C / (Log(1 + k))$)

Burada T_k azaltmadan sonraki sıcaklığı temsil etmektedir. C başlangıç sıcaklığından küçük bir değer ve α da 0 ile 1 arasında soğutma katsayısı olarak ifade edilen bir değerdir. Yapılan çalışmalarda α değeri genellikle 0.8 ile 0.99 arasında bir değer almaktadır.

3.3.3. Hedef Sıcaklık

Hedef sıcaklık değeri durdurma kuralı olarak kullanılan değerdir. Algoritma başlangıç sıcaklığı ile çözüm adımlarına başlayarak her adımda belirlenen soğutma katsayısı ile sıcaklığı düşürür. SA algoritması belirlenen bir hedef sıcaklığa ulaşana kadar işlemlerine devam eder. Hedef sıcaklığa erişilince yöntem durdurulur. Bu kuralla beraber belli bir döngü sonucunda herhangi bir iyileştirme olmuyorsa, gereksiz işlemleri bitirmek adına yöntem durdurulabilir. Durdurma kriteri için kullanılabilecek bir diğer yöntem ise zamandır. Yöntem zaman yönünden kısıtlanıp belli bir zaman aralığı içinde bulunan en iyi sonuca odaklanabilir.

3.3.4. İterasyon Sayısı

İterasyon sayısı, SA algoritmasının birden fazla çalıştırılmasını ifade etmektedir. Her bir iterasyonda SA belirlenen parametrelerle tekrar adımlarına başlar. İlk iterasyondan sonraki her iterasyon başlangıcında bir önceki iterasyonda elde edilen çözüm ile devam edilir. Bu sayede global en iyi çözüme ulaşmak amaçlanır.

3.4. ALGORİTMANIN AVANTAJLARI VE DEZAVANTAJLARI

SA algoritması birçok çalışmada farklı yöntemler ile karşılaştırılmış, problemin özgünlüğüne ve parametrelerine göre başarılı ya da başarısız sonuçlar vermiştir. Yine SA algoritmasının performansını artırmak amaçlı birçok hibrit yöntem geliştirilmiş ve bu yöntemler de yine diğer sezgisel yöntemlerle karşılaştırılarak analiz edilmiştir.

3.4.1. Algoritmanın Güçlü Yönleri

SA algoritması yüksek dereceli doğrusal olmayan modeller, kaotik ve gürültülü veriler ve birçok kısıtlamalar ile başa çıkabilecek nitelikte olan genel ve sağlam bir tekniktir. Diğer yerel arama yöntemlerine göre başlıca avantajları, esnekliği ve global en iyi çözümü bulma yaklaşımındaki yeteneğidir. Bu modelin herhangi kısıtlayıcı özelliklere bağlılığının olmamasından dolayı çok yönlülük özelliği vardır. SA algoritmasının işlem adımları kolaylıkla düzenlenebilir bir yapıdadır. Algoritmanın önemli özelliklerin birisi de uygulanacak sistem için verilen optimizasyon algoritmasının performasını artırmak amaçlı yapılacak kod değişikleriyle bunu mümkün kılabilmek ve birden fazla probleme uyarlanabilir hale getirilebilme yeteneğidir [22].

3.4.2. Algoritmanın Zayıf Yönleri

SA algoritması sezgisel bir yöntem olduğundan, gerçek bir algoritmaya dönüştürebilmek için birçok seçenek gereklidir. Çözümler arasında bağlantılara ve onların hesaplanması için zamana ihtiyaç duyar. Problemi çözmek için gereken hesaplamalarda ve algoritmada hassas parametreler olabilir ve bunun için ince ayarlar yapmak gerekebilir. Sayıların hassaslığı SA algoritmasının elde ettiği sonucun kalitesine önemli derecede etki etmektedir [22].

3.4.3. Algoritmanın Diğer Yöntemlerle Karşılaştırılması

Verimli bir optimizasyon algoritması global en iyi çözümü bulmak için iki teknik kullanmalıdır. Bunlar; arama uzayında yeni ve bilinmeyen alanları araştırmak için keşif yapmak ve daha iyi noktaları bulmaya yardımcı olmak için önceden ziyaret edilen noktalarda elde edilen bilgilerden istifade etmektir. Bu iki durum birbiriyle çelişkilidir ve iyi bir arama algoritması bu iki durum arasında dengeli bir yol izlemelidir [22].

SA algoritması sinir ağları ile karşılaştırıldığında temel fark SA algoritmasının global en iyi çözüm için arama yaparken sinir ağlarının bu işlemi öğrenerek (fonksiyon ile

yaklaşımda bulunarak) yapmasıdır. SA algoritması akıllı rastgele arama yöntemi kullanırken sinir ağları esnek bir fonksiyonel yaklaşımda bulunur. Sinir ağları adaptif özellikleri değişen ortamları modelleme için avantajlıdır. SA algoritması Genetik algoritma (GA) ile karşılaştırıldığında temel fark GA'nın kendi başına bir sınıf olmasıdır. Çünkü GA bir optimizasyon algoritması olarak geliştirilmemiştir ve temel olarak GA optimal bir noktaya global yakınsama için istatistiksel bir garanti sunmamaktadır [22].

Bu bağlamda SA algoritmasının avantajları şöyle sıralanabilir:

- Farklı disiplinlerden gelen geniş bir problem yelpazesini optimize etmek için kullanılabilir [21].
- İstatistiksel olarak optimum çözümü bulma konusunda garanti verir [22].
- Karmaşık problemlerde dahi programın kolay bir şekilde kodlanabilmesini sağlar [22].
- Genellikle en iyi çözümü sunar [22].
- Arama stratejisini geliştirmek adına geçmiş bilgilerden yararlanabilir [23].
- Farklı komşuluk yapılarının kullanılmasına imkan sağlar [23].

SA algoritmasının dezavantajları ise şu şekilde sıralanabilir:

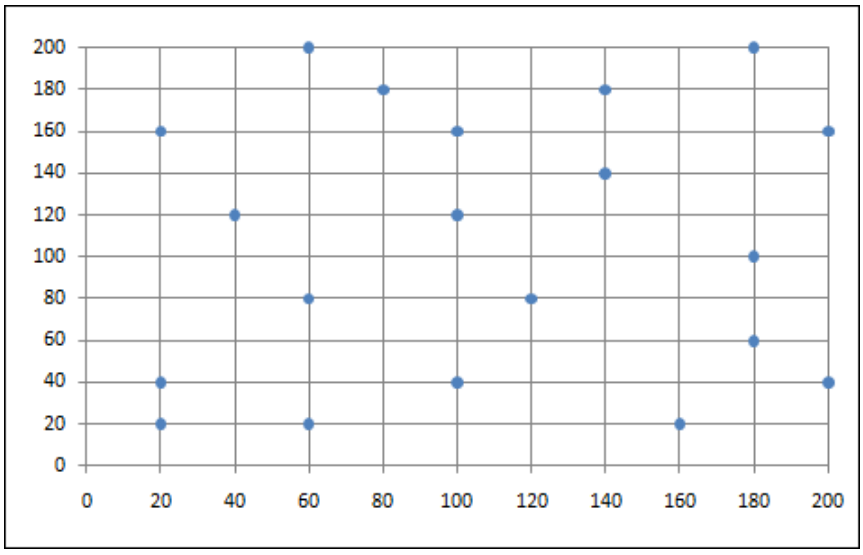
- Maliyet fonksiyonunun hesaplamasının zor olduğu durumlarda tavlama işlemi süresi uzamaktadır [22].
- Yerel minimum sayısının az olduğu problemlerde SA algoritması aşırı yükleme yapar, gereksiz uzar. Bunun yerine daha basit yöntemler tercih edilmelidir. Fakat genellikle çözülecek problemin yerel minimum sayısı bilinmemektedir [22].
- Yöntem optimal çözümü bulduğunu söyleyemez. Bunun için tamamlayıcı yöntemler kullanmak gereklidir [22].

3.5. GEZGİN SATICI PROBLEMİNİN BENZETİLMİŞ TAVLAMA İLE ÇÖZÜMÜ VE ANALİZİ

Gezgin Satıcı Problemi (Traveling Salesman Problem, TSP) bir seyyar satıcının elindeki mallarını *n* şehirde satmak istemesinden oluşmaktadır. Satıcı tüm şehirlere mümkün olduğu kadar en kısa yoldan ve maksimum bir kez uğrayarak turlamak istemektedir. Problemin çözümündeki amaç ise satıcıya en kısa yolu sunabilmektir.

Bu problem, 1930'lu yıllarda matematiksel olarak formüle edilmiştir. Optimizasyon konusunda başı çeken konular arasında yer alır. "Hesaplamanın karmaşıklığı" teorisine göre çözümü NP-tam olan en önemli algoritma problemlerinden biridir. Bundan dolayı bu problemleri tam bir şekilde çözebilecek bir algoritma olmadığı kabul edilmektedir. Şu anda çözülmeye çalışılan en büyük problem dünya üzerinde kayıtlı yerleşimi olan her nokta için en kısa yol problemidir. Bu problem 1,904,711 şehir içermektedir.

Problem için 20 adet şehir bilgisinin 2 boyutlu (2B) verilerinden yararlanılmıştır (Şekil 3.2). 20 şehri dolaşabilecek alternatif yol sayısı 19! gibi büyük bir sayıdır. En kısa yolun bulunabilmesi için 19! sayıda rotanın test edilmesi gerekmektedir. Bu sayı ise bugünkü teknolojide büyük bir problem demektir.



Şekil 3.2. Şehirlerin koordinat düzleminde gösterimi.

SA algoritmasının kodlanması sırasında TSP'ye uyarlanırken yapılan adımlar şunlardır:

1. Şehirlerin koordinat bilgilerinin tutulacağı diziler oluşturulur.
2. Her iterasyonda, iki şehir rastgele yer değiştirilir. Daha sonra maliyet hesaplanır ve bir önceki maliyet ile karşılaştırılır.
3. Maliyet düşükse yer değiştirme direkt olarak onaylanır değilse belli bir olasılık dahilinde yer değiştirme kabul ya da ret edilir.
4. Her iterasyonda sıcaklık belli bir katsayı ile düşürülür.

Programın çalıştırılmasından sonraki aşamada sonuçların daha iyi olabilmesi amacıyla parametrelerde değişiklik yapılarak sonuçlar test edilmiştir. İlk çalışmada *Soğutma Katsayısı (α) = 0,95, Başlangıç Sıcaklığı (T) = 1e+10, Durma Sıcaklığı (DT) = 0.001* olarak belirlenmiştir. Sonuçlar programın çalıştığı ilk hesaplama ile son hesaplama arasındaki verimliliği referans alınarak değerlendirilmiştir. Buna göre ilk çalışmada **%28** oranında gelişme sağlanmıştır. İterasyon sayısının *10* olarak belirlendiği diğer testlerde ise sonuçlar şu şekildedir:

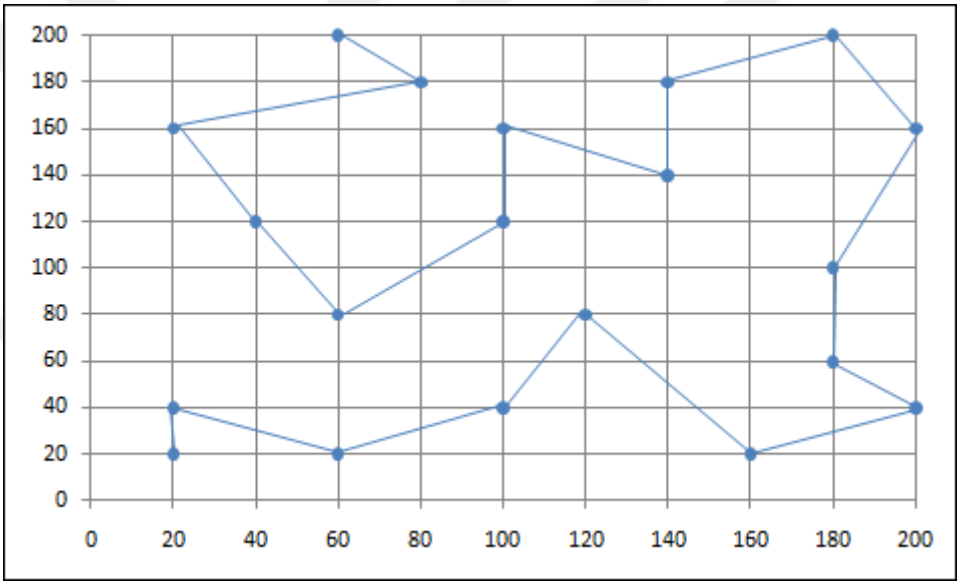
*α = 0.95, T= 1e+10, Gelişme: **%28**, Süre: 0.034 s (ilk test)*
*α = 0.95, T= 1e+30, Gelişme: **%33**, Süre: 0.033 s*
*α = 0.99, T= 1e+10, Gelişme: **%50**, Süre: 0.058 s*
*α = 0.99, T= 1e+30, Gelişme: **%43**, Süre: 0.07 s*

Bu sonuçlara bakıldığında soğutma katsayısının *0.99* olarak değiştirilmesinden sonra sonuçların daha iyi olduğu gözlemlenmiştir. Başlangıç sıcaklığını artırmak ise sonuca direkt olarak bir katkı sağlamamıştır. Bir diğer parametre olan durma sıcaklığı ile ilgili testlerde ise sonuçlar şu şekildedir *(α = 0.95, T= 1e+30):*

*DT = 0.001, Gelişme: **%42**, Süre: 0.009 s*
*DT = 0.01, Gelişme: **%41**, Süre: 0.013 s*
*DT = 0.1, Gelişme: **%31**, Süre: 0.033 s*
*DT = 1, Gelişme: **%24**, Süre: 0.006 s*

Sonuçlara bakılacak olursa durma sıcaklığının düşürülmesi daha iyi sonuçların elde edilmesini sağlamıştır. Bu parametrenin mümkün olduğu kadar düşük olması daha iyi sonuçlar elde edileceği anlamına gelmemektedir. Zira bu sayının azaltılması yerel minimuma takılma riski olduğundan tavlama sürecinin sonlarına doğru düzgün bir konfigürasyon değişikliğine engel olur [24].

Tüm bu testlerin sonucunda parametreler $\alpha = 0.99$, $T = 1e+10$, $DT = 0.0001$ olarak belirlenmiştir. Bu parametreler ile program iterasyon sayısı 1000 yapılarak test edilmiş ve **%53** gelişme sağlanmıştır. Sonuçta elde edilen yol ise Şekil 3.3'te gösterilmiştir.



Şekil 3.3. Bulunan en kısa yolun koordinat düzlemi üzerinde gösterimi.

Sonuçlara bakılacak olursa SA algoritmasının bu tarz problemler için uygun bir yöntem olduğunu söylemek mümkündür. Ayrıca problemin çözümüne göre uygun parametreler belirlemek optimum çözüm elde etmek konusunda etkilidir. İterasyon sayısının artması yani algoritmayı birden çok kez çalıştırmak daha iyi sonuçların elde edilmesine imkan sağlamak ile birlikte zaman problemini ortaya çıkarmaktadır. Çözülecek problemin boyutu arttıkça zaman problemi de büyük bir problem haline gelmektedir. Bu amaçla uygulamaların paralelleştirilmesi sonuca ulaşmak için gereken süreyi ciddi miktarda düşürecektir.

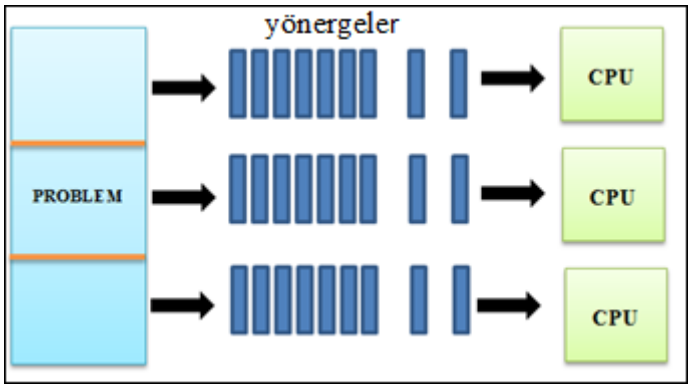
BÖLÜM 4

CUDA İLE PARALEL PROGRAMLAMA

Bu bölümde GPU'lar hakkında kısaca bilgi verilmiştir. GPGPU teknolojisinin ortaya çıkma nedenlerine değinilmiş ve paralel programlama üzerindeki etkisi belirtilmiştir. Tez çalışmasında kullanılan CUDA mimarisi hakkında bilgi verilmiş, CUDA'nın programlama modeli ve bellek hiyerarşisi anlatılmıştır. Bölüm, CUDA ile programlamanın nasıl yapıldığına değinilerek sonlandırılmıştır.

4.1. GPU VE GPGPU

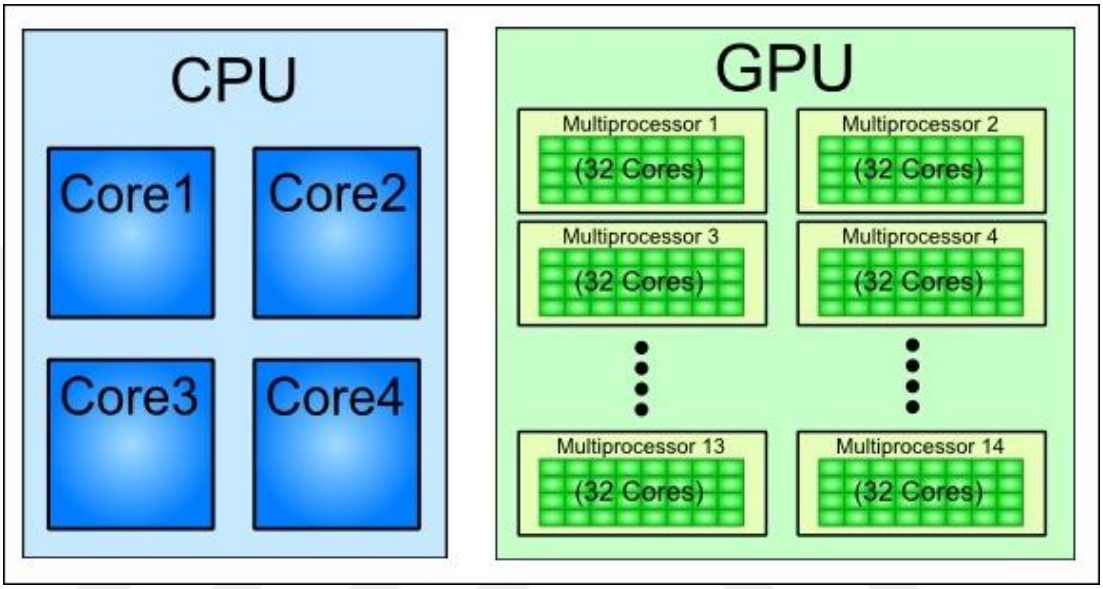
Paralel hesaplamada mevcut hesaplama kaynaklarının (bir bilgisayar için tüm işlemciler) aynı anda kullanılabilmesi söz konusudur. Paralel sistemlerde bir problem için geliştirilen adımlar eş zamanlı olarak farklı işlemciler üzerinde yürütülmektedir. Şekil 4.1'de yer alan görsel bu durumu özetlemektedir [1].



Şekil 4.1. Paralel hesaplama.

Son yıllarda kişisel bilgisayarlar üzerinde bulundan paralel sistemlerin gelişimi hızlanmıştır. Oyun sektöründeki gelişim sayesinde grafik kartlarının CPU'lara göre daha hızlı bir gelişme göstermesi sonucu GPGPU teknolojisi paralel programlamada önemli bir çalışma alanı haline gelmiştir [25].

GPU'nun CPU'ya nazaran en önemli özelliği çekirdek (core) sayısının fazla olmasıdır. GPU üzerinde bulunan binlerce iş parçacığı aynı anda işlem yapabilmekte ve paralel hesaplamaya uygun dizayn edilmesi neticesinde verimli sonuçlar elde edilebilmektedir. Şekil 4.2'de CPU ve GPU arasındaki çekirdek sayılarının karşılaştırılması gösterilmiştir [26].



Şekil 4.2. CPU ve GPU arasındaki çekirdek sayılarının karşılaştırılması.

Şekil4.2'de görüleceği üzere GPU üzerindeki çekirdek sayısında CPU'daki çekirdek sayısına oranla büyük bir artış söz konusudur. Bu sayede GPU mimarisi yüksek işlem performansı sağlamaktadır. GPGPU aşağıdaki uygulama alanlarına uygun bir yapı olarak sunulabilir [27]:

- Büyük veri setleri,
- Yüksek paralellik,
- Veri öğeleri arasında minimum bağımlılıklar,
- Yüksek aritmetik yoğunluk,
- CPU müdahalesi olmadan yapılacak fazla sayıda iş/işlem.

GPGPU teknolojisini uygulayan ve kullanımı yaygın olan iki temel platform vardır. Bunlar OpenCL (Open Computing Language) ve CUDA'dır. OpenCL, AMD, Intel, NVIDIA ve ARM tarafından desteklenen kar amacı gütmeyen, paralel hesaplamanın

hızlandırılması gibi konularda standartlar sunan bir işbirliği şirketi olan Khronos Group tarafından geliştirilmektedir. Gerek ATI gerekse NVIDIA tarafından üretilen ekran kartları üzerinde paralel programlamaya imkan sağlar. CUDA ise NVIDIA firması tarafından geliştirilen genel amaçlı bir paralel programlama mimarisidir. CUDA yalnızca NVIDIA tarafından üretilen ekran kartları üzerinde paralel programlamaya imkan tanır.

4.2. CUDA PROGRAMLAMA MODELİ

CUDA™, NVIDIA tarafından Kasım 2006'da tanıtılan genel amaçlı bir paralel hesaplama mimarisi olup, birçok karmaşık hesaplama problemini CPU'dan daha verimli bir şekilde çözmek için NVIDIA tarafından üretilen GPU'larda bulunan paralel hesaplama motorunu kullanmaktadır [28].

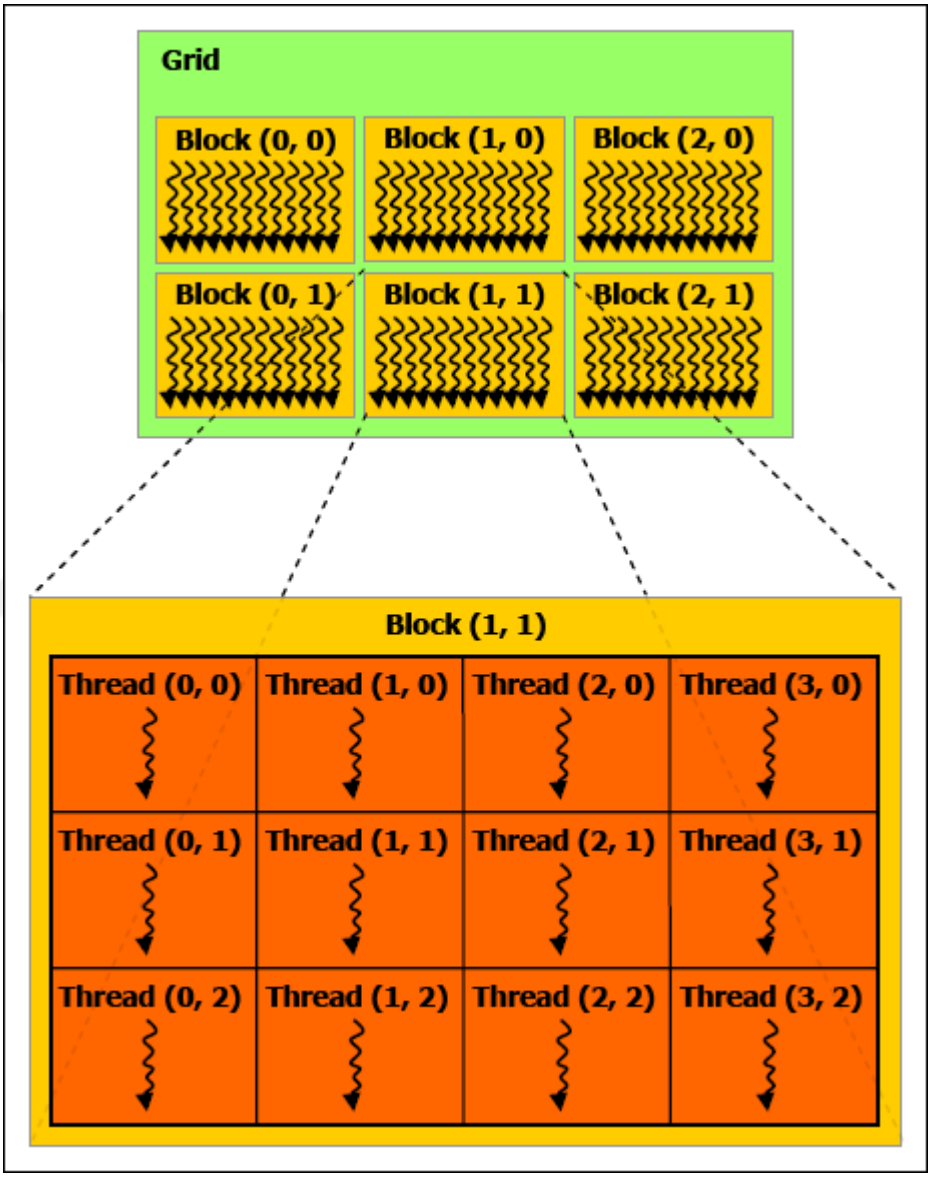
GPU, GPU modeline bağlı olarak sayısı değişen Eşzamanlı Çoklu İşlemci'lerden (Streaming Multiprocessor, SM) oluşur. Her SM'nin kendi saklayıcıları (register) ve paylaşılan bellek bloğu (shared memory block) vardır. Bundan sonraki bölümlerde, NVIDIA tarafından kullanılan terminolojiye uygun olarak ekran kartına *device*, CPU'ya ise (ana bellek ile birlikte) *host* denilecektir.

4.2.1. İş Parçacığı Organizasyonu

GPU üzerindeki temel işlem birimleri olan iş parçacıkları (threads) üç aşamalı bir hiyerarşide sunulmuştur. Alt seviyede iş parçacıkları *warp* olarak gruplandırılır, her bir *warp*ta 32 iş parçacığı vardır ve her bir warp iki *half-warp*tan oluşur. *Half-warp*ın karakteristiği SIMD mimarisine benzemektedir, çünkü *warp*ta bulunan tüm iş parçacıkları aynı komut kümesini aynı anda yürütürler.

Warp, CUDA dil tanımının bir parçası olmadığından (programcı *warp* üzerine direkt olarak etki edemez) programlama bakış açısıyla iş parçacıkları bloklara (*thread blocks*) ayrılır. Bir iş parçacığı bloğunun boyutu programlayıcı tarafından belirlenir ve *warp* boyutunun katları olan bir boyutun seçilmesi önerilmektedir. Her bir blok birbirinden bağımsız olarak çalışmaktadır. Her bir bloğun bağımsız olarak çalışması,

bloklar-arası senkronizasyonu ve hızlı bellek paylaşımını zor hale getirmektedir [29]. Bununla birlikte bir blok içinde, iş parçacıkları, paylaşılan belleği (*shared memory*) kullanarak birbirleriyle işbirliği yapar ve iş parçacıklarının işlemlerini senkronize etmek için bariyer fonksiyonu olarak *__syncthreads()* fonksiyonunu kullanabilirler. CUDA iş parçacıklarının organizasyonu Şekil 4.3'te gösterilmiştir.



Şekil 4.3. CUDA iş parçacıkları organizasyonu [28].

İş parçacıklarının blokları, sırasıyla bir veya iki boyutlu *grid*lere (ızgara) ayrılır. İş parçacıklarının boyutu genel olarak optimizasyon işlemi tarafından belirlenirken, bir

*grid*deki blokların sayısı ele alınan veri boyutuna göre belirlenir. Bir grafik kartında yalnızca bir *grid* bulunur ve bir sistemde her *device* için ayrı bir *grid* bulunmaktadır.

4.2.2. Bellek Hiyerarşisi

CUDA iş parçacıkları işlemlerini yürütürken Şekil 4.4'te gösterildiği gibi, birden fazla bellek alanındaki verilere erişebilirler. Her bir iş parçacığı kendine özel yerel belleğe (*local memory*) sahiptir. Her bir iş parçacığı bloğu, bloğun tüm iş parçacıkları tarafından erişilebilen ve blokla aynı ömre sahip olan paylaşılan belleğe (*shared memory*) sahiptir. Tüm iş parçacıklarının aynı global belleğe (*global memory*) erişimi vardır.

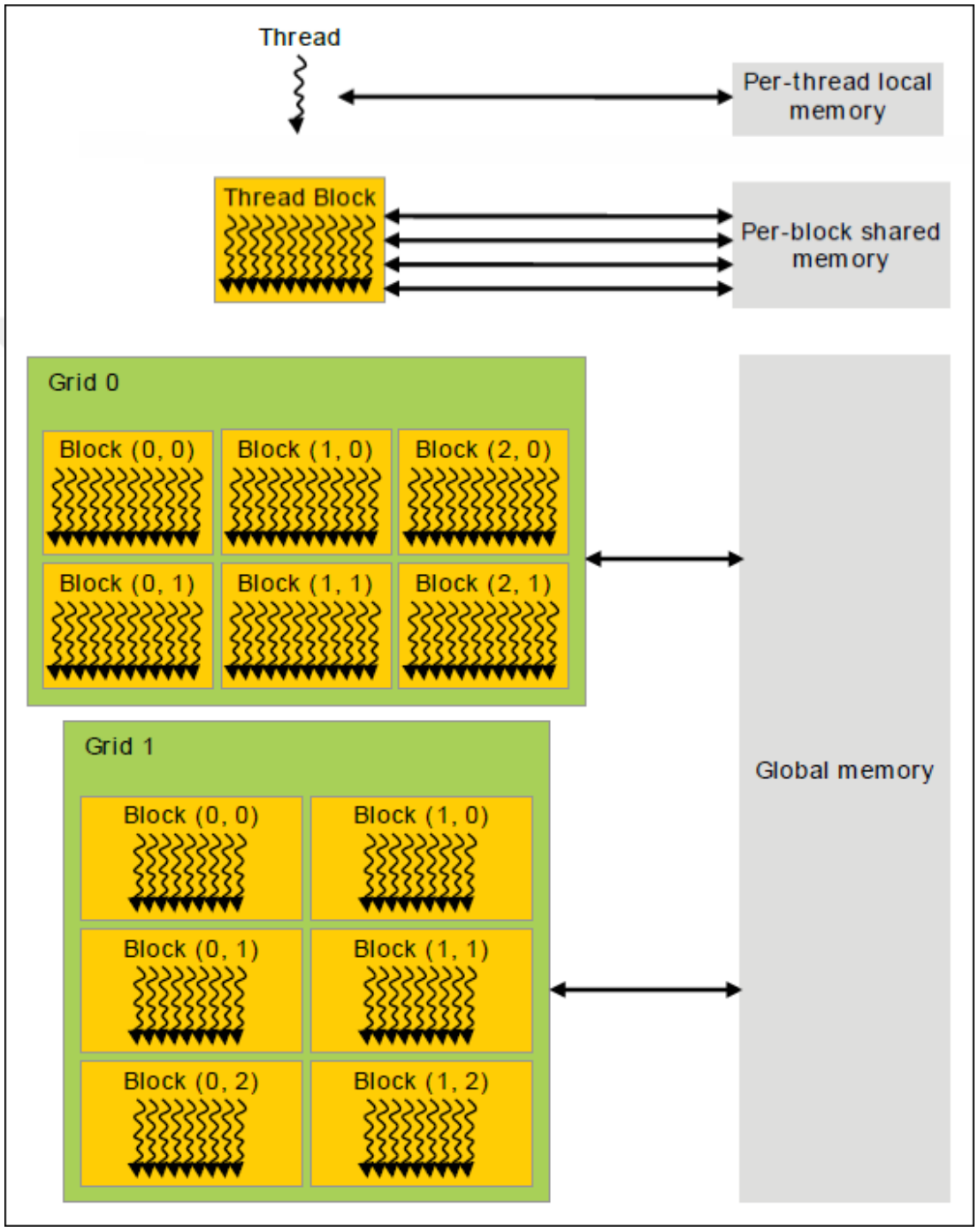
Ayrıca, tüm iş parçacıkları tarafından erişilebilen salt-okunur (*read-only*) bellek alanı olan *constant* ve *texture* bellek alanları vardır. *Global*, *constant* ve *texture* bellek alanları, farklı bellek kullanımı için optimize edilmiştir. Şekil 4.4'te bellek hiyerarşisine ait görsel sunulmuştur.

4.2.3. CUDA İle Programlama

CUDA, birkaç programlama dilini içeren eklentilere sahiptir. Bunlardan bir tanesi de C programlama dilidir. Bu eklenti *device* üzerindeki belleği yönetmek, GPU ile CPU belleği arasında veri aktarmak ve birden fazla grafik kartına sahip olan sistemleri yönetmek için fonksiyonlar sunan bir Uygulama Programlama Arayüzü'dür (Application Programming Interface, API).

Temelde bir CUDA programında ortaya çıkan üç fonksiyon grubu vardır, bunlar; *host*, *global* ve *device* fonksiyonlarıdır. *Host* fonksiyonları normal bir C programındaki fonksiyonlar gibi CPU üzerinde yürütülür. *Kernel* olarak adlandırılan genel fonksiyonlar *void* türündedir ve bir *host* fonksiyonu tarafından çağırılarak GPU üzerinde yürütülür. Bunlar, kullanıcıların blok sayısını ve her bloğun boyutunu belirtmesi gereken özel yapılandırmaları da içermektedir. Bir *device* fonksiyonu, bir *host* fonksiyonu ya da diğer *device* fonksiyonları tarafından çağırılabilir. Bununla birlikte bir fonksiyonu *device* ya da *host* olarak tanımlamak mümkündür. Bunun için

fonksiyon türüne göre *__host__*, *__global__* veya *__device__* ön ekini alarak tanımlanmalıdır. Bir fonksiyon tanımlanırken herhangi bir önek kullanılmaz ise CUDA fonksiyon türünün *__host__* olduğunu varsaymaktadır.



Şekil 4.4. CUDA bellek hiyerarşisi [28].

Şekil 4.5'te N boyutundaki A ve B vektörünün toplamını C vektörüne atayan bir örnek kod gösterilmiştir:

Kod 1. Device Kodu ve Host Üzerinden Kernelin Başlatılması

```
1.  // Kernel definition
2.  __global__ void VecAdd(float* A, float* B, float* C)
3.  {
4.  int i = threadIdx.x;
5.  C[i] = A[i] + B[i];
6.  }
7.  int main()
8.  {
9.  ...
10. // Kernel invocation with N threads
11. VecAdd<<<1, N>>>(A, B, C);
12. ...
13. }
```

Şekil 4.5. *Device* kodu ve *Host* üzerinden kernel'in başlatılması [28].

Device üzerindeki *global* bellek üzerinde, *cudaMalloc* ve *cudaFree* komutları kullanılarak yer ayrılmalı ve boşaltılmalıdır. Bellek alanları bu komutlara göre belirlenir (Şekil 4.6).

Kod 2. Global Bellek Yönetimi

```
1.  /* Memory allocation */
2.  cudaMalloc (( void **)& deviceMemoryPointer ,
3.  memorySize );
4.  /* Memory deallocation */
5.  cudaFree ( deviceMemoryPointer );
```

Şekil 4.6. Global bellek yönetimi.

Host bellek ile *device* bellek arasında veri aktarımı yapmak için *cudaMemcpy* fonksiyonu kullanılır. Şekil 4.7'deki ilk iki parametre hedef ve kaynak göstericidir. Sonraki parametre, aktarılacak olan verinin boyutudur. Son parametre ise verinin aktarım yönünü belirler - bu da *cudaMemcpyHostToDevice*, *cudaMemcpyDeviceToHost* veya *cudaMemcpyDeviceToDevice* olabilir [29].

Kod 3. Device Bellek Transferi

```
1.  /* memory allocated on host */
2.  hData = (int *) malloc (n* sizeof (int ));
3.  /* Memory transfer */
4.  cudaMalloc (( void **)& dData , n* sizeof (int ));
5.  cudaMemcpy (dData ,hData ,n* sizeof (int),
6.  cudaMemcpyHostToDevice );
7.  ...
8.  free ( hData ); cudaFree ( dData );
```

Şekil 4.7. Device bellek transferi.

Bir *kernel* çağrıldığında, iş parçacıkları bazı yapılandırma değişkenleri kullanabilir. Bu değişkenlerin türü *uint3* veya *dim3* olarak adlandırılır. *GridDim* yürütülecek olan *kernel*'de kullanılacak blok sayısını tanımlar. *BlockDim*, her bir bloğun boyutunu tanımlar, *blockIdx* ise belirli bir bloğun dizin numarasını verir. *ThreadIdx*, bir blok içindeki iş parçacığının dizin numarasına erişmek için kullanılır. Belirli bir iş parçacığının genel dizin numarasını elde etmek için, Şekil 4.8'de sunulan programdaki değişkenleri kullanarak hesaplama yapmak gerekir. Bu program, b vektöründeki her bir elemanı 2 ile çarpıp sonucu a vektörüne kaydeder [29].

Kod 4. Kernel İçinde Global İş Parçacığı Dizinini Belirleme

```
1.  __global__ void kernel (int *a, int *b)
2.  {
3.  int localIndex = threadIdx.x;
4.  int globalIndex=blockIdx.x*blockDim.x + localIndex;
5.
6.  a[ globalIndex ] = 2*b[ globalIndex ];
7.  }
```

Şekil 4.8. Kernel içinde global iş parçacığı dizinini belirleme.

ThreadIdx.x, x boyutundaki elemanlara erişildiğini, bu da bir boyutlu bir dizi için olduğu anlamına gelir. Bu boyut 1, 2 veya 3 boyutlu olarak tanımlanabilir.

BÖLÜM 5

DENEYSEL ÇALIŞMALAR

Bu bölümde SA algoritmasının GPU üzerinde CUDA aracılığıyla paralelleştirilerek farklı problemlere uygulanmasından bahsedilmiştir. Karesel Atama Problemi, Sırt Çantası Problemi ve Silah-Hedef Atama Problemi olmak üzere üç farklı problem üzerinde yapılan çalışmalar anlatılmıştır. Her bir problem için sonuçlar seri algoritma ile karşılaştırılarak hızlanma değerlendirilmesi yapılmış, literatürde mevcut veri setleri olan problemler ise bu problemlerin bilinen en iyi değerleri ile karşılaştırılmak suretiyle değerlendirmeye tabi tutulmuştur.

5.1. KARESEL ATAMA PROBLEMİ (QAP)

Karesel Atama Problemi (Quadratic Assignment Problem, QAP) kombinatoryal optimizasyon problemlerinden biri olan ve NP-zor sınıfında gösterilen uygulama alanı geniş bir problemdir. Problem, eşit sayıda tesis (facility) ve bölge (location) içermekte ve tesislerin bölgelere yerleşimini en az maliyet ile gerçekleştirmeyi hedeflemektedir. İlk olarak 1957 yılında Koopmans ve Beckmann tarafından sunulmuştur [30]. Problemin çözümünden sonra her tesisin bir bölgeye atanması sağlanmalı ve hiçbir tesisin boşta kalmaması gerekmektedir. QAP için toplam maliyet, probleme atanan herhangi bir permütasyon(π) için aşağıdaki gibi formülize edilebilir:

$$c(\pi) = \sum_{n=1}^{n} d_{i,j} \ . f_{\pi(i),\pi(j)} \tag{5.1}$$

d: i bölgesi ile j bölgesi arasındaki mesafe,

f: i tesisi ile j tesisi arasındaki birim uzaklığın maliyeti,

n: d ve f değerlerini içeren matrislerin boyutu,

c: maliyet.

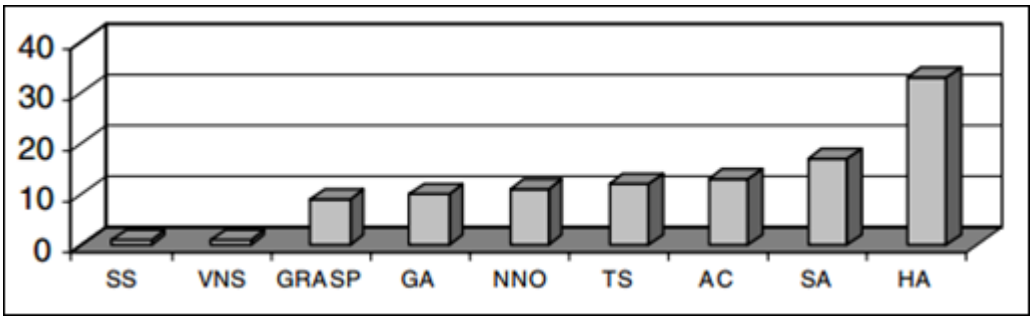
QAP; pota kablolama bileşenleri arasındaki bağlantı sayısını tespit etmede [31], zaman çizelgeleme probleminde [32], daktilo klavyesi ve kontrol paneli tasarımında [33], bilgisayar destekli plan tasarımında [34], nümerik analizde [35] kullanılmıştır. Bununla birlikte en yaygın kullanım alanı tesis yerleşim problemi olarak görülmektedir. Dickey ve Hopkins üniversite yerleşkesi içerisindeki binaların yerleşiminde [36], Elshafei hastane yerleşim planında [37], Bos ise orman parklarının imarı ile ilgili bir problemde [38] QAP'den yararlanmışlardır.

5.1.1. QAP'nin Çözüm Yöntemleri

QAP NP-zor sınıfı bir problem olduğu için problemin boyutu arttıkça çözüm zamanı da üstel olarak artmaktadır. QAP için global en iyi çözüme ulaşmak amaçlı kullanılan değişik yöntemler vardır. Bunlar, dal-sınır, kesen düzlemler ya da bu yöntemlerin kombinasyonları olan dal-kesim yöntemi ve dinamik programlamadır. Dal-sınır en bilinen ve kullanılan yöntemlerin başında yer alır. Son yıllarda, dal-sınır teknikleri ile kombine edilen yöntemlerin paralel uygulamaları literatürde yer almaktadır. Bununla birlikte başarılı sonuçlar elde edilmeye başlanmıştır. Ancak problemin boyutu ve donanım teknolojisi burada önemli rol oynamaktadır [39]. QAP NP-zor bir problem olduğundan ve kesin çözüm yöntemlerinin yüksek boyutlu problemlerdeki sorunlarından dolayı makul bir sürede en iyi ya da en iyi çözüme yakın sonucu sağlayan sezgisel yöntemler tercih edilmektedir.

QAP'nin çözümünde kullanılan sezgisel yöntemlere örnek olarak Benzetilmiş Tavlama (SA), Genetik Algoritma (Genetic Algorithm, GA), Sinir Ağları (Neural Network, NNO), Dağılım Araması (Scatter Search, SS), Karınca Kolonisi (Ant Colony, AC), Tabu Arama (Tabu Search, TS), Açgözlü Rastgele Adaptif Arama Yöntemi (Greedy Randomized Adaptive Search Procedure, GRASP), Değerli Komşuluk Arama (Variable Neighborhood Search, VNS) verilebilir. Bununla birlikte farklı sezgisel yöntemlerin bir arada kullanıldığı hibrit çalışmalar da literatürde

mevcuttur. Yalın olarak kullanılan sezgisel yöntemlere bakıldığında ise SA, TS ve AC en popüler yöntemler arasında yer almaktadır. Şekil 5.2'de ise QAP için kullanılan sezgisel yöntemlerin dağılımı gösterilmiştir.



Şekil 5.1. QAP için kullanılan sezgisel yöntemlerin dağılımı [39].

Sezgisel yöntemlerin çalışma sürelerinin çok uzun olması paralel çalışabilecek algoritmaların tasarlanmasını zorunlu kılmıştır. Birçok sezgisel yöntem CPU veya GPU üzerinde paralelleştirilmiştir. GPU üzerinde yapılan çalışmalarda CUDA mimarisi kullanılmış ve farklı alanlarda uygulamaları yapılmıştır.

5.1.2. Literatürde QAP'nin Çözümü İçin Sunulan Paralel Yöntemler

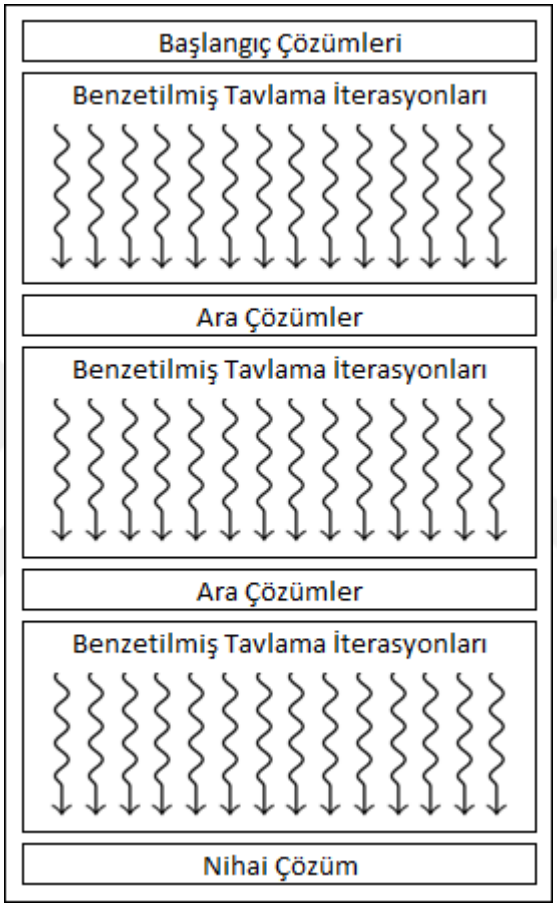
Günümüz teknolojisinin geldiği nokta dikkate alınırsa literatüre bakıldığında QAP'nin çözümü için yapılan paralelleştirme çalışmalarının yeterli sayıda olmadığı görülmektedir. Sezgisel yöntemlerle yapılan birçok çalışmanın belli ölçülerde başarıya ulaştığı görülmekte fakat sonuca ulaşırken yöntemle ilgili düzenlemeler yapılmakta ve sürenin büyük bir problem olduğu ortaya çıkmaktadır. Bunun en temel nedeni ise seri yöntemlerle genellikle aynı anda sadece bir çözümün gerçekleştirilmesidir. Ancak paralel yöntemlerle aynı anda eş zamanlı olarak farklı çözümler de denenebilmekte bu da çözüme ulaşma olasılığını artırmaktadır. Aynı zamanda günümüz donanım teknolojisi ile paralel yöntemler zaman yönünden büyük faydalar sağlamaktadır. Ayrıca GPU teknolojisinin günümüzdeki teknolojik yükselişi paralel çalışmaları CPU'dan GPU tarafına doğru yöneltmiştir. Ancak GPU üzerinde paralelleştirme çalışması CPU'ya nazaran daha çok bilgi ve tecrübe istemektedir. Bu nedenle GPU üzerindeki çalışmalar az sayıdadır. QAP uygulamalarında kullanılan yöntemlerinin başında SA ve TS algoritması gelmektedir.

James vd. [40] CPU üzerinde TS algoritmasını kullanarak bir çalışma yapmışlar, her adımda iş parçacıkları arasında bir işbirliği yaparak global çözüme ulaşmayı amaçlamışlardır. Bu amaçla yaptıkları çalışmanın adına "İşbirlikçi Paralel Tabu Arama Algoritması" adını vermişlerdir. QAP için birçok çalışmada da kullanılan QAPLIB [41] kütüphanesindeki 60 adet veri setinde yaptıkları çalışmada, birçok veri setinde makul bir sürede iyi sonuçlar elde ettiklerini belirtmişlerdir. Yine literatürde bulunan seri ve paralel yöntemlerin sonuçlarına yakın veya daha iyi sonuçlar elde ettiklerini söylemişlerdir. Çalışmalarında Çoklu-Başlangıç (Multi-Start) kullanmışlardır. Kováč [42] yaptığı çalışmada, SA algoritması ile QAP'yi paralel yöntemle çözmeyi amaçlamış birkaç QAPLIB veri seti üzerinde test etmiştir. Genel ve yüzeysel olarak CPU üzerinde en fazla 32 iş parçacığında yaptığı testleri literatürdeki üç farklı sonuçla karşılaştırarak analiz etmiştir. GPU üzerinde yapılan çalışmalara bakılacak olursa literatürde QAP için önerilmiş çalışmalar çok azdır. Farklı problemler üzerinde yapılmış özelleştirilmiş çalışmalar mevcuttur. QAP için ise GPU üzerinde CUDA kullanılarak yapılan iki çalışma dikkati çekmektedir. Zhu vd. [43] SIMD-TS ismini verdikleri çalışmalarında GPU üzerinde TS algoritması kullanarak QAP için bir yöntem geliştirmişler ve CPU üzerinde seri bir yönteme nazaran 20 kattan 45 kata kadar hız sağlamışlardır. Ancak bu çalışmada iş parçacıkları arasında herhangi bir iletişim gerçekleşmemektedir. Czapiński [44] ise yaptığı çalışmada, Zhu ve arkadaşlarının yaptığı gibi TS algoritmasını kullanmış ve GPU üzerinde çoklu-başlangıç tekniği (multi-start) kullanarak bir yöntem geliştirmiştir. Yaptığı uygulamada iş parçacıkları arasında haberleşme sağlayarak en iyi çözüme ulaşırken bir sonraki adım için geliştirilecek permütasyonu iş parçacıklarına dağıtmıştır. Yaptığı analizlere göre 420 kata kadar hızlanma sağlamıştır. Tosun [45] yaptığı çalışmada, paralel hibrit algoritmaların QAP'nin çözümü üzerindeki performası ile ilgili bir çalışma yapmış, genetik algoritma ile birçok yerel arama algoritmalarının başarılı bir şekilde birleştirilerek kullanılabileceğini belirtmiş ve başarılı sonuçlar elde edilebileceğini göstermiştir.

5.1.3. QAP'nin Çözümü İçin Sunulan Paralel SA Algoritması (PSA-QAP)

Farklı başlangıç permütasyonları veya konfigürasyonları ile sezgisel yöntemlerin kombinatoryal problemlerin çözümü için kullanılmasında başarılı sonuçlar elde

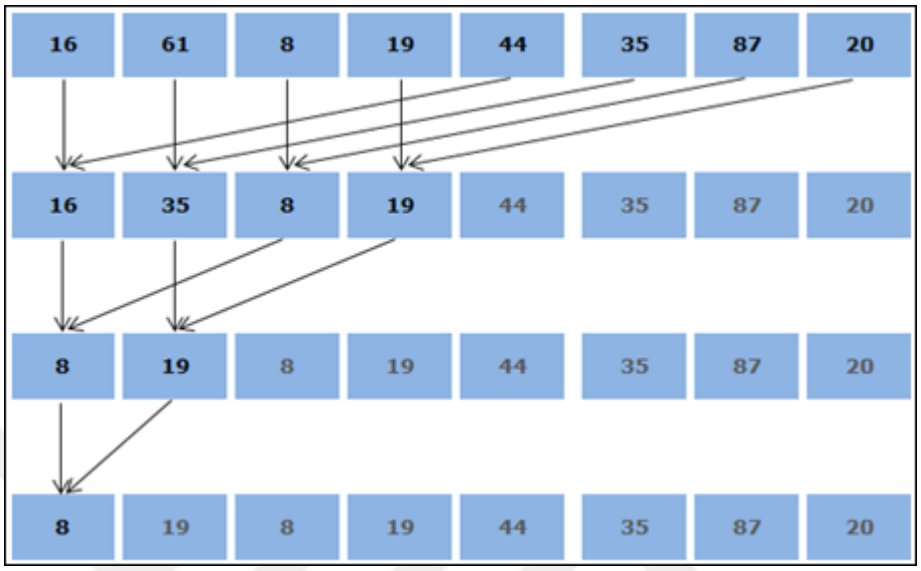
edildiği ortaya koyulmuştur [46]. Bu yöntem ile gerek CPU gerekse GPU üzerinde yapılan paralelleştirme ile aynı anda farklı başlangıç permütasyonlarına sahip sezgisel algoritmayı yürütmek mümkündür. Birçok sezgisel algoritmada olduğu gibi SA algoritmasında da bu yöntem kullanılmıştır [47]. Şekil 5.2'de literatürde sıkça kullanılan çoklu-başlangıç tekniği gösterilmiştir.



Şekil 5.2. Paralel SA algoritması.

GPU üzerindeki bloklarda bulunan iş parçacıkları, kendi bloklarındaki iş parçacıkları ile haberleşmektedir. CUDA platformunda *__syncthreads()* kullanılarak aynı blok içerisindeki iş parçacıklarının işlemlerini tamamlamasını sağlamak mümkündür. Bu sayede iş parçacıklarının istenen işlemleri tamamlamasını sağlamaya imkan kılınmaktadır. PSA-QAP algoritmasında, her iterasyon sonucu blok içerisinde bulunan ve en iyi sonuca sahip olan iş parçacığına ait permütasyon dizisi diğer iş parçacıkları ile paylaşılır. Bir sonraki iterasyona blok içerisindeki tüm iş parçacıklarının bu konfigürasyon ile başlaması sağlanmaktadır. İş parçacıkları

arasında en iyi sonuca ait iş parçacığının bulunması için reduction (indirgeme) yöntemi kullanılmıştır (Şekil 5.3). Bu yöntem ile hesaplama karmaşıklığı *O(n)* yerine *O(logn)* olmaktadır.



Şekil 5.3. İndirgeme yöntemi kullanarak en küçük değeri bulma.

GPU'da bulunan bloklar üzerinde iş parçacıkları sayısına ait bir üst limit vardır. Bloklar-arası haberleşme ile ilgili herhangi bir mekanizma da mevcut değildir. Bunun için bir bloklar-arası haberleşmeye imkan sağlayan bir yöntem kullanılmıştır.

Bloklar-arası haberleşmenin en temel yolu GPU üzerinde çalışan *kernel*'i sonlandırmaktır. CPU aktif olduktan sonra *kernel*'i tekrar başlatmak tüm iş parçacıklarının senkronize olmasını sağlar. Ancak bu yöntem CPU ile GPU arasındaki veri alışverişini artırmakta ve zaman yönünden verimli bir yöntem olarak görülmemektedir. Bu nedenle farklı bloklardaki iş parçacıkları arasında haberleşmeye imkan sağlayan Xiao ve Feng [48] tarafından sunulan yöntem kullanılmıştır. Bu yöntem global bir değişken üzerinde iş parçacıklarının hedef bir değeri elde edene kadar beklemesi esasına dayanmaktadır. Ancak bu yöntemde bloklar-arası bekleme sırasında kilitlenme (deadlock) gerçekleşmemesi için kullanılan blok sayısı Streaming Multiprocessor (SM) sayısı ile sınırlandırılmaktadır. Bu yöntemden önce blok-içi haberleşme yöntemi ile blok içinde bulunan en iyi sonuca sahip iş parçacığı tespit edilmektedir. Daha sonra her blokta bulunan bu iş

parçacıkları, sunulan yöntem kullanılarak haberleşmekte ve tüm bloklar içindeki en uygun sonuca ait permütasyon dizisi elde edilmektedir. Global bir değişken aracılığıyla bu permütasyon dizisi tüm iş parçacıkları ile paylaştırılarak bir sonraki iterasyona bu permütasyon ile başlanması sağlanır. Geliştirilen yöntemde her iş parçacığı tarafından yapılan adımlar şu şekildedir:

1. İlk iterasyona başlamadan önce iş parçacıklarına özel, rastgele bir permütasyon dizisi oluştur. (Bu sayede çoklu-başlangıç yaklaşımı sağlanır.)
2. Oluşturulan permütasyon dizisi ile problemin çözümüne başla.
3. Rastgele belirlenen iki sayı ile yer değiştirme (swapping) işlemi sonucu oluşabilecek maliyet farkını (*delta* fonksiyonu (5.2) kullanarak) hesapla.
4. *Delta* fonksiyon değeri 0'dan küçük ise ya da metropolis kriterine göre hesaplanan olasılık değeri rastgele belirlenen bir sayıdan (0 ve 1 arası) büyük ise yer değiştirme işlemini kabul et.
5. *Delta* fonksiyon değeri 0'dan küçük ise yer değiştirme işlemi neticesinde oluşan yeni permütasyon dizisini kabul et ve sakla.
6. Hedef sıcaklığa ulaşılmadıysa Adım 3'e git, ulaşıldı ise bir sonraki adımla devam et.
7. İterasyon sonucu ortaya çıkan maliyet fonksiyonu değerini hesaplayarak *__shared__* bir değişkende iş parçacığına ait yere ata.
8. Aynı bloktaki diğer iş parçacıklarının Adım 7'yi tamamlamasına kadar bekle.
9. İndirgeme yöntemi ile blok içersindeki iş parçacıkları arasında optimum sonuca ait iş parçacığının tespit edilmesi işlemi için gereken iş birliğini gerçekleştir.
10. Her bloktaki en iyi iş parçacıklarının değerlerinin global bir değişkene atanması için gereken katılımı gerçekleştir.
11. Tüm iş parçacıklarının Adım 10'a kadar yapılan işlemleri sağlaması için bekle.
12. Tüm bloklar içinde optimum sonuca ait iş parçacığının bulunması için gerekli iş birliğine katıl. İşlemin tüm iş parçacıkları tarafından tamamlandığından emin olana kadar bekle.
13. Optimum sonucu global optimum sonuca (*best*) ait değişkene ata. Bu sonuca ait iş parçacığının permütasyon dizisinin bir sonraki iterasyonda tüm iş parçacıklarıyla paylaşmak üzere global bir değişkene atanmasını sağla. İşlemin tüm iş parçacıkları tarafından tamamlandığından emin olana kadar bekle.

14. Bir sonraki iterasyona geçmeden önce optimum sonuca ait permütasyon dizisini global değişkenden al ve belirlenen iterasyon sayısı tamamlanana kadar Adım 3'den devam et. İterasyon sayısı hedef sayıya ulaştığında işlemi sonlandır.

5.1.4. PSA-QAP Algoritmasının CUDA İle Uygulanması

PSA-QAP'nin CUDA üzerinde gerçekleştirilmesi aşamasında veri setlerinin boyutları dikkate alınarak daha hızlı hesaplama yapılması amacıyla paylaşılan bellekten (*shared memory*) yararlanılmıştır. Her iş parçacığına özgü aşağıdaki değişkenler tanımlanmıştır:

- *permutation[size_of_dataset]* (İterasyon anındaki permütasyon dizisi),
- *best_cost* (O ana kadar bulunan en iyi sonucun değeri),
- *temperature* (SA için sıcaklık parametresi değeri),
- *delta_cost* (QAP'a ait delta değeri).

İş parçacıkları kendi başlangıç değerleri ile SA algoritmasını seri olarak yürütmektedirler. Komşuluk arama yöntemi olarak ise klasik ikili yer değiştirme (swapping) kullanılmıştır. İki maliyet arasındaki farkı bulmak için Burkard ve Rendl tarafından sunulan delta fonksiyonu kullanılmıştır [49]. π dizisine ait r ve s dizinindeki yer değiştirme maliyeti $\Delta(\pi, r, s)$ olarak tanımlanacak olursa fonksiyon (5.2)'deki gibi ifade edilir. (*O(n)* karmaşıklıkta hesaplama yapmaktadır.)

Geliştirilen yöntemde blok-içi bir haberleşme yapıldığından iş parçacıkları arasındaki en iyi sonuca ait permütasyonun tutulduğu veri seti boyutlarına eşit bir permütasyon dizisine (*shared_permutation_array[])* ihtiyaç duyulmuştur. Bu dizi *__shared__* olarak tanımlanmıştır. Bir blokta kullanılan iş parçacığı sayısı kullanılan grafik kartında üst limit olan 1024 olarak belirlenmiştir. Bu yöntem için *__shared__* olarak tanımlanan değişkenler şunlardır:

- *shared_cost[number_of_threads_in_a_block]*, (İş parçacıklarına ait maliyet değerleri),
- *thread_id_in_a_block[number_of_threads_in_a_block]* (İş parçacıklarının dizin numaraları),

- *shared_permutation_array[size_of_dataset]* (En iyi permütasyona ait dizi).

$$\begin{aligned}\Delta(\pi,r,s) = {} & d_{rr}\left(f_{\pi(s)\pi(s)} - f_{\pi(r)\pi(r)}\right) + \\ & d_{rs}\left(f_{\pi(s)\pi(r)} - f_{\pi(r)\pi(s)}\right) + \\ & d_{sr}\left(f_{\pi(r)\pi(s)} - f_{\pi(s)\pi(r)}\right) + \\ & d_{ss}\left(f_{\pi(r)\pi(r)} - f_{\pi(s)\pi(s)}\right) + \\ & \sum_{k=0,\ k\neq r,s}^{n-1}\left(\begin{array}{l} d_{kr}\left(f_{\pi(k)\pi(s)} - f_{\pi(k)\pi(r)}\right) + \\ d_{ks}\left(f_{\pi(k)\pi(r)} - f_{\pi(k)\pi(s)}\right) + \\ d_{rk}\left(f_{\pi(s)\pi(k)} - f_{\pi(r)\pi(k)}\right) + \\ d_{sk}\left(f_{\pi(r)\pi(k)} - f_{\pi(s)\pi(k)}\right) \end{array}\right)\end{aligned} \tag{5.2}$$

İndirgeme yöntemi *cost[]* dizisi üzerinde gerçekleştirilerek işlem sonunda *cost[0]* dizininde en iyi sonuç elde edilir. Bu değerin hangi iş parçacığına ait olduğunu belirlemek amacıyla *thread_id_in_a_block[]* isimli bir değişken kullanılmıştır. Bu iş parçacığı daha sonra kendisine ait permütasyon dizisini *shared_permutation_array[]* aracılığıyla tüm iş parçacıklarıyla paylaşmaktadır.

Bloklar-arası haberleşmeyi sağlamak amacıyla diğer değişkenlerin dışında global değişkenlere ihtiyaç duyulmaktadır. İş parçacıkları blok-içi en iyi sonucu bulduktan sonra bu sonucu global bir değişken dizisine (*global_cost[]*) atarlar. Görevlendirilen bir iş parçacığı bu dizide bulunan sonuçlar arasından en iyi sonucu bulur. Bu işlem yapılırken diğer iş parçacıkları bloklar-arası haberleşme için kullanılan yöntem kullanılarak beklerler. Bulunan en iyi sonuca sahip iş parçacığı kendi permütasyon dizisini diğer iş parçacıklarıyla paylaşmak için global bir diziye atar. Yine bu işlem yapılırken diğer iş parçacıkları beklerler. Sonraki iterasyonda kullanılmak üzere diğer iş parçacıkları bu permütasyonu ilgili diziden alır ve işlemlerine devam ederler. Bu yöntem için global olarak tanımlanan değişkenler şunlardır:

- *global_cost[number_of_blocks]*, (İş parçacıklarına ait maliyet değerleri),

- *thread_id[number_of_blocks]* (İş parçacıklarının dizin numaraları),
- global_permutation_array[dataset_size] (En iyi permütasyona ait dizi),
- *ain[number_of_blocks]* (Bloklar-arası haberleşme için [48]),
- *aout[number_of_blocks]* (Bloklar-arası haberleşme için [48]).

5.1.5. Deneysel Sonuçlar

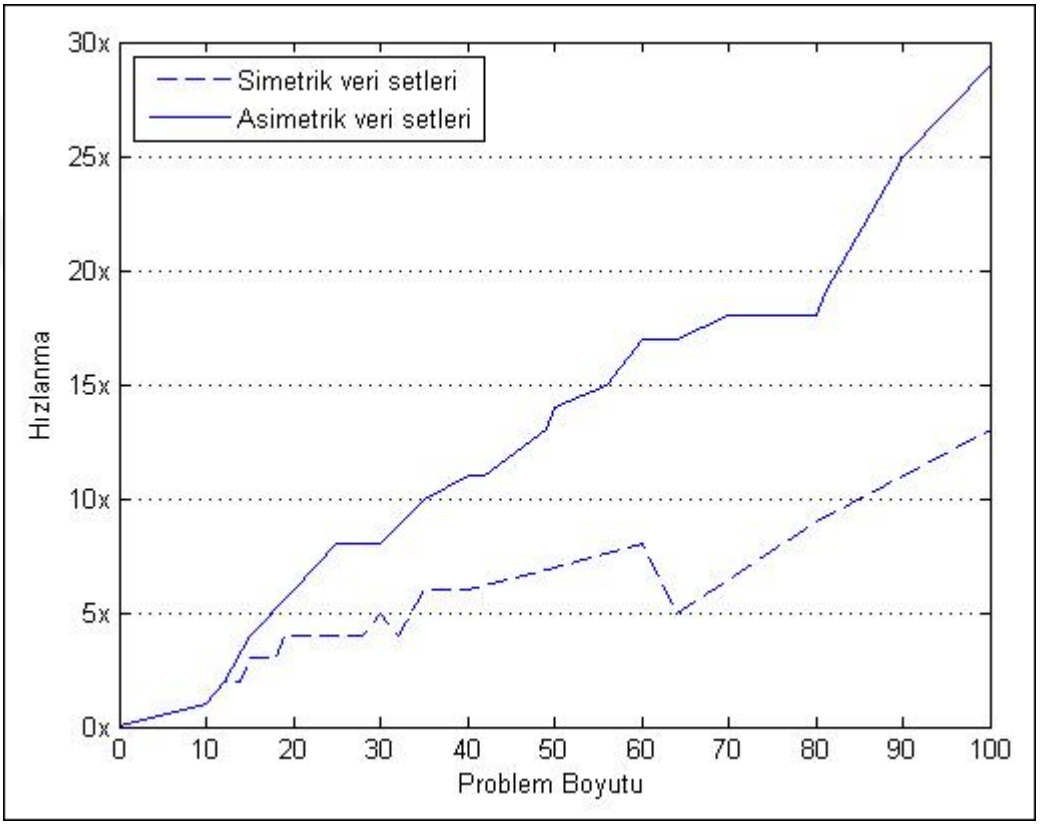
Geliştirilen yöntem QAPLIB [41] kütüphanesine ait 132 adet veri seti üzerinde çalıştırılıp test edilmiştir. Kütüphanede bulunan veri setleri simetrik ve asimetrik olmak üzere iki sınıfa ayrılmaktadır. Çalışmada kullanılan veri setleri 10 ile 100 boyutları arasında değişmektedir.

Deneysel sonuçlar Intel(R) Xeon(R) CPU E5-2630 v3 @2.40 GHz işlemci üzerinde Windows 8.1 İşletim Sisteminde NVIDIA Geforce GTX Titan X GPU (12 GB Memory, 24 Multiprocessors) ile test edilmiştir. Tüm yöntemler C++ programlama dili ile gerçekleştirilmiştir.

Bilindiği üzere SA algoritmasının performansı, büyük ölçüde yönteme ait parametrelere bağlıdır. Bu bağlamda uygun parametre değerlerinin seçilmesi amacıyla çalışmalar gerçekleştirilmiştir. Parametre seçimlerinde veri seti boyutları dikkate alınarak iki farklı parametre konfigürasyonu belirlenmiştir. Veri seti boyutları 0 ile 39 arasında olan veri setleri için parametre değerleri T_0 = 1000, T_{min} = 0.1, α = 0.999, 40 ile 100 boyutları arasında olan veri setleri için parametre değerleri T_0 = 10000, T_{min} = 0.1, α = 0.99999 olarak belirlenmiştir. Her iki parametre konfigürasyonuna göre yöntemler bir ve beş olmak üzere iki farklı iterasyon sayısı ile değerlendirilmiştir. Algoritmanın bir kez çalıştırılmasında iş parçacıkları arasında herhangi bir işbirliği gerçekleştirilmemiştir (PSA-QAP I). Beş kez çalıştırılan yöntemlerde ise iş parçacıkları arasında iterasyon sayısı kadar haberleşme gerçekleştirilerek işbirliği sağlanmıştır (PSA-QAP II). İterasyon sayısının artırılması sonuçlar üzerinde ek bir katkı yapmamakla birlikte çalışma sürelerini uzattığından beşin üzerinde bir iterasyon sayısı tercih edilmemiştir.

CPU ve GPU üzerinde geliştirilen yöntemlerin arasındaki hız farkını değerlendirmek amacıyla hızlanma (speedup) kriteri kullanılmıştır. Hızlanma aşağıdaki gibi ifade edilmektedir:

$$hızlanma = \frac{çalışma\ süresi_{CPU}}{çalışma\ süresi_{GPU}} \tag{5.3}$$



Şekil 5.4. Farklı boyutlardaki veri setlerine ait hızlanma grafiği.

Elde edilen sonuçlara göre simetrik veri setlerinde hızlanmanın 13x'e kadar, asimetrik veri setlerinde ise 29x'e kadar çıktığı görülmüştür (Şekil 5.4). Veri setlerinin boyutları arttıkça hızlanmanın da arttığı sonucuna varılmıştır. Gerek simetrik gerekse asimetrik veri setlerinde en yüksek hızlanma değeri 100 boyutlu veri setlerinde gerçekleşmiştir. PSA-QAP I ve PSA-QAP II yöntemleri değerlendirildiğinde iki farklı iterasyon sonucuna göre hızlanma değerlerinin birbirine yakın değerler olduğu gözlemlenmiştir.

Değerlendirme yapılırken karşılaştırma kriteri olarak Bağıl Yüzde Sapma (Relative Percentage Deviation, RPD) kullanılmıştır. RPD aşağıdaki formül ile ifade edilmektedir:

$$RPD = \frac{cost(\pi) - cost(\pi^*)}{cost(\pi^*)} \, x \, 100\% \tag{5.4}$$

$cost(\pi)$ geliştirilen yöntem ile bulunan sonucu, $cost(\pi^*)$ ise Bilinen En İyi Sonucu (Best Known Solution, BKS) temsil etmektedir.

PSA-QAP, 83 adet simetrik veri setinden 73 tanesinde, 49 adet asimetrik veri setinden 40 tanesinde BKS'yi elde etmiştir. BKS bulunan 113 veri setinden 99'u bir dakikanın altında çalışma süresi elde edilmiştir. Ortalamada, PSA-QAP I algoritması %0.09, PSA-QAP II algoritması %0.07 BKS'den daha kötüdür.

Çizelge 5.1. Sko* veri setleri için PSA-QAP algoritmasının sonuçları.

Veri Seti	**BKS**	**PSA-QAP I**		**PSA-QAP II**	
		RPD (%)	**Süre (s)**	**RPD (%)**	**Süre (s)**
sko42	15,812	0.000	61.3	0.000	306.4
sko49	23,386	0.000	71.2	0.000	357.5
sko56	34,458	0.000	80.6	0.000	406.2
sko64	48,498	0.000	123.1	0.000	618.0
sko72	66,256	0.006	103.1	0.000	516.3
sko81	90,998	0.011	111.0	0.000	555.8
sko90	115,534	0.042	122.6	0.029	613.5
sko100a	152,002	0.022	136.7	0.000	680.8
sko100b	153,890	0.000	137.0	0.000	680.1
sko100c	147,862	0.008	136.0	0.003	682.1
sko100d	149,576	0.027	135.8	0.013	682.6
sko100e	149,150	0.020	135.4	0.009	679.9
sko100f	149,036	0.043	136.2	0.038	682.0
Ortalama		**0.014**	**112.9**	**0.007**	**574.0**

Sko* ve Tai*b (asimetrik) veri setlerine ait sonuçlar Çizelge 5.1 ve Çizelge 5.2'de yer almaktadır. İterasyon sayısı arttıkça çalışma süresi da artmakta ve sonuçların

iyileştiği görülmektedir. Çalışmada kullanılan tüm veri setlerine ait sonuçlar "Ek Açıklamalar A" bölümünde verilmiştir.

Çizelge 5.2. Tai*b veri setleri için PSA-QAP algoritmasının sonuçları.

Veri Seti	BKS	PSA-QAP I		PSA-QAP II	
		RPD (%)	Süre (s)	RPD (%)	Süre (s)
tai10b	1,183,760	0.000	0.1	0.000	0.5
tai12b	39,464,925	0.000	0.1	0.000	0.4
tai15b	51,765,268	0.000	0.1	0.000	0.4
tai20b	122,455,319	0.000	0.1	0.000	0.5
tai25b	344,355,646	0.000	0.1	0.000	0.6
tai30b	637,117,113	0.000	0.2	0.000	0.8
tai35b	283,315,445	0.000	0.2	0.000	0.9
tai40b	637,250,948	0.000	32.2	0.000	167.1
tai50b	458,821,517	0.050	39.5	0.000	187.9
tai60b	608,215,054	0.107	48.9	0.000	240.8
tai80b	818,415,043	0.244	86.9	0.098	484.8
tai100b	1,185,996,137	0.436	82.7	0.382	413.1
Ortalama		**0.070**	**24.3**	**0.040**	**124.8**

Elde edilen sonuçlara bakıldığında PSA-QAP algoritmasının kısa süre içerisinde birçok veri setinde bilinen en iyi sonucu ya da en iyi sonuca yakın bir değeri elde ettiğini söylemek mümkündür.

5.2. SIRT ÇANTASI PROBLEMİ (KP)

Sırt Çantası Problemi (Knapsack Problem, KP) bilinen en eski kombinatoryal optimizasyon problemlerinden birisidir. 0/1 KP, çok-boyutlu KP, çoklu KP ve KP paylaşım problemi gibi birçok çeşidi mevcuttur. Bu çalışma, 0/1 KP üzerinde uygulanmıştır. Problemin amacı, verilen bir nesne kümesinin içerisinden sırt çantasının kapasitesini aşmayacak şekilde maksimum fayda sağlayacak (en değerli) nesneleri seçip çantaya yerleştirmektir. 0/1 KP problemi aşağıdaki gibi formülize edilmektedir:

$$max \sum_{i=1}^{n} p_i x_i \,, \tag{5.5}$$

$$s.t. \sum_{i=1}^{n} w_i x_i \leq C, \tag{5.6}$$

$$x_i \in \{0,1\}, i \in \{1, \dots, n\} \tag{5.7}$$

p_i: *i. nesnenin değeri, getirdiği fayda (profit).*

w_i: *i. nesnenin ağırlığı (weight).*

x_i: *i. nesnenin sırt çantasında bulunma durumu.*

n: Toplam nesne sayısı,

C: Bir sırt çantasının kapasitesi.

0/1 KP NP-zor bir kombinatoryal optimizasyon problemidir. Literatürde KP'nin çözümü için birkaç kesin çözüm yöntemi önerilmiştir. Genellikle dinamik programlama ile dallanma ve sınırlandırma [50, 51] çözümleri tercih edilmiştir. Diğer yöntemler, makul bir zamanda optimum veya optimum çözüme yakın çözümleri sağlayan sezgisel yöntemlerdir. SA [52], Karınca Koloni Algoritması [53], Genetik Algoritma [54], Tabu Arama Algoritması [55] vb. sezgisel yöntemler [56–59] 0/1 KP için önerilmiştir. Bazı kesin çözüm yöntemleri CPU [60, 61] ve GPU [62] üzerinde paralelleştirilmiştir. Öte yandan, KP için paralel sezgisel yöntemler literatürde sınırlıdır [63–65]. GPU'lar KP gibi kombinatoryal optimizasyon problemini çözmek için güçlü bir mimariye sahiptirler. Bu amaçla 0/1 KP'yi çözmek için SA algoritmasını kullanan bir yöntem geliştirilmiştir.

5.2.1. 0/1 KP'nin Çözümü İçin Sunulan Paralel SA Algoritması (PSA-KP)

0/1 KP probleminin çözümü için heterojen bir yöntem geliştirilmiştir. Öncelikle probleme aç gözlü bir yaklaşım sağlanarak çözüme gidilmesi amaçlanmıştır. Bu kısım CPU üzerinde seri bir şekilde gerçekleştirilmiştir. Daha sonra ise GPU

üzerinde paralel SA algoritması ile problemin çözümü iyileştirilmeye çalışılmıştır. Önerilen yöntemin adımları aşağıdaki gibidir:

1. Verilen bir sırt çantası problemindeki her nesnenin yoğunluğunu ($d = p_i / w_i$) hesapla ve nesneleri yoğunluklarına göre büyükten küçüğe doğru sırala.
2. Sırt çantasının kapasitesini aşmayacak şekilde nesneleri sıra ile çantaya yerleştir.
3. SA parametreleri (T, T_{target}, α), ve diğer değişkenler (*Total_Capacity*, *Total_Profit* and *Total_Profit_Best)* için gerekli düzenlemeleri gerçekleştir.
4. Sırt çantasında bulunmayan bir *X* nesnesini rastgele seç.
5. *X* nesnesini çantaya yerleştir. *Total_Capacity* çantanın kapasitesini aşmıyor ise bir sonraki adıma geç, aksi taktirde Adım 10'a git.
6. Sırt çantasında bulunmayan bir *Y* nesnesini rastgele seç. *X* nesnesini çantadan çıkar ve yerine *Y* nesnesini çantaya yerleştir. *Total_Capacity* çantanın kapasitesini aşmıyor ise bir sonraki adıma geç, aksi taktirde Adım 9'a git.
7. *X* nesnesi ile *Y* nesnesi arasındaki değer farkını hesapla ($\Delta P = P_x - P_y$).
8. ΔP sıfırdan küçük ise ya da metropolis kriterine göre hesaplanan olasılık değeri rastgele belirlenen bir sayıdan (0 ve 1 arası) büyük ise *X* nesnesi yerine *Y* nesnesini seç, çantaya yerleştir ve P_y değerini *Total_Profit* değerine ekle. Aksi takdirde bir sonraki adıma geç.
9. *X* nesnesini seçerek çantaya yerleştir. P_x değerini *Total_Profit* değerine ekle. Adım 15'e git.
10. Sırt çantasında bulunan bir *Z* nesnesini rastgele seç.
11. *Z* nesnesini çantadan çıkar. *Total_Capacity* çantanın kapasitesini aşmıyor ise bir sonraki adıma geç, aksi taktirde Adım 14'e git.
12. *Z* nesnesi ile *X* nesnesi arasındaki değer farkını hesapla ($\Delta P = P_z - P_x$).
13. ΔP sıfırdan küçük ise ya da metropolis kriterine göre hesaplanan olasılık değeri rastgele belirlenen bir sayıdan (0 ve 1 arası) büyük ise *X* nesnesini seç ve P_x değerini *Total_Profit* değerine ekle. Aksi takdirde bir sonraki adıma geç.
14. *X* nesnesi yerine *Z* nesnesini seç, tekrar çantaya yerleştir ve P_z değerini *Total_Profit* değerine ekle.
15. *Total_Profit* değeri *Total_Profit_Best* değerinden büyük ise yeni en iyi değer olarak *Total_Profit_Best* değerine ata.
16. Soğutma katsayısını (α) kullanarak *T* değerini azalt.

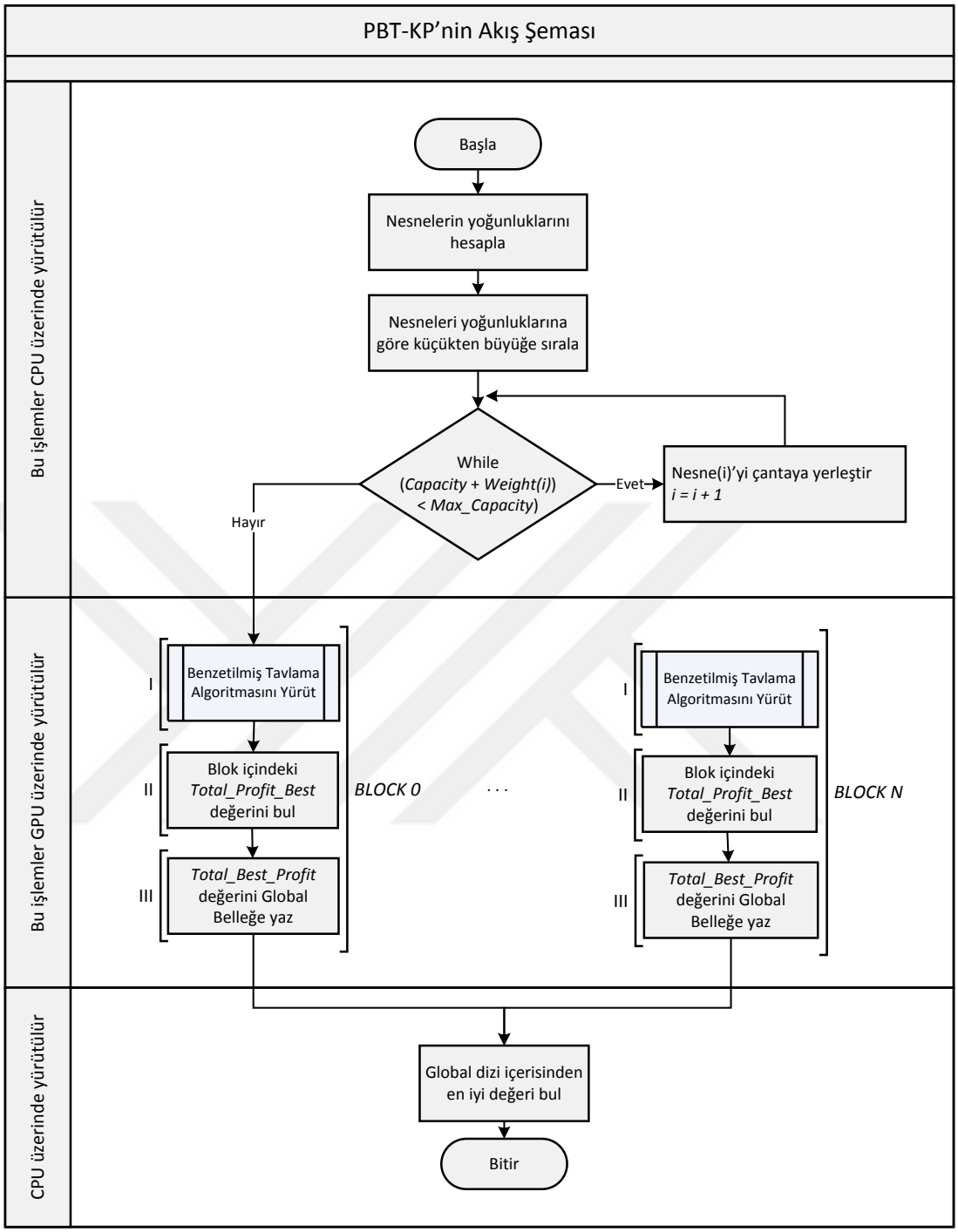
17. Sıcaklık değeri T_{target} değerine ulaşmadıysa Adım 4'e git, ulaştı ise bir sonraki adım ile devam et.
18. *Total_Profit_Best* değerini en iyi değer olarak sonuçlandır ve yöntemi sonlandır.

5.2.2. PSA-KP Algoritmasının CUDA İle Uygulanması

Önceki bölümde sunulan yöntemin üçüncü adımından itibaren gerçekleştirilen adımlar GPU üzerinde paralel bir şekilde yürütülmektedir. Belirli bir kapasiteye sahip sırt çantası için en iyi sonucu elde etmek amacıyla GPU'daki her iş parçacığı farklı bir aday nesne ile yönteme başlarlar. İş parçacıklarının farklı sıralı dizilerde rastgele sayılar üretmesini sağlamak için CUDA platformunda bir kütüphane (CUDA Random Number Generation Library, CuRAND) geliştirilmiştir. Her bir iş parçacığı SA için verilen parametrelerle bağımsız olarak önerilen yöntemi gerçekleştirir. Bu sürecin sonunda, tüm blokların her bir iş parçacığı, paylaşılan bellek kullanarak blok-içi iletişim kurar. Her bir blok içerisinde en iyi değer, indirgeme yöntemi kullanılarak paralel bir şekilde bulunur. Paralel yaklaşımlı önerilen algoritmanın akış şeması Şekil 5.5'te gösterilmiştir. İş parçacıklarının aktif ya da inaktif olduğu durumlar vardır. Bu üç durum Şekil 5.5'te görüldüğü üzere şu şekildedir:

I. Blok içerisindeki tüm iş parçacıkları aktiftir.
II. Blok içerisindeki iş parçacıklarının ilk yarısı aktiftir.
III. Her bir bloğun ilk iş parcacığı aktiftir.

GPU üzerinde gerçekleştirilen sürecin sonrasında, CPU üzerinde her bir bloğunun en iyi değerleri arasından da en iyi değer elde edilerek yöntem sonlandırılır. Çalışmada toplamda 1024 blok kullanılmış ve her bir blokta 1024 iş parçacığı kullanılmıştır. Toplam iş parçacığı sayısı 1024x1024'tür.



Şekil 5.5. PSA-KP algoritmasının akış şeması.

5.2.3. Deneysel Sonuçlar

Geliştirilen yöntem problem örnekleri üzerinde test edilmiştir. Standart bilinen on adet problem örneği (KP1-KP10) Çizelge 5.3'te gösterilmiştir. Bunun haricinde daha

büyük boyutlu problem örnekleri üzerinde test etmek amacıyla altı adet problem örneği rastgele oluşturulmuştur (KP11-KP16).

Çizelge 5.3. 0/1 KP için standart problem örnekleri.

Problem	**n**	**C**	**Optimum**	**Ağırlık *w* & Değer *p***
KP1	10	269	295	***w*** = {95, 4, 60, 32, 23, 72, 80, 62, 65, 46} ***p*** = {55, 10, 47, 5, 4, 50, 8, 61, 85, 87}
KP2	20	878	1024	***w*** = {92, 4, 43, 83, 84, 68, 92, 82, 6, 44, 32, 18, 56,83, 25, 96, 70, 48, 14, 58} ***p*** = {44, 46,90, 72, 91, 40, 75, 35, 8, 54, 78, 40, 77, 15, 61, 17,75, 29, 75, 63}
KP3	4	20	35	***w*** = {6, 5, 9, 7}, ***p*** = {9, 11, 13, 15}
KP4	4	11	23	***w*** = {2, 4, 6, 7}, ***p*** = {6, 10, 12, 13}
KP5	15	375	481.0694	***w*** = {56.358531, 80.874050, 47.987304, 89.596240, 74.660482, 85.894345, 51.353496,1.498459, 36.445204, 16.589862, 44.569231, 0.466933, 37.788018, 57.118442, 60.716575}, ***p*** = {0.125126, 19.330424, 58.500931, 35.029145, 82.284005, 17.410810, 71.050142, 30.399487, 9.140294, 14.731285, 98.852504, 11.908322, 0.891140, 53.166295, 60.176397}
KP6	10	60	52	***w*** = {30, 25, 20, 18, 17, 11, 5, 2, 1, 1}, ***p*** = {20, 18, 17, 15, 15, 10, 5, 3, 1, 1}
KP7	7	50	107	***w*** = {31, 10, 20, 19, 4, 3, 6}, ***p*** = {70, 20,39, 37, 7, 5, 10}
KP8	23	10,000	9767	***w*** = {983, 982, 981, 980, 979, 978, 488, 976, 972, 486, 486, 972, 972, 485, 485, 969, 966, 483, 964, 963, 961, 958, 959}, ***p*** = {981, 980, 979, 978, 977, 976, 487, 974, 970, 485, 485, 970, 970, 484, 484, 976, 974, 482, 962, 961, 959, 958, 857}
KP9	5	80	130	***w*** = {15, 20, 17, 8, 31}, ***p*** = {33, 24, 36,37, 12}
KP10	20	879	1025	***w*** = {84, 83, 43, 4, 44, 6, 82, 92, 25, 83, 56, 18, 58, 14, 48, 70, 96, 32, 68, 92}, ***p*** = {91, 72, 90, 46, 55, 8, 35, 75, 61, 15, 77, 40, 63, 75, 29, 75, 17, 78, 40, 44}

PSA-KP, NVIDIA GeForce GTX Titan X grafik kartı (3072 çekirdek, 1.0 GHz) üzerinde yürütülmüştür. Çözüm kalitesi ve hesaplama süresi, CPU üzerinde Intel Xeon 2.4 GHz ile elde edilen seri yöntem ile karşılaştırılmıştır. Tüm yöntemler C++ programlama dili ile geliştirilmiştir.

SA algoritmasının performansı önceki bölümlerde söylendiği üzere parametrelere bağlıdır. Parametre konfigürasyonları $T = 1000$, $T_{target} = 1$, $\alpha = 0.95$ olarak belirlenmiştir. Her bir problem için sonuçlar ve çalışma süreleri CPU üzerinde seri SA algoritmasının on kez bağımsız şekilde çalıştırılıp ortalaması alınarak elde edilmiştir. Standart problem örnekleri üzerinde yapılan testlere ait sonuçlar Çizelge 5.4'te sunulmuştur. Tüm problem örnekleri için en iyi, en kötü, ortalama, medyan ve standart sapma (SD) değerleri Çizelge 5.4'te listelenmiştir. SA algoritması bu on problem örneği için en iyi çözümü elde etmiştir.

Çizelge 5.4. KP1-KP10 için seri SA algoritmasının sonuçları.

Problem	**En İyi**	**En Kötü**	**Ortalama**	**Medyan**	**SD**
KP1	295	295	295	295	0.00
KP2	1024	1024	1024	1024	0.00
KP3	35	35	35	35	0.00
KP4	23	23	23	23	0.00
KP5	481.0694	437.9345	472.4424	481.0694	17.25
KP6	52	52	52	52	0.00
KP7	107	81	100.5	105	7.53
KP8	9767	9754	9755.4	9758	4.22
KP9	130	130	130	130	0.00
KP10	1025	1025	1025	1025	0.00

Diğer problem örnekleri için *n*, 100, 200, 300, 500, 800 ve 1000 olarak belirlenmiştir. Her bir nesne için ağırlık (5-20 arasında) ve değer (50-100 arasında) rastgele oluşturulmuştur. Sırt çantalarının kapasiteleri sırasıyla 1100, 1500, 1700, 2000, 5000 ve 10000 olarak belirlenmiştir. On bağımsız çalışmanın sonuçları Çizelge 5.5'te sunulmuştur. En iyi sonuçlar kalın yazı tipi ile gösterilmiştir.

Çizelge 5.5. KP10-KP16 için seri SA algoritmasının sonuçları.

Problem	**n**	**En İyi**	**En Kötü**	**Ortalama**	**Medyan**	**SD**
KP11	100	**6673**	6647	6666.6	6673	9.83
KP12	200	**11708**	11660	11684.9	11684	18.39
KP13	300	**13658**	13622	13652.4	13658	11.76
KP14	500	**18621**	18594	18612.1	18613	6.86
KP15	800	**40111**	40069	40106.8	40111	12.60
KP16	1000	**66249**	66213	66241.0	66249	12.27

PSA-KP paralel olarak GPU üzerinde yalnızca bir kez çalıştırılmıştır. Çizelge 5.6, PSA-KP algoritmasının tüm problem örnekleri üzerinde yapılan testlerin sonuçlarını göstermektedir. Çizelge 5.6 sonuçların karşılaştırılması için seri algoritmanın en iyi ve ortalama değerlerini de göstermektedir. En iyi sonuçlar kalın yazı tipi olarak gösterilmiştir. PSA-KP algoritmasının seri algoritmaya nazaran daha iyi sonuçlar elde ettiği açıktır.

Çizelge 5.6. KP1-KP16 için seri SA ve PSA-KP algoritmasının sonuçları.

Problem	**PSA-KP**	**Seri En İyi**	**Seri Ortalama**
KP1	**295**	**295**	**295**
KP2	**1024**	**1024**	**1024**
KP3	**35**	**35**	**35**
KP4	**23**	**23**	**23**
KP5	**481.0694**	**481.0694**	472.4424
KP6	**52**	**52**	**52**
KP7	**107**	**107**	100.5
KP8	**9767**	**9767**	9755.4
KP9	**130**	**130**	**130**
KP10	**1025**	**1025**	**1025**
KP11	**6711**	6673	6666.6
KP12	**11738**	11708	11684.9
KP13	**13668**	13658	13652.4
KP14	**18644**	18621	18612.1
KP15	**40121**	40111	40106.8
KP16	**66270**	66249	66241.0

Her problem örneği için çalışma süresinin sonuçları Çizelge 5.7'de (Seri Algoritmanın Çalışma Süresi-S.R., PSA-KP Algoritması'nın Çalışma Süresi-P.R) verilmiştir ve PSA-KP algoritmasına ait sonuçlar, GPU üzerinde on bağımsız çalışmanın ortalaması ile elde edilmiştir. KP1-KP10 için seri algoritmanın çalışma süresi sonuçlarına göre, çalışma süreleri problem örneklerinde birbirine yakındır ve ortalama süre 9.13 saniyedir. KP11-KP16 için ise ortalama süre 6.25 saniyedir.

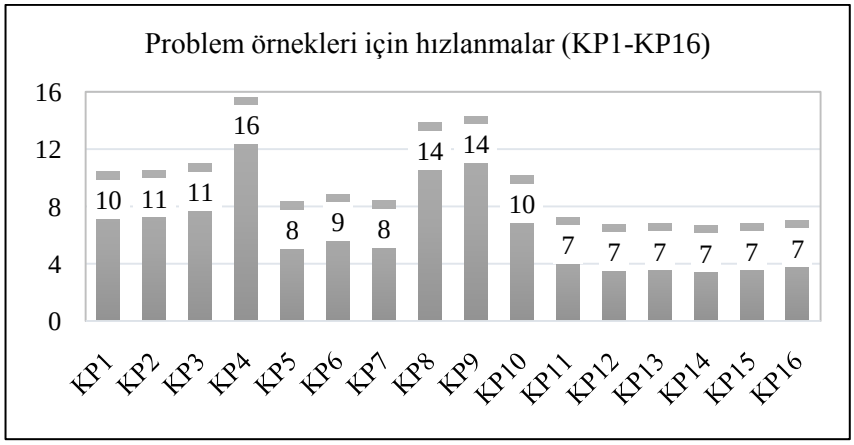
KP1-KP10 problem örneklerinin çalışma sürelerinin ortalaması, KP11-KP16 problem örneklerinin ortalamasından daha yüksektir. Bunun nedeni, rastgele sayıların eşit olma ihtimalinin küçük boyutlu örnekler için yüksek olması ve global belleğe erişimden kaynaklanan gecikmeler olarak düşünülmektedir.

Çizelge 5.7. KP1-KP16 için seri SA ve PSA-KP algoritmasının çalışma süreleri.

Problem	**S.R. (s)**	**P.R. (s)**	**Problem**	**S.R. (s)**	**P.R. (s)**
KP1	9.03	0.87	KP11	6.61	0.91
KP2	8.81	0.84	KP12	6.08	0.90
KP3	9.13	0.83	KP13	6.12	0.90
KP4	12.40	0.79	KP14	5.93	0.89
KP5	6.84	0.82	KP15	6.28	0.92
KP6	7.06	0.80	KP16	6.46	0.92
KP7	6.74	0.80			
KP8	11.36	0.82			
KP9	11.09	0.77			
KP10	8.82	0.87			
Ortalama	**9.13**	**0.82**	**Ortalama**	**6.25**	**0.91**

Yöntemler arasındaki hız farkını değerlendirmek amacıyla hızlanma kriteri kullanılmıştır. Hızlanma aşağıdaki gibi ifade edilmektedir:

$$hızlanma = \frac{çalışma\ süresi_{SERİ}}{çalışma\ süresi_{PBT-KP}} \tag{5.8}$$



Şekil 5.6. KP1-KP16 problem örnekleri için hızlanma değerleri.

Hızlanma sonuçları Şekil 5.6'da gösterilmiştir. 16x'e kadar hızlanma elde edilmiştir. KP1-KP10 problem örnekleri için ortalama hızlanma 11x ve KP10-KP16 problem örnekleri için ortalama hızlanma 7x'dir.

Elde edilen sonuçlara bakıldığında PSA-KP algoritmasının kısa süre içerisinde küçük ve büyük boyutlu örnekler için kaliteli sonuçlar verdiği söylenebilmektedir.

5.3. SİLAH-HEDEF ATAMA PROBLEMİ (WTA)

Silah-Hedef Atama (Weapon-Target Assignment, WTA) problemi, askeri operasyon araştırması alanında NP-tam olan bir optimizasyon problemidir [66]. WTA problemi, hayatta kalma şansını artırmak için savunulan alanın beklenen hasarını en aza indirgemek için silahların hedeflere en iyi şekilde atanmasını hedeflemektedir.

WTA probleminde, savunma gücü varlıkları hedefin yönlendirdiği saldırıları yok etmeyi amaçlamaktadır. Savunma, gelen saldırıları bertaraf etmek için sınırlı sayıda silaha sahiptir. WTA probleminde statik ve dinamik olarak iki model tipi vardır. Bu çalışmada, WTA probleminin statik modeli kullanılmıştır. Statik modelde problemin tüm girdileri sabittir ve silahların hedeflere atanması tek bir adımda gerçekleştirilir. Atamalar için öngörülen hasar, tüm silah-hedef atamaları tamamlandıktan sonra değerlendirilir. Problemin parametreleri ve değişkenleri aşağıdaki gibi tanımlanmıştır:

- n , hedeflerin sayısı (1, 2, …, n),
- m , silahların sayısı (1, 2, …, m),
- v_i , i numaralı hedefin değeri,
- p_{ij} , i numaralı hedefe atanan j numaralı silahın hedefi yok etme ihtimali,
- $x = [x_{ij}]$, nxm boyutlu karar verme değişkeni, öyleki;

$$x_{ij} = \begin{cases} j\ numaralı\ silah\ i\ numaralı\ hedef\ atanmış\ ise\ 1, \\ aksi\ taktirde\ 0. \end{cases} \tag{5.9}$$

Bir i numaralı hedefin j numaralı silah tarafından saldırıya uğradığındaki hayatta kalma olasılığı $(1 - p_{ij})^{x_{ij}}$ olarak ifade edilir. Problem aşağıdaki gibi formulize edilmektedir :

$$f(\pi) = \sum_{i=1}^{n} v_i \prod_{j=1}^{m} (1 - p_{ij})^{x_{ij}} \tag{5.10}$$

$$s.t. \sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1,2,\dots,m \tag{5.11}$$

Problemdeki tüm silahların hedeflere atanması gerekmektedir. Yapılan çalışmada, eşit sayıda silah ve hedefin olduğu senaryolar oluşturulmuştur ve her bir silahın bir hedefe atandığı varsayılarak çözümün elde edilmesi amaçlanmıştır.

Literatürde WTA problemi için birkaç kesin çözüm yöntemi sunulmuştur [67–69] ancak bu yöntemler yalnızca küçük boyutlu problemleri çözebilecek yeterliliktedir. Bundan dolayı WTA probleminin çözümünde Benzetilmiş Tavlama [70, 71], Genetik Algoritma [71, 72], Tabu Arama [71], Değişken Komşuluk Arama [68, 71], Karınca Kolonisi [72–74] ve Parçacık Sürüsü Optimizasyonu [75] gibi sezgisel yöntemler önerilmiştir.

5.3.1. WTA Probleminin Çözümü İçin Sunulan Paralel SA Algoritması (PSA-WTA)

Geliştirilen yöntemde yeni bir komşu çözüm, rastgele seçilen iki silahın (q ve r olsun) hedef atamalarındaki yer değiştirmeleri (swapping) ile elde edilir. Δf iki komşu çözüm maliyeti arasındaki farktır ve aşağıdaki gibi ifade edilir:

$$\Delta f = v_q(p_{qq} - p_{rq}) + v_r(p_{rr} - p_{qr}) \tag{5.12}$$

Yer değiştirme işlemi gerçekleştirildikten sonra, yeni aday çözüm aşağıdaki formül ile bulunur:

$$f(\pi') = f(\pi) + \Delta f \tag{5.13}$$

Önerilen yöntemin ana adımları şöyledir:

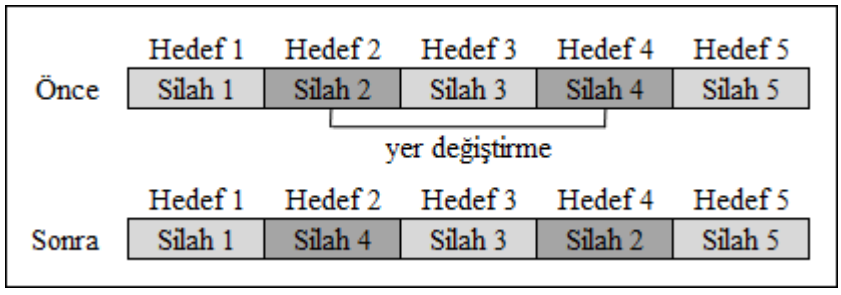
Aşama 1: Düzenlemeler

1) Girdiler: Yok etme olasılık matrisi p, hedeflerin değerleri v, her bir silahın bir hedefe atandığı permütasyon dizisi s.
2) SA parametlerini ayarla: T,T_{final} and a.
3) Rastgele oluşturulan permütasyon dizisi s ile (5.10)'u kullanarak WTA problem için ilk çözümü bul.

Aşama 2: SA Algoritmasının Yürütülmesi

4) Rastgele iki farklı dizin numarası üret ve yer değiştirmeyi uygula.
5) Δf'i (5.12)'yi kullanarak hesapla.
6) Aday çözümün kabul edilmesi koşullarını uygula.
7) Aday çözüm kabul edildi ise permütasyon dizisini güncelle ve bir sonraki adım ile devam et, aksi taktirde Adım 9'a git.
8) Yeni maliyeti (5.13)'ü kullanarak hesapla.
9) Yeni çözüm daha iyi ise en iyi çözüm olarak gerekli atamaları gerçekleştir.
10) Soğutma katsayısını (α) kullanarak T değerini azalt.
11) T değeri T_{final} değerine ulaşmadıysa Adım 4'git, aksi taktirde işlemi sonlandır.

Yukarıdaki aşamalarda, ikinci aşama SA algoritmasını gerçekleştirmektedir. Yeni bir çözüm bulunurken silah-hedef atamasındaki yer değiştirme için kullanılan teknik Şekil 5.7'de gösterilmiştir.



Şekil 5.7. Silah-hedef ataması arasındaki yer değiştirme.

5.3.2. PSA-WTA Algoritmasının CUDA İle Uygulanması

PSA-WTA algoritmasında GPU'daki her iş parçacığı farklı bir *s* permütasyon dizisi ile işleme başlar. Bu işlem için CUDA platformunda tanımlı ve çoklu-başlangıç tekniğini sağlamak için kullanıma uygun, rastgele sayı üretmeye imkan sağlayan cuRAND kütüphanesi kullanılmıştır. Her bir iş parçacığı, önceki bölümde anlatılan Aşama 2'deki adımları bağımsız olarak gerçekleştirir. Her iş parçacığı bu aşamayı yürüttükten sonra, aynı bloğun iş parçacıkları paylaşılan bellek kullanarak birbirleriyle iletişim kurarlar. En iyi çözüm değeri, indirgeme yöntemi kullanılarak paralel yöntemle bulunur. Her bir bloğa ait en iyi değer bulunduktan sonra bu değerler global bellek kullanılarak CPU'ya aktarılır. PSA-KP algoritmasında olduğu gibi en iyi çözüm son olarak CPU üzerinde elde edilir.

Bu çalışmada, her bir blokta 1024 iş parçacığı kullanılmıştır. Blok sayısı arttıkça, PSA-WTA'nın çalışma süresi de artmaktadır. Bu artışın nedeni, birçok iş parçacığının aynı anda global belleğe erişmesinden kaynaklanmaktadır. PSA-WTA'nın çalışma süresini optimize etmek için birkaç yapılandırma gerçekleştirilmiştir. Bunlar:

- Veri tipi olarak uygun yerlerde *int* yerine *short* kullanılmıştır.
- *--use_fast_math* derleyici parametresi aktif edilerek matematiksel fonksiyonların hızlı bir şekilde çözümü amaçlanmıştır.
- CUDA platformunda global belleğe erişim yavaştır. Hedeflerin değerleri, daha hızlı veri okuma yapabilmek adına paylaşılan bellekte tutulmuştur. Paylaşılan belleğin limitinden dolayı iş parçacıklarının ortak erişmesi gereken diğer değişkenler için global bellek kullanılmıştır.

Her iş parçacığı yer değiştirme işlemini kendi permütasyon dizisi üzerinde gerçekleştirir. Bu bağlamda, her iş parçacığına özel aşağıdaki değişkenlere ihtiyaç vardır:

- Permütasyon dizisi,
- Her adımdaki yer değiştirme işlemi için rastgele oluşturulan dizin numarasının tutulduğu iki adet *short* tipinde değişken,
- Fonksiyona ait maliyet değeri,
- Delta değeri (Mevcut ve aday çözüm arasındaki maliyet farkı),
- Sıcaklık değeri (SA parametresi).

5.3.3. Deneysel Sonuçlar

Literatürde WTA problemi için herhangi bir problem örneği bulunmamaktadır. Bu nedenle, önerilen yöntemlerin performansını test etmek için çeşitli senaryolar oluşturulmuştur. Bu çalışmada, farklı boyutlarda 12 problem örneği üzerinde hesaplama testleri yapılmıştır. Hedeflerin değerleri, 25-100 aralığında rastgele üretilmiştir. Silah-hedef atamaları için hedefleri yok etme ihtimalleri, 0.60-0.90 aralığında yine rastgele olarak üretilmiştir. Problem örnekleri, 5-200 aralığındaki farklı silah-hedef sayılarını içerir. Problem örneklerinin boyutları (WTA1-WTA12) Çizelge 5.8'de gösterilmektedir.

Çizelge 5.8. WTA için oluşturulan problem örneklerinin boyutları.

Problem	Silah	Hedef
WTA1	5	5
WTA2	10	10
WTA3	20	20
WTA4	30	30
WTA5	40	40
WTA6	50	50
WTA7	60	60
WTA8	70	70
WTA9	80	80
WTA10	90	90
WTA11	100	100
WTA12	200	200

SA algoritmasının parametre konfigürasyonları $T = 1000$, $T_{final} = 0.1$, $\alpha = 0.99999$ olarak belirlenmiştir. Her problem için, CPU üzerinde on bağımsız çalıştırmanın ortalaması alınarak sonuçlar elde edilmiştir. Tüm testler CPU üzerinde Intel Xeon 2.4 GHz ile yapılmıştır. Problem örnekleri üzerinde yapılan seri algoritmanın sonuçları

Çizelge 5.9'da sunulmuştur. Tüm problem örnekleri için en iyi, en kötü, ortalama, medyan ve standart sapma (SD) değerleri sonuçları Çizelge 5.9'da listelenmiştir.

Çizelge 5.9. WTA1-WTA12 için seri SA algoritmasının sonuçları.

Problem	**En iyi**	**En Kötü**	**Ortalama**	**Medyan**	**SD**
WTA1	48.3640	48.3640	48.3640	48.3640	0.00
WTA2	96.3123	96.3123	96.3123	96.3123	0.00
WTA3	142.1070	142.1070	142.1070	142.1070	0.00
WTA4	248.0285	248.0285	248.0285	248.0285	0.00
WTA5	305.5016	305.5016	305.5016	305.5016	0.00
WTA6	353.0767	353.5702	353.3112	353.2610	0.14
WTA7	415.0528	415.7079	415.4068	415.4371	0.21
WTA8	498.1049	499.0167	498.5918	498.5860	0.30
WTA9	534.4408	536.2618	535.4559	535.5937	0.57
WTA10	594.0639	596.1228	595.3277	595.6466	0.72
WTA11	699.8357	702.1189	701.0054	701.2495	0.75
WTA12	1306.9126	1309.4616	1308.3382	1308.5187	0.86

PSA-WTA, NVIDIA GeForce GTX Titan X (3072 çekirdeği, 1.0 GHz) ekran kartı üzerinde CUDA kullanılarak yürütülmüştür. Her problem örneği için PSA-WTA'nın aynı yapılandırmalar ile elde ettiği sonuçlar daima aynıdır. Bu sonucun nedeni, her iş parçacığı tarafından üretilen rastgele sayı dizisinin her çalıştırmada aynı olmasıdır. PSA-WTA, GPU üzerinde ortalama çalışma süresini elde etmek için her bir problem örneğini üç kez yürütmüştür. CPU ile performans karşılaştırması yapmak adına öncelikle GPU üzerinde 24 blok kullanılmıştır. Çizelge 5.10, problem örnekleri üzerine PSA-WTA'nın sonuçlarını göstermektedir. Çizelge 5.10 aynı zamanda sonuçların karşılaştırılması için seri SA algoritmasının en iyi ve ortalama değerlerini de göstermektedir. En iyi sonuçlar kalın yazı tipi olarak gösterilmiştir.

Çizelge 5.10, seri algoritmanın elde ettiği en iyi sonuçların 12 problemin dördünde daha iyi değerlere sahip olduğunu göstermektedir. Ortalama sonuçlar göz önüne alındığında, PSA-WTA'nın seri yönteme göre daha iyi sonuçlar elde ettiği görülmüştür. PSA-WTA, 12 problem örneğinin üçünde daha iyi sonuç bulmuştur. Seri yöntem ile PSA-WTA algoritması 12 problemin beşinde aynı sonuçlara sahiptir. Sonuçların aynı olması, bu beş problem örneği için optimum değerlerin bulunduğu anlamına gelebilir.

Çizelge 5.10. WTA1-WTA12 için seri SA ve PSA-WTA algoritmasının sonuçları.

Problem	PSA-WTA	Seri En iyi	Seri Ortalama
WTA1	**48.3640**	**48.3640**	**48.3640**
WTA2	**96.3123**	**96.3123**	**96.3123**
WTA3	**142.1070**	**142.1070**	**142.1070**
WTA4	**248.0285**	**248.0285**	**248.0285**
WTA5	**305.5016**	**305.5016**	**305.5016**
WTA6	353.4801	**353.0767**	353.3112
WTA7	**414.7340**	415.0528	415.4068
WTA8	**497.2972**	498.1049	498.5918
WTA9	535.5422	**534.4408**	535.4559
WTA10	595.3730	**594.0639**	595.3277
WTA11	**699.4143**	699.8357	701.0054
WTA12	1307.0154	**1306.9126**	1308.3382

Her bir problem örneği için, çalışma süresi ve hızlandırma Çizelge 5.11'de sunulmuştur (Seri Algoritmanın Çalışma Süresi-S.R., PSA-WTA Algoritması'nın Çalışma Süresi-P.R). Hızlanma aşağıdaki formül ile elde edilmiştir:

$$hızlanma = \frac{çalışma\ süresi_{SERİ}}{çalışma\ süresi_{PBT-WTA}} \tag{5.14}$$

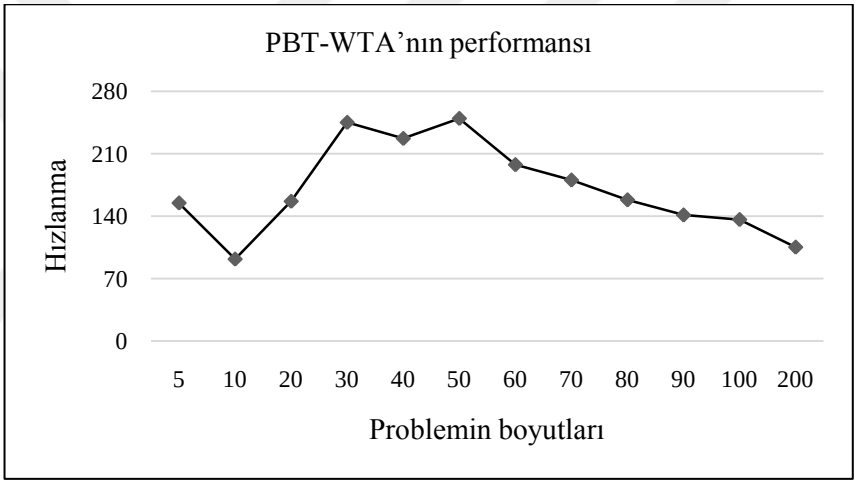
Çizelge 5.11. WTA1-WTA12 için seri SA ve PSA-WTA algoritmasının çalışma süreleri.

Problem	S.R. (s)	P.R. (s)	Hızlanma
WTA1	2985.92	19.28	155x
WTA2	2841.04	30.94	92x
WTA3	2752.49	17.56	157x
WTA4	2754.31	11.23	245x
WTA5	2760.78	12.15	227x
WTA6	2790.03	11.17	250x
WTA7	2787.45	14.09	198x
WTA8	2841.02	15.73	181x
WTA9	2868.79	18.12	158x
WTA10	2812.57	19.90	141x
WTA11	2805.83	20.60	136x
WTA12	2902.15	27.57	105x
Ortalama	*2985.92*	*19.28*	*155x*

Problem örnekleri için seri algoritmanın çalışma süreleri tüm örnekler için birbirine yakındır ve ortalama süre 2985.92 saniyedir. PSA-WTA algoritmasının ortalama çalışma süresi ise 19.28 saniyedir. Şekil 5.8'de hızlanma değerleri görsel olarak

sunulmuştur. 12 problem örneğinde ortalama hızlanma 155x olarak elde edilmiştir. Bu ivmenin SA algoritmasını daha verimli hale getirebildiği rahatlıkla söylenebilir.

Global belleğe erişim, iş parçacıklarının sıralı bir şekilde belleğe erişmesiyle verimli olmaktadır. Bu sıralı erişim *coalesced* olarak adlandırılmaktadır. PSA-WTA algoritmasında her bloğun global belleğe en iyi sonuçlarını yazma sürecinde erişim *coalesced* değildir yani iş parçacıkları sıralı bir erişim gerçekleştirmemiştir. Bundan dolayı, hızlanma değerleri çeşitli boyutlar için değişmektedir (Şekil 5.8). En iyi hızlanma, WTA6 problemi örneği için 250x'tir ve en kötü hızlanma WTA2 problemi örneği için 92x'dir.



Şekil 5.8. WTA1-WTA12 problem örnekleri için hızlanma değerleri.

PSA-WTA algoritmasında kullanılan blok sayısının artırılması global belleğe daha fazla erişime neden olacaktır. Bu nedenle, çalışma süresi daha önce belirtildiği üzere artacaktır. Diğer yandan, PSA-WTA'nın daha çok iş parçacığı ile çalışması, çoklu-başlangıç tekniğine katkı sunarak sonuçların kalitesini artırma imkanı sağlar. 24, 48, 96, 128, 512 ve 1024 sayıda blok kullanılarak elde edilen sonuçlar Çizelge 5.12'de gösterilmiştir. En iyi sonuçlar kalın yazı tipi olarak sunulmuştur. İlk beş problem örneğinin sonuçları (WTA1-WTA5) tüm çalışmalarda aynıdır. PSA-WTA için 1024 blok kullanıldığında, ilk beş problem örneği dışında kalan tüm problem örneklerinde daha iyi sonuçlar elde edilmiştir. Ayrıca, PSA-WTA için 1024 blok kullanıldığında, çalışma süresi, 24 blok kullanan PSA-WTA algoritmasına karşılık gelen seri algoritmanın çalışma süresinin yarısından daha azdır.

Çizelge 5.12. WTA1-WTA12 için farklı blok sayılarının kullanıldığı PSA-WTA algoritmasının sonuçları.

Problem	**Blok Sayıları (İlk sütunda sonuçlar, ikinci sütunda çalışma süreleri gösterilmiştir)**											
	24		***48***		***96***		***128***		***512***		***1024***	
WTA1	**48.3640**	19.28	**48.3640**	38.49	**48.3640**	75.96	**48.3640**	75.96	**48.3640**	383.39	**48.3640**	764.63
WTA2	**96.3123**	30.94	**96.3123**	59.76	**96.3123**	119.16	**96.3123**	119.16	**96.3123**	623.34	**96.3123**	1245.74
WTA3	**142.1070**	17.56	**142.1070**	33.25	**142.1070**	65.78	**142.1070**	65.78	**142.1070**	324.42	**142.1070**	642.34
WTA4	**248.0285**	11.23	**248.0285**	21.24	**248.0285**	41.66	**248.0285**	41.66	**248.0285**	217.12	**248.0285**	431.41
WTA5	**305.5016**	12.15	**305.5016**	23.18	**305.5016**	45.07	**305.5016**	45.07	**305.5016**	198.64	**305.5016**	391.05
WTA6	353.4801	11.17	353.3609	21.15	**353.0102**	41.03	**353.0102**	41.03	**353.0102**	233.08	**353.0102**	459.42
WTA7	414.7340	14.09	414.7340	26.85	414.7340	52.08	414.7340	52.08	**414.6011**	232.36	**414.6011**	428.31
WTA8	497.2972	15.73	497.2972	30.01	497.2972	58.23	497.2972	58.23	**496.7948**	278.67	**496.7948**	536.04
WTA9	535.5422	18.12	534.6199	34.76	534.3872	68.47	534.3872	68.47	**533.9201**	343.87	**533.9201**	670.37
WTA10	595.3730	19.90	594.4800	38.66	594.4658	76.44	594.4658	76.44	593.6951	382.01	**592.7965**	749.48
WTA11	699.4143	20.60	699.4143	40.25	699.4143	79.31	699.4143	79.31	**697.6242**	401.19	**697.6242**	787.51
WTA12	1307.0154	27.57	1306.8689	55.00	1304.4195	110.80	1304.4195	147.11	1304.4195	586.39	**1302.9348**	1181.51

BÖLÜM 6

SONUÇLAR

Bu tezde SA algoritmasının GPU üzerinde paralelleştirmesine yönelik bir yöntem geliştirilmiştir. SA algoritmasının daha efektif bir şekilde çalışmasını sağlayan çoklu-başlangıç tekniği ile problem üzerinde elde edilecek sonuçların kalitesini artırmak amaçlanmış, aynı zamanda SA algoritmasının dezavantajı olarak görülen çalışma süresi problemine çözüm getirmek amacıyla paralel bir yöntem geliştirilmiştir. Geliştirilen yöntem, üç farklı problem üzerinde test edilmiş, probleme özgü gerekli optimizasyonlar yapılarak maksimum fayda sağlanması amaçlanmıştır. Yapılan paralelleştirme çalışmalarının sonuçları incelendiğinde, SA algoritmasının çalışma süresi üzerinde etkili olduğu gözlenmiştir. Çalışma süresi üzerindeki hızlanma, probleme ve veri boyutuna göre değişmekle birlikte genel olarak değerlendirildiğinde paralel SA algoritmasının önemli ölçüde katkı sağladığı görülmüştür. Bunun yanı sıra problemlerin çözüm kalitesi üzerinde de büyük katkılar sağlanmıştır. Özet olarak tez çalışmasının sonucu olarak sağlanan bu iki katkı ile SA algoritmasının daha etkin bir şekilde kullanılabileceği söylenebilir.

Tez kapsamında aşağıda belirtilen akademik çalışmalar gerçekleştirilmiştir:

- Proje: "Benzetilmiş Tavlama Algoritmasının Grafik İşlemci Ünitesi Kullanılarak Paralelleştirilmesi ve NP-zor Problemlere Uygulanması", Karabük Üniversitesi Bilimsel Araştırma Projesi, Araştırmacı, Proje No: KBU-BAP-15/2-DR-027, Aralık 2015-Haziran 2017.
- Uluslararası Makale: "A Parallel Simulated Annealing Algorithm for Weapon-Target Assignment Problem", International Journal of Advanced Computer Science and Applications, 2017.

- Uluslararası Makale: “Parallel Approach for Solving 0/1 Knapsack Problem using Simulated Annealing Algorithm on CUDA Platform”, International Journal of Computer Science and Information Security, 2016.
- Uluslararası Bildiri: “Solving Bin Packing Problem using Simulated Annealing”, ISERD- 127th International Conference on Recent Innovations in Engineering and Technology (ICRIET), 2017.
- Uluslararası Bildiri: “Comparison Of Neighborhood Search Techniques for Solving Weapon-Target Assignment Problem using Simulated Annealing Algorithm”, International Conference on Mathematics and Engineering (ICOME), 2017.
- Ulusal Bildiri: “Gezgin Satıcı Probleminin Benzetilmiş Tavlama Yöntemiyle Çözümünde Paralel Hesaplamanın Kullanılması”, Akademik Bilişim, 2015.

Yapılan çalışmalar neticesinde SA algoritmasının GPU üzerinde paralelleştirilerek kullanılması, algoritmanın zayıf yönlerinin ortadan kaldırılmasını sağlamış ve algoritmayı efektif hale getirmiştir. Bununla birlikte literatürde iki ya da daha fazla sezgisel algoritmanın birlikte kullanıldığı hibrit yöntemlerin sıkça kullanıldığı ve başarılı sonuçlar elde edildiği görülmektedir. Geliştirilen paralel SA algoritması diğer sezgisel algoritmalarla birlikte kullanılarak probleme özgü daha etkin hibrit yöntemlerin geliştirilmesi mümkün olabilir.

KAYNAKLAR

1. Güneş, A., "Paralel programlama ile el yazısı rakamlarının tanınması", Yüksek Lisans Tezi, ***Isparta Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği A.B.D.***, Isparta, (2011).

2. Akçay, M. and Erdem, H. A., "Paralel Hesaplama ve MATLAB Uygulamaları", ***Akademik Bilişim***, Muğla, (2011).

3. Internet: Lawrence Livermore National Laboratory, "Introduction to Parallel Computing", **https://computing.llnl.gov/tutorials/parallel_comp/** (2017).

4. Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E., and Purcell, T. J., "A survey of general-purpose computation on graphics hardware", ***Computer Graphics Forum***, 80–113 (2007).

5. Czech, Z. J. and Czarnas, P., "Parallel simulated annealing for the vehicle routing problem with time windows", ***Parallel, Distributed and Network-Based Processing, 2002. Proceedings. 10th Euromicro Workshop on***, 376–383 (2002).

6. Onbaşoğlu, E. and Özdamar, L., "Parallel simulated annealing algorithms in global optimization", ***Journal Of Global Optimization***, 19 (1): 27–50 (2001).

7. Debudaj-Grabysz, A. and Rabenseifner, R., "Nesting OpenMP in MPI to implement a hybrid communication method of parallel simulated annealing on a cluster of SMP nodes", ***European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting***, 18–27 (2005).

8. Chang, Y.-L., Chen, K.-S., Huang, B., Chang, W.-Y., Benediktsson, J. A., and Chang, L., "A parallel simulated annealing approach to band selection for high-dimensional remote sensing images", ***IEEE Journal Of Selected Topics In Applied Earth Observations And Remote Sensing***, 4 (3): 579–590 (2011).

9. Bajrami, E., Ašić, M., Cogo, E., Trnka, D., and Nosovic, N., "Performance comparison of simulated annealing algorithm execution on GPU and CPU", ***MIPRO, 2012 Proceedings of the 35th International Convention***, 1785–1788 (2012).

10. Han, Y., Roy, S., and Chakraborty, K., "Optimizing simulated annealing on GPU: A case study with IC floorplanning", ***Quality Electronic Design (ISQED), 2011 12th International Symposium on***, 1–7 (2011).

11. Fobel, C., Gréwal, G., Collier, R., and Stacey, D., "GPU Approach to FPGA placement based on star+", ***New Circuits and Systems Conference (NEWCAS), 2012 IEEE 10th International***, 229–232 (2012).

12. Li, H. and Liu, C., "Prediction of protein structures using GPU based simulated annealing", ***Machine Learning and Applications (ICMLA), 2012 11th International Conference on***, 630–633 (2012).

13. Fabry-Asztalos, L., Lőrentz, I., and Andonie, R., "Molecular distance geometry optimization using geometric build-up and evolutionary techniques on GPU", ***Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), 2012 IEEE Symposium on***, 321–328 (2012).

14. Fobel, C., Grewal, G., and Stacey, D., "A scalable, serially-equivalent, high-quality parallel placement methodology suitable for modern multicore and GPU architectures", ***Field Programmable Logic and Applications (FPL), 2014 24th International Conference on***, 1–8 (2014).

15. Kataoka, M., Tsukiyamia, S., Kambe, T., and Fukui, M., "An effective method to use GPU for rectangle packing", ***New Circuits and Systems Conference (NEWCAS), 2012 IEEE 10th International***, 129–132 (2012).

16. Ferreiro, A., García, J., López-Salas, J. G., and Vázquez, C., "An efficient implementation of parallel simulated annealing algorithm in GPUs", ***Journal Of Global Optimization***, 57 (3): 863–890 (2013).

17. Stivala, A. D., Stuckey, P. J., and Wirth, A. I., "Fast and accurate protein substructure searching with simulated annealing and GPUs", ***BMC Bioinformatics***, 11 (1): 446 (2010).

18. Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E., "Equation of state calculations by fast computing machines", ***The Journal Of Chemical Physics***, 21 (6): 1087–1092 (1953).

19. Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., "Optimization by simulated annealing", ***Science***, 220 (4598): 671–680 (1983).

20. Černý, V., "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm", ***Journal Of Optimization Theory And Applications***, 45 (1): 41–51 (1985).

21. Şahin, S., "Çelik kafes enerji nakil hattı direklerinin benzetimsel tavlama yöntemi kullanılarak bilgisayar destekli optimizasyonu", Doktora Tezi, ***Orta Doğu Teknik Üniversitesi, Fen Bilimleri Enstitüsü, İnşaat Mühendisliği A.B.D.***, Ankara, (2016).

22. İnternet: Busetti, F., "Simulated Annealing Overview", **http://personalpages.to.infn.it/~ferraro/retineurali/articoli_rn/simantut.pdf** (2003).

23. Alizamir, S., Rebennack, S., and Pardalos, P. M., "Improving the neighborhood selection strategy in simulated annealing using the optimal stopping problem", Simulated Annealing, ***InTech***, (2008).

24. İnternet: Code Capsule, "Simulated Annealing Applied to the Traveling Salesman Problem | Code Capsule", **http://codecapsule.com/2010/04/06/simulated-annealing-traveling-salesman/** (2017).

25. Öz, K., "Yüz tanıma sisteminin paralel programlama ile optimizasyonu", Yüksek Lisans Tezi, ***Karabük Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği A.B.D.***, Karabük, (2012).

26. Internet: E2Matrix Research Lab, "E2Matrix Research Lab | CPU V/S GPU", **http://www.e2matrix.com/blog/cpu-vs-gpu/** (2017).

27. Houston, M., "General Purpose Computation on Graphics Processors (GPGPU)", (2005).

28. İnternet: NVIDIA, Inc., "CUDA C Programming Guide", **http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html** (2017).

29. Karch, G., "GPU-based acceleration of selected clustering techniques", Master Thesis, ***Silesian University of Technology in Gliwice Faculty of Automatic Control, Electronics and Computer Science***, (2010).

30. Koopmans, T. C. and Beckmann, M., "Assignment problems and the location of economic activities", ***Econometrica: Journal Of The Econometric Society***, 53–76 (1957).

31. Steinberg, L., "The backboard wiring problem: A placement algorithm", ***Siam Review***, 3 (1): 37–50 (1961).

32. Geoffrion, A. M. and Graves, G. W., "Scheduling parallel production lines with changeover costs: Practical application of a quadratic assignment/LP approach", ***Operations Research***, 24 (4): 595–610 (1976).

33. Pollatschek, M., Gershoni, H., and Radday, Y., "Optimization Of Typewriter Keyboard By Computer-Simulation", ***Angewandte Informatik***, (10): 438–439 (1976).

34. Krarup, J. and Pruzan, P. M., "Computer-aided layout design", ***Mathematical Programming In Use***, 75–94 (1978).

35. Brusco, M. J. and Stahl, S., "Using quadratic assignment methods to generate initial permutations for least-squares unidimensional scaling of symmetric proximity matrices", ***Journal Of Classification***, 17 (2): 197–223 (2000).

36. Dickey, J. and Hopkins, J., "Campus building arrangement using TOPAZ", ***Transportation Research***, 6 (1): 59–68 (1972).

37. Elshafei, A. N., "Hospital layout as a quadratic assignment problem", ***Journal Of The Operational Research Society***, 28 (1): 167–179 (1977).

38. Bos, J., "Zoning in forest management: a quadratic assignment problem solved by simulated annealing", ***Journal Of Environmental Management***, 37 (2): 127–145 (1993).

39. Loiola, E. M., de Abreu, N. M. M., Boaventura-Netto, P. O., Hahn, P., and Querido, T., "A survey for the quadratic assignment problem", ***European Journal Of Operational Research***, 176 (2): 657–690 (2007).

40. James, T., Rego, C., and Glover, F., "A cooperative parallel tabu search algorithm for the quadratic assignment problem", ***European Journal Of Operational Research***, 195 (3): 810–826 (2009).

41. Burkard, R. E., Karisch, S. E., and Rendl, F., "QAPLIB–a quadratic assignment problem library", ***Journal Of Global Optimization***, 10 (4): 391–403 (1997).

42. Kováč, M., "Solving Quadratic Assignment Problem in parallel using local search with simulated annealing elements", ***Digital Technologies (DT), 2013 International Conference on***, 18–20 (2013).

43. Zhu, W., Curry, J., and Marquez, A., "SIMD tabu search for the quadratic assignment problem with graphics hardware acceleration", ***International Journal Of Production Research***, 48 (4): 1035–1047 (2010).

44. Czapiński, M., "An effective parallel multistart tabu search for quadratic assignment problem on CUDA platform", ***Journal Of Parallel And Distributed Computing***, 73 (11): 1461–1468 (2013).

45. Tosun, U., "On the performance of parallel hybrid algorithms for the solution of the quadratic assignment problem", ***Engineering Applications Of Artificial Intelligence***, 39267–278 (2015).

46. Martí, R., Resende, M. G., and Ribeiro, C. C., "Multi-start methods for combinatorial optimization", ***European Journal Of Operational Research***, 226 (1): 1–8 (2013).

47. Wang, J.-C., "A multistart simulated annealing algorithm for the quadratic assignment problem", ***Innovations in Bio-Inspired Computing and Applications (IBICA), 2012 Third International Conference on***, 19–23 (2012).

48. Xiao, S. and Feng, W., "Inter-block GPU communication via fast barrier synchronization", ***Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on***, 1–12 (2010).

49. Burkard, R. E. and Rendl, F., "A thermodynamically motivated simulation procedure for combinatorial optimization problems", ***European Journal Of Operational Research***, 17 (2): 169–174 (1984).

50. Martello, S., Pisinger, D., and Toth, P., "Dynamic programming and strong bounds for the 0-1 knapsack problem", ***Management Science***, 45 (3): 414–424 (1999).

51. Sahni, S., "Approximate algorithms for the 0/1 knapsack problem", ***Journal Of The ACM (JACM)***, 22 (1): 115–124 (1975).

52. Liu, A., Wang, J., Han, G., Wang, S., and Wen, J., "Improved simulated annealing algorithm solving for 0/1 knapsack problem", 2***Intelligent Systems Design and Applications, 2006. ISDA'06. Sixth International Conference on***, 1159–1164 (2006).

53. Shi, H., "Solution to 0/1 knapsack problem based on improved ant colony algorithm", ***Information Acquisition, 2006 IEEE International Conference on***, 1062–1066 (2006).

54. Raidl, G. R., "An improved genetic algorithm for the multiconstrained 0-1 knapsack problem", ***Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on***, 207–211 (1998).

55. Hanafi, S. and Freville, A., "An efficient tabu search approach for the 0–1 multidimensional knapsack problem", ***European Journal Of Operational Research***, 106 (2–3): 659–675 (1998).

56. Akbar, M., Manning, E., Shoja, G., and Khan, S., "Heuristic solutions for the multiple-choice multi-dimension knapsack problem", ***Computational Science-ICCS 2001***, 659–668 (2001).

57. Fréville, A., "The multidimensional 0–1 knapsack problem: An overview", ***European Journal Of Operational Research***, 155 (1): 1–21 (2004).

58. Zou, D., Gao, L., Li, S., and Wu, J., "Solving 0–1 knapsack problem by a novel global harmony search algorithm", ***Applied Soft Computing***, 11 (2): 1556–1564 (2011).

59. Kong, X., Gao, L., Ouyang, H., and Li, S., "A simplified binary harmony search algorithm for large scale 0–1 knapsack problems", ***Expert Systems With Applications***, 42 (12): 5337–5355 (2015).

60. El Baz, D. and Elkihel, M., "Load balancing methods and parallel dynamic programming algorithm using dominance technique applied to the 0–1 knapsack problem", ***Journal Of Parallel And Distributed Computing***, 65 (1): 74–84 (2005).

61. Goldman, A. and Trystram, D., "An efficient parallel algorithm for solving the knapsack problem on hypercubes", ***Journal Of Parallel And Distributed Computing***, 64 (11): 1213–1222 (2004).

62. Boyer, V., El Baz, D., and Elkihel, M., "Solving knapsack problems on GPU", ***Computers & Operations Research***, 39 (1): 42–47 (2012).

63. Pospichal, P., Schwarz, J., and Jaros, J., "Parallel genetic algorithm solving 0/1 knapsack problem running on the gpu", 2010***16th International Conference on Soft Computing MENDEL***, 64–70 (2010).

64. Hajarian, M., Shahbahrami, A., and Hoseini, F., "A parallel solution for the 0–1 knapsack problem using firefly algorithm", ***Swarm Intelligence and Evolutionary Computation (CSIEC), 2016 1st Conference on***, 25–30 (2016).

65. Fingler, H., Cáceres, E. N., Mongelli, H., and Song, S. W., "A CUDA based solution to the multidimensional knapsack problem using the ant colony optimization", ***Procedia Computer Science***, 2984–94 (2014).

66. Lloyd, S. P. and Witsenhausen, H. S., "Weapons allocation is NP-complete.", ***1986 Summer Computer Simulation Conference***, 1054–1058 (1986).

67. Ma, F., Ni, M., Gao, B., and Yu, Z., "An efficient algorithm for the weapon target assignment problem", ***Information and Automation, 2015 IEEE International Conference on***, 2093–2097 (2015).

68. Ahuja, R. K., Kumar, A., Jha, K. C., and Orlin, J. B., "Exact and heuristic algorithms for the weapon-target assignment problem", ***Operations Research***, 55 (6): 1136–1146 (2007).

69. Sikanen, T., "Solving weapon target assignment problem with dynamic programming", ***Technical Report, Mat− 2.4108 Independent Research Projects In Applied Mathematics***, (2008).

70. Madni, A. M. and Andrecut, M., "Efficient heuristic approach to the weapon-target assignment problem", ***Journal Of Aerospace Computing, Information, And Communication***, 6 (6): 405–414 (2009).

71. Tokgöz, A. and Bulkan, S., "Weapon target assignment with combinatorial optimization techniques", ***IJARAI) International Journal Of Advanced Research In Artificial Intelligence***, 2 (7): (2013).

72. Zhang, J., Wang, X., Xu, C., and Yuan, D., "ACGA algorithm of solving weapon-target assignment problem", ***Open Journal Of Applied Sciences***, 2 (04): 74 (2013).

73. Lee, Z.-J., Lee, C.-Y., and Su, S.-F., "An immunity-based ant colony optimization algorithm for solving weapon–target assignment problem", ***Applied Soft Computing***, 2 (1): 39–47 (2002).

74. Shang, G., "Solving weapon-target assignment problems by a new ant colony algorithm", 1***Computational Intelligence and Design, 2008. ISCID'08. International Symposium on***, 221–224 (2008).

75. Zeng, X., Zhu, Y., Nan, L., Hu, K., Niu, B., and He, X., "Solving weapon-target assignment problem using discrete particle swarm optimization", ***Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on***, 3562–3565 (2006).

EK AÇIKLAMALAR A.

QAPLIB VERİ SETLERİ İÇİN PSA-QAP ALGORİTMASININ SONUÇLARI

PSA-QAP algoritmasının QAPLIB'de bulunan asimetrik veri setlerinde üzerindeki sonuçları Çizelge Ek A.1'de gösterilmiştir.

Çizelge Ek A.1. Asimetik veri setleri için PSA-QAP algoritmasının sonuçları.

Veri Seti	BKS	PSA-QAP I			PSA-QAP II		
		RPD (%)	Süre (s)	Hızlanma	RPD (%)	Süre (s)	Hızlanma
bur26a	5426670	0.000	0.18	6.2	0.000	0.79	6.8
bur26b	3817852	0.000	0.18	6.4	0.000	0.84	6.6
bur26c	5426795	0.000	0.16	7.4	0.000	0.75	7.3
bur26d	3821225	0.000	0.17	6.9	0.000	0.84	6.6
bur26e	5386879	0.000	0.16	7.2	0.000	0.77	7.1
bur26f	3782044	0.000	0.18	6.5	0.000	0.88	6.2
bur26g	10117172	0.000	0.15	7.5	0.000	0.74	7.3
bur26h	7098658	0.000	0.17	6.9	0.000	0.81	6.7
lipa20a	3683	0.000	0.16	4.4	0.000	0.80	3.9
lipa20b	27076	0.000	0.13	5.1	0.000	0.64	4.6
lipa30a	13178	0.000	0.32	5.0	0.000	1.60	4.7
lipa30b	151426	0.000	0.21	7.5	0.000	0.94	7.9
lipa40a	31538	0.000	61.30	6.4	0.000	313.98	6.0
lipa40b	476581	0.000	43.16	8.4	0.000	215.94	8.3
lipa50a	62093	0.000	78.00	7.9	0.000	390.73	7.8
lipa50b	1210244	0.000	50.23	11.2	0.000	252.63	11.1
lipa60a	107218	0.000	91.35	9.6	0.000	456.46	9.6
lipa60b	2520135	0.000	58.99	14.1	0.000	292.66	14.0
lipa70a	169755	0.426	104.38	11.8	0.000	522.60	11.9
lipa70b	4603200	0.000	64.52	17.5	0.000	322.50	17.5
lipa80a	253195	0.582	130.82	11.2	0.543	655.02	11.3
lipa80b	7763962	0.000	92.62	16.7	0.000	497.79	15.6
lipa90a	360630	0.548	130.42	14.7	0.529	654.90	14.7
lipa90b	12490441	0.000	77.95	24.3	0.000	361.64	25.8
sko42	15812	0.000	61.32	6.9	0.000	306.45	7.0
sko49	23386	0.000	71.17	8.3	0.000	357.53	8.2
sko56	34458	0.000	80.65	9.4	0.000	406.17	9.3
sko64	48498	0.000	123.13	8.1	0.000	618.00	7.9
sko72	66256	0.006	103.10	12.1	0.000	516.29	12.3
sko81	90998	0.011	111.01	14.2	0.000	555.84	14.1
sko90	115534	0.042	122.64	15.9	0.029	613.55	16.0
sko100a	152002	0.022	136.66	17.8	0.000	680.83	17.9
sko100b	153890	0.000	136.97	17.8	0.000	680.07	17.9
sko100c	147862	0.008	135.96	17.8	0.003	682.15	17.9
sko100d	149576	0.027	135.81	18.0	0.013	682.64	17.9
sko100e	149150	0.020	135.40	18.2	0.009	679.87	18.0

Veri Seti	BKS	PSA-QAP I			PSA-QAP II		
		RPD (%)	Süre (s)	Hızlanma	RPD (%)	Süre (s)	Hızlanma
sko100f	149036	0.043	136.23	18.2	0.038	682.04	17.9
tai10b	1183760	0.000	0.11	1.0	0.000	0.54	0.7
tai12b	39464925	0.000	0.09	2.0	0.000	0.43	1.7
tai15b	51765268	0.000	0.09	3.7	0.000	0.43	3.2
tai20b	122455319	0.000	0.11	6.2	0.000	0.51	5.6
tai25b	344355646	0.000	0.14	7.6	0.000	0.61	7.6
tai30b	637117113	0.000	0.18	8.5	0.000	0.76	9.4
tai35b	283315445	0.000	0.23	9.2	0.000	0.94	10.5
tai40b	637250948	0.000	32.24	11.4	0.000	167.13	10.6
tai50b	458821517	0.050	39.55	14.3	0.000	187.89	14.3
tai60b	608215054	0.107	48.88	17.0	0.000	240.80	16.8
tai80b	818415043	0.244	86.86	18.1	0.098	484.84	16.1
tai100b	1185996137	0.436	82.70	28.6	0.382	413.06	28.4
Ortalama		*0.052*	*56.47*	*11.0*	*0.034*	*283.81*	*11.0*

PSA-QAP algoritmasının QAPLIB'te bulunan simetrik veri setlerinde üzerindeki sonuçları Çizelge Ek A.2'de ise gösterilmiştir.

Çizelge Ek A.2. Simetik veri setleri için PSA-QAP algoritmasının sonuçları.

Veri Seti	BKS	PSA-QAP I			PSA-QAP II		
		RPD (%)	Süre (s)	Hızlanma	RPD (%)	Süre (s)	Hızlanma
chr12a	9552	0.000	0.09	1.5	0.000	0.44	1.4
chr12b	9742	0.000	0.09	1.9	0.000	0.41	1.5
chr12c	11156	0.000	0.08	2.0	0.000	0.40	1.5
chr15a	9896	0.000	0.09	2.9	0.000	0.44	2.1
chr15b	7990	0.000	0.09	2.8	0.000	0.43	2.1
chr15c	9504	0.000	0.09	2.9	0.000	0.43	2.1
chr18a	11098	0.000	0.10	3.7	0.000	0.48	2.8
chr18b	1534	0.000	0.12	3.1	0.000	0.59	2.4
chr20a	2192	0.000	0.13	3.7	0.000	0.64	2.8
chr20b	2298	0.000	0.13	3.9	0.000	0.63	2.8
chr20c	14142	0.000	0.13	4.0	0.000	0.63	3.0
chr22a	6156	0.000	0.15	4.0	0.000	0.74	3.0
chr22b	6194	0.710	0.15	4.2	0.710	0.73	3.0
chr25a	3796	0.000	0.17	4.0	0.000	0.86	3.4
els19	17212548	0.000	0.16	3.8	0.000	0.64	3.2
esc16a	68	0.000	0.13	2.4	0.000	0.63	1.9
esc16b	292	0.000	0.13	2.5	0.000	0.64	1.8
esc16c	160	0.000	0.12	2.5	0.000	0.62	1.9

Veri Seti	BKS	PSA-QAP I			PSA-QAP II		
		RPD (%)	Süre (s)	Hızlanma	RPD (%)	Süre (s)	Hızlanma
esc16d	16	0.000	0.13	2.5	0.000	0.63	1.8
esc16e	28	0.000	0.13	2.6	0.000	0.64	1.8
esc16f	0	0.000	0.10	2.3	0.000	0.51	1.8
esc16g	26	0.000	0.13	2.5	0.000	0.64	1.8
esc16h	996	0.000	0.12	2.6	0.000	0.59	1.9
esc16i	14	0.000	0.13	2.3	0.000	0.63	1.8
esc16j	8	0.000	0.13	2.5	0.000	0.65	1.7
esc32a	130	0.000	0.35	3.6	0.000	1.77	2.6
esc32b	168	0.000	0.35	3.3	0.000	1.77	2.6
esc32c	642	0.000	0.35	3.4	0.000	1.76	2.6
esc32d	200	0.000	0.36	3.3	0.000	1.79	2.5
esc32e	2	0.000	0.35	3.3	0.000	1.78	2.5
esc32g	6	0.000	0.35	3.4	0.000	1.78	2.4
esc32h	438	0.000	0.35	3.4	0.000	1.78	2.6
esc64a	116	0.000	123.47	4.6	0.000	632.85	4.3
had12	1652	0.000	0.10	1.7	0.000	0.48	1.3
had14	2724	0.000	0.10	2.2	0.000	0.50	1.7
had16	3720	0.000	0.12	2.7	0.000	0.60	1.9
had18	5358	0.000	0.14	3.0	0.000	0.68	2.1
had20	6922	0.000	0.16	3.2	0.000	0.79	2.3
kra30a	88900	0.000	0.20	5.6	0.000	0.97	4.2
kra30b	91420	0.000	0.20	5.3	0.000	0.97	4.2
kra32	88700	0.000	0.28	4.6	0.000	1.39	3.6
nug12	578	0.000	0.09	1.8	0.000	0.44	1.4
nug14	1014	0.000	0.10	2.2	0.000	0.49	1.7
nug15	1150	0.000	0.11	2.4	0.000	0.52	1.8
nug16a	1610	0.000	0.12	2.6	0.000	0.59	1.9
nug16b	1240	0.000	0.12	2.7	0.000	0.59	1.9
nug17	1732	0.000	0.12	2.8	0.000	0.61	2.1
nug18	1930	0.000	0.13	2.8	0.000	0.66	2.2
nug20	2570	0.000	0.15	3.3	0.000	0.77	2.4
nug21	2438	0.000	0.16	3.5	0.000	0.81	2.5
nug22	3596	0.000	0.17	3.5	0.000	0.85	2.6
nug24	3488	0.000	0.20	3.8	0.000	0.98	2.6
nug25	3744	0.000	0.22	3.8	0.000	1.08	2.7
nug27	5234	0.000	0.25	3.8	0.000	1.25	2.7
nug28	5166	0.000	0.27	3.5	0.000	1.33	2.8
nug30	6124	0.000	0.29	3.5	0.000	1.46	2.9
rou12	235528	0.000	0.08	1.4	0.000	0.42	1.2
rou15	354210	0.000	0.09	2.5	0.000	0.43	2.0
rou20	725522	0.008	0.11	4.4	0.000	0.52	3.2

Veri Seti	BKS	PSA-QAP I			PSA-QAP II		
		RPD (%)	Süre (s)	Hızlanma	RPD (%)	Süre (s)	Hızlanma
scr12	31410	0.000	0.08	1.8	0.000	0.41	1.3
scr15	51140	0.000	0.09	2.6	0.000	0.44	2.0
scr20	110030	0.000	0.11	4.2	0.000	0.54	3.2
ste36a	9526	0.294	0.33	5.0	0.042	1.63	3.8
ste36b	15852	0.000	0.28	5.5	0.000	1.40	4.4
ste36c	8239110	0.000	0.23	6.8	0.000	0.93	6.2
tai10a	135028	0.000	0.09	1.0	0.000	0.47	0.7
tai12a	224416	0.000	0.08	1.5	0.000	0.42	1.2
tai15a	388214	0.000	0.10	2.3	0.000	0.43	2.0
tai17a	491812	0.000	0.09	3.2	0.000	0.46	2.5
tai20a	703482	0.000	0.11	4.2	0.000	0.51	3.2
tai25a	1167256	0.409	0.14	5.7	0.409	0.61	4.3
tai30a	1818146	1.244	0.17	6.1	1.147	0.76	5.3
tai35a	2422002	1.660	0.21	6.7	1.222	0.92	6.0
tai40a	3139370	0.462	34.00	5.5	0.000	179.16	5.3
tai50a	4938796	1.204	42.20	6.9	0.946	210.60	6.9
tai60a	7205962	1.190	50.83	8.4	1.190	254.15	8.3
tai64c	1855928	0.000	113.70	4.8	0.000	597.93	4.7
tai80a	13499184	1.670	87.99	9.3	1.523	445.77	9.1
tai100a	21052466	1.705	86.72	14.3	1.568	434.10	14.3
tho30	149936	0.000	0.22	4.8	0.000	0.98	4.1
tho40	240516	0.000	44.99	4.3	0.000	224.48	4.3
wil50	48816	0.000	72.28	4.5	0.000	363.29	4.4
wil100	273038	0.006	135.15	9.7	0.006	679.88	9.5
Ortalama		*0.127*	*9.68*	*3.8*	*0.106*	*49.16*	*3.1*

ÖZGEÇMİŞ

Emrullah SONUÇ 1986 yılında Karabük'te doğdu. İlkokulu Karabük Anayasa İlkokulu'nda tamamladı. Ortaokul ve lise eğitimini Karabük Anadolu İmam-Hatip Lisesi'nde tamamlayarak 2003 yılında mezun oldu. 2004 yılında Lefke Avrupa Üniversitesi Mühendislik-Mimarlık Fakültesi Bilgisayar Mühendisliği Bölümü'nü kazanarak bu bölümde lisans öğrenimine başladı. 2006 yılında yatay geçiş yoluyla geçtiği Selçuk Üniversitesi Mühendislik-Mimarlık Fakültesi Bilgisayar Mühendisliği Bölümü'nde lisans eğitimini tamamladı ve 2008 yılında lisans diploması almaya hak kazandı. 2010 yılında Karabük Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda başladığı yüksek lisans eğitimini 2012 yılında tamamladı. Aynı yıl aynı anabilim dalında doktora eğitimine başladı ve doktora eğitimini 2017 yılı Haziran ayında tamamladı. Karabük Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Yazılım Anabilim Dalı'nda Araştırma Görevlisi olarak 2009 yılında başladığı görevini halen sürdürmektedir. Evli ve biri kız biri erkek olmak üzere iki çocuk babasıdır.

ADRES BİLGİLERİ

Adres : Karabük Üniversitesi, Mühendislik Fakültesi,
Bilgisayar Mühendisliği Bölümü
Demir-Çelik Kampüsü / KARABÜK

Tel : 0 (536) 675 2104

E-posta : esonuc@karabuk.edu.tr