T.C.
SELÇUK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

AĞAÇ-TOHUM ALGORİTMASI İÇİN CUDA TABANLI BİR PARALEL PROGRAMLAMA YAKLAŞIMI

Ahmet Cevahir ÇINAR

YÜKSEK LİSANS TEZİ

Bilgisayar Mühendisliği Anabilim Dalı

Nisan-2016
KONYA

TEZ KABUL VE ONAYI

Ahmet Cevahir ÇINAR tarafından hazırlanan AĞAÇ-TOHUM ALGORİTMASI İÇİN CUDA TABANLI BİR PARALEL PROGRAMLAMA YAKLAŞIMI adlı tez çalışması 25.04.2016 tarihinde aşağıdaki jüri tarafından oy birliği ile Selçuk Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda YÜKSEK LİSANS TEZİ olarak kabul edilmiştir.

Jüri Üyeleri	**İmza**
Başkan Doç. Dr. Halife KODAZ	……………………
Danışman Doç. Dr. Mustafa Servet KIRAN	……………………
Üye Yrd.Doç.Dr.Onur İNAN	……………………

Yukarıdaki sonucu onaylarım.

Prof. Dr. Aşır GENÇ
FBE Müdürü

TEZ BİLDİRİMİ

Bu tezdeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edildiğini ve tez yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yapıldığını bildiririm.

DECLARATION PAGE

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Ahmet Cevahir ÇINAR

Tarih:

ÖZET

YÜKSEK LİSANS TEZİ

AĞAÇ-TOHUM ALGORİTMASI İÇİN CUDA TABANLI BİR PARALEL PROGRAMLAMA YAKLAŞIMI

Ahmet Cevahir ÇINAR

Selçuk Üniversitesi Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Doç.Dr. Mustafa Servet KIRAN

2016, 102 Sayfa

Jüri
Doç.Dr. Mustafa Servet KIRAN
Doç.Dr. Halife KODAZ
Yrd. Doç.Dr. Onur İNAN

Son yıllarda, grafik işlem birimi ile genel amaçlı hesaplama (GPGPU) paralel hesaplama alanında büyük bir popülerlik kazanmıştır ve GPGPU kullanımı gün geçtikçe merkezi işlem birimi (CPU) üretim teknolojisinin fiziksel sınırlarına yaklaşmasından dolayı pek çok yüksek boyutlu zaman alıcı sorunları çözmek için artmaktadır.

GPGPU popülerleşmesinde son yıllarda veri miktarında meydana gelen artış gösterilebilir çünkü seri programlama ile bu verinin işlenmesinde zaman sorunu ortaya çıkmıştır. Ayrıca kolay kullanılabilir paralel programlama geliştirme ortamlarının artması da bu süreci desteklemiştir.

Bu çalışmada popülasyon tabanlı metasezgisel algoritmalardan ağaçlar ile tohumları arasındaki neslin devamını modelleyen Ağaç-Tohum Algoritması' nın (TSA) paralelleştirilmesi üzerinde durulmuştur. Bu tezde TSA paralel geliştirme ortamlarından CUDA üzerinde gerçeklenmiştir. Deneysel sonuçlar yüksek boyutlu ve yüksek ağaç sayılı(popülasyon boyutu) problemlerin çözümünde paralel TSA'nın seri olan gerçeklenmesine göre yaklaşık 184 kat daha hızlı olduğunu göstermiştir.

Anahtar Kelimeler: Ağaç-Tohum Algoritması, CUDA, Doğa Esinli Sezgiseller, Optimizasyon, Paralel Hesaplama, Paralel Programlama

ABSTRACT

MS THESIS

A CUDA-BASED PARALLEL PROGRAMMING APPROACH TO TREE-SEED ALGORITHM

Ahmet Cevahir ÇINAR

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCE OF
SELÇUK UNIVERSITY
THE DEGREE OF MASTER OF SCIENCE
IN COMPUTER ENGINEERING

Advisor: Assoc. Prof. Mustafa Servet KIRAN

2016, 102 Pages

Jury
Assoc. Prof. Mustafa Servet KIRAN
Assoc. Prof. Halife KODAZ
Asst. Prof. Dr. Onur İNAN

In recent years, the general purpose computing on graphical processing unit (GPGPU) have gained a huge popularity in field of parallel computing and the usage of GPGPU has increased to solve many high-dimensional time-consuming problems day by day because central processing unit (CPU) technology approaches to the physical limits in increment processor speed.

The main motivation behind the popularity of GPGPU is the huge amount of data emergent in recent years, time problem in serial programming to process these data and developed lucidity and wieldy tools for parallel programming.

This paper focuses on parallelization of tree-seed algorithm (TSA) which is one of the population-based metaheuristic algorithms, inspired by natural behaviors of trees and their seed production. In this thesis, TSA is implemented on the computer unified device architecture (CUDA) parallel development environment. Experimental results show that when the high-dimensional or high number of trees (population) is used while the problemis solved, parallel implementation of TSA is faster than serial implementation about 184 times.

Keywords: Tree-Seed Algorithm, CUDA, Nature-Inspired Heuristics, Optimization, Parallel Computing, Parallel Programming

ÖNSÖZ

2010 yılında başlamış olduğum Yüksek Lisans eğitimime çeşitli sebeplerden dolayı uzun süre ara vermemin ardından 2015 Şubat'ında danışman hocam Doç. Dr. Mustafa Servet KIRAN'ın "paralel programlama çalışacağız" demesiyle yeniden başlamış oldum.

Çocukluğumdan beri hayatta 3 şeyi severim, "hız, hız, hız" şeklinde bir yaklaşımla yaşamaya çalıştığımdan, sıra beklemekten nefret ettiğimden, işlerin imkanlar dahilinde çok hızlı olması gerektiğini düşündüğümden, az emek çok yemek düsturu ile hareket ettiğimden tez konusu olarak çalıştığım ve bundan sonraki hayatımda da çalışmaya devam etmeyi düşündüğüm GPU ile Paralel hesaplama konusunu zevkle öğrendim ve öğrenmeye devam edeceğim.

Süreçte gerek mesai içerisinde, gerekse sonrasında her daim, her türlü soruma sabırla cevap vererek yeniden akademik hayata ısınmama vesile olduğu için danışman hocam Doç. Dr. Mustafa Servet KIRAN'a özellikle teşekkür ederim.

Yüksek lisans eğitimime ilk başladığım dönemde danışmanlığımı yapan Doç.Dr. Halife KODAZ'a, lisans tezi danışmanım ve süreçte manevi destekleriyle yanımda olan Yrd. Doç. Dr. Mesut GÜNDÜZ'e, kendisine başvurduğum bir konuda detaylı anlatımı ve ilgisinden ötürü Doç.Dr. Sadettin Erhan KESEN'e, Güneysınır MYO'da bir dönem ders vermemi sağlayarak eğitim camiasının içinde yer almam gerektiğini düşünmeme vesile olan ve tez sürecinde birçok fikir veren Öğr. Gör. Mehmet Akif ŞAHMAN'a, orijinal fikirleriyle sürece katkıda bulunan Elektrik Mühendisi H.Oğuz ÖZSOY'a, süreçte her daim yanımda bulunan anneme, babama, eşime, oğlum Ramazan Fadlullah'a ve kızım Zeynep Şüheda'ya teşekkür ederim.

Ahmet Cevahir ÇINAR
KONYA-2016

İÇİNDEKİLER

SİMGELER VE KISALTMALAR

Simgeler

Kısaltmalar

ABC : Yapay Arı Kolonisi (Artificial Bee Colony)
ACS : Karınca Koloni Sistemi (Ant Colony System)
CC-UMA : Ön Bellek Uyumlu Tek Düze Erişim (Cache Coherent UMA)
CMOS : Bütünleyici Metal Oksit Yarı İletken (Complementary Metal Oxide Semiconductor)
CPU : Merkezi İşlem Birimi (Central Processing Unit)
CUDA : Birleşik Hesap Cihazı Mimarisi (Compute Unified Device Architecture)
CURAND : CUDA Rastgele Sayı Üretme Kütüphanesi
DEA : Diferansiyel Evrim Algoritması (Differential Evolution Algorithm)
DES : Doğa Esinli Sezgiseller (Nature Inspired Heuristics)
FLOPS : Saniyede Gerçekleştirilen Kayan Nokta Sayılı İşlem Sayısı(Floating-Point Operations Per Second)
GA : Genetik Algoritma (Genetic Algorithm)
GSP : Gezgin Satıcı Problemi (Travelling Salesman Problem)
GPGPU : Grafik İşlemcisinde Genel Amaçlı Hesaplama(General Purpose Computing on Graphical Processing Unit)
GPU : Grafik İşleme Ünitesi (Graphical Processing Unit)
IPS : Saniyedeki Komut Sayısı (Instructions Per Second)
KKO : Karınca Kolonisi Optimizasyonu (Ant Colony Optimization)
MIMD : Çok Komut Çok Veri (Multiple-Instruction Multiple-Data)
MISD : Çok Komut Tek Veri (Multiple-Instruction Single-Data)
NP-Zor : Deterministik olmayan Polinomsal Zor (Non-deterministic Polinominal Hard)
NUMA : Tek düze olmayan bellek erişimi (Non-Uniform Memory Access)
OpenCL : Açık Hesaplama Dili (Open Computing Language)
PSO : Parçacık Sürü Optimizasyonu (Particle Swarm Optimization)
SA : Benzetimli Tavlama (Simulated Annealing)
SIMD : Tek Komut Çok Veri (Single-Instruction Multiple-Data)
SISD : Tek Komut Tek Veri (Single-Instruction Single-Data)
SP : Akış İşlemcisi(Streaming Processor)
SPMD : Tek Program Çok Veri (Single Program, Multiple Data)
SM : Çoklu Akış İşlemcisi (Streaming Multiprocessor)
SMP : Simetrik Çoklu İşleme(Symmetric Multiprocessing)
ST : Araştırma eğilimi(Search Tendency)
TSA : Ağaç-Tohum Algoritması(Tree-Seed Algorithm)
UMA : Tek düze bellek erişimi (Uniform Memory Access)

1.GİRİŞ

1.1.Teze Giriş ve Amacı

Her geçen gün artan hesaplama ihtiyacını karşılamak için donanım ve yazılım alanında gelişmeler sürmektedir. Teknolojinin gelişimine paralel olarak verilerin çok hızlı bir şekilde artması ve bu verilerle gerçekleştirilecek olan hesaplamaların daha hızlı yapılarak sonuçlarının değerlendirilmesi için son zamanlarda paralel hesaplama ve programlama yaklaşımının önemi gittikçe artırmaktadır. NVIDIA® firması tarafından sunulan bir paralel programlama mimarisi olan CUDA®[1]'nın kullanımı kolay kullanım ve arkasındaki güçlü destekten dolayı gittikçe yaygınlaşmaktadır. Çalışmamızda ülkemizde yeni yeni kullanımı artan fakat dünyada 2006 yılından beri etkin olarak kullanılan bu mimarinin optimizasyon problemlerinin çözümünde kullanılan doğa esinli sezgisel algoritmaların daha hızlı ve daha başarılı sonuçlar üretmesi için gerekli uyarlamanın yapılması sağlanacaktır.

Kaynak araştırması yapılırken görülmüştür ki son bir yılda paralel programlama için CUDA tabanlı 300'den fazla çalışma bulunmasına rağmen, Türkiye adresli yapılmış çalışma sayısı ne yazık ki sayısal olarak oldukça az görünmektedir.

Paralel hesaplama birbirini beklemek zorunda olmayan işlemler için mükemmel bir yöntem olarak karşımıza çıkmaktadır. Bu doğrultuda GPGPU (General Purpose Computing on Graphical Processing Unit – Grafik İşlemcisinde Genel Amaçlı Hesaplama) yaygınlaşmadan önce bilgisayarlar(ortak bellekli ve dağıtık bellekli) birleştirilerek çeşitli paralel işlemler yaptırılmaktaydı. GPGPU ile ilk etapta görüntü işleme alanında aktif olarak kullanılan GPU kartları hesaplamalar için de kullanılmaya başlanmıştır.

Çözümü uzun zaman alan optimizasyon problemlerinin daha makul sürelerde çözümü için yeni bir çığır açılmıştır. Paralel programlama yapmak için CUDA'dan başka ortamlar mevcutsa da CUDA'nın arkasında NVIDIA firmasının güçlü bir şekilde durmasından(birçok programlama ortamının desteklenmesi ve her geçen gün yenilerinin eklenmesi, sürekli güncellenerek eksikliklerinin giderilmesi, yaygın işlemler için kütüphanelerin oluşturulması vb.) dolayı diğer ortamlar çok ön plana çıkamamıştır.

Optimizasyon (en iyileme); bir sistemin tasarlanmasında olası tüm çözümlerin arasından en iyisinin bulunması olarak ifade edilebilir. "En uygun şekle sokma" anlamına gelen optimizasyon, her zaman için hedeflenen bir sonuçtur. Optimizasyon

---

[1] CUDA® NVIDIA® A.Ş'nin sunduğu bir mimari ve teknolojidir.

teknikleri, yapılmakta olan işin en iyi çözümünü ortaya koymak için kullanılır. Bu teknikler kullanılarak ortaya konulan çözüm, en uygun çözüm olarak adlandırılır. Hedef her zaman için en uygun çözümü yakalayabilmektir. Klasik tekniklerle en uygun çözümün elde edilemediği durumlarda en uyguna yakın çözümlerde yeterli olabilir. Bu sebeple yapay zeka tabanlı optimizasyon teknikleri önerilmiştir. Önerilen bu tekniklerde doğal hayat taklit edilerek problemleri çözmek için yeni algoritmalar geliştirilmektedir.

Doğa esinli sezgisel optimizasyon algoritmaları literatürde oldukça çok çalışılan konuların başında gelmektedir. Bu algoritmalar karmaşık problemlerin çözümünde göstermiş oldukları başarılardan dolayı birçok bilim dalında, mühendislik problemlerinde ve askeri uygulamalarda sıklıkla kullanılmaktadır(Haklı, 2013). Bu ilginin altında yatan en önemli sebeplerden birisi havacılık, tıp, mühendislik vb. alanlarda karşılaşılan problemlerin gittikçe karmaşık bir hal alması bu problemlerin çözümünde doğa esinli sezgisel optimizasyon algoritmalarının gösterdiği başarıdır.

Bu algoritmaların esin kaynağı doğadaki varlıkların(canlı ve cansız) özellik ve davranışlarıdır. Çalışmamızda kullandığımız Ağaç-Tohum algoritması sürekli optimizasyon problemlerinin çözümü için ağaçlar-tohumlar arasındaki çoğalma (neslin devamı)ilişkisinden yola çıkılarak önerilmiştir. Algoritmanın iki ana unsuru vardır, keşfetme ve faydalanma. Keşif aşamasında arama uzayına dağılmış rastgele noktalarda yerleşmiş ağaçlar bulunmaktadır, faydalanma aşamasında ise ağaçlar ile aynı özellikteki tohumlar kullanılmaktadır. Ağaç-Tohum algoritması küçük boyutlu problemlerin çözümünde literatürde sıklıkla kullanılan Yapay Arı Kolonisi, Parçacık Sürü Optimizasyonu, Harmoni Arama Algoritması, Ateşböceği Algoritması ve Yarasa Algoritmasından daha iyi sonuçlar üreterek literatürdeki yerini almıştır(Kıran, 2015).

Bu tez kapsamında Ağaç-Tohum algoritmasının paralel bir algoritma olarak gerçeklenmesi sağlanmış ve problemleri çok daha hızlı çözdüğü gösterilmiştir. Paralelleştirme yapılırken adım adım yapılan bazı işlemlerin aynı anda yapılması için çeşitli teknikler kullanılmış ve bunlar ayrıntılı olarak anlatılmıştır.

Ağaç-Tohum algoritmasının paralelleştirilmesi iki ana aşama olarak veri paralelliği (data parallelism) ve görev paralelliği (task parallelism) çerçevesinde gerçeklenmiştir. Veri paralelliği olarak algoritmada her adımda üretilerek kullanılan rastgele sayılar toplu halde GPU belleğine gönderilmiş, her defasında rastgele sayı üretmek için bir kütüphane(CURAND) çağrısı yapmak zorunda olan iş parçacıkları bu iş yükünden kurtarılmış, böylelikle hız kazanımı elde edilmiştir. Algoritmada ağaç ve tohumları temsil eden değişkenler bellekte önceden belirlenmiş alanlara yerleştirilmiş, böylelikle sabit noktalardaki verilere dayanarak sorunsuz bir şekilde eş zamanlı paralel işlemi yapılmıştır. Görev paralelliği çerçevesinde ise kıyas fonksiyonları, ağaçların oluşturulması, tohumların oluşturulması paralel olarak yeniden kodlanmış, adım adım işlenen süreçler eş zamanlı işlem görecek şekilde ayarlanmıştır.

Ağaç-Tohum Algoritması seri ve paralel olarak kodlanmış, 10 farklı kıyas fonksiyonu(Sphere, Ackley, Griewank, Rastrigin, Rosenbrock, Schwefel, Styblinski Tang, Sum of Different Powers, Sum Squares, Zakharov) için D=10, ST=0.1 sabit, ağaç sayıları N=10,20,30,40,50,60,70,80,90,100, yine sırasıyla tohum sayıları önerildiği üzere ağaç sayısının yüzde 20'si olan NS=2,4,6,8,10,12,14,16,18,20 alınarak çalıştırılmış ve yaklaşık **2 ila 184 kat arasında hızlanma** elde edilmiştir.

1.2.Literatüre Katkısı

Son yıllarda doğa esinli sürü zekâsı algoritmaları üzerinde birçok çalışma yapılmakta ve günümüz gerçek dünya problemlerine uygulanmaktadır. Bu algoritmalar en iyi çözümü garanti etmese de en iyiye yakın bir çözümü uygun zaman aralıklarında üretebilmektedir. Çalışmamızın ana amacı bu uygun zamanları olabildiğince kısaltmaktır.

Doğa esinli sezgisel algoritmaların en yeni ve başarılı olanlarından Ağaç-Tohum Algoritması'nın paralelleştirilmesi ile üretilen kaliteli sonuçlara daha hızlı ulaşılması hedeflenmiştir.

Yapılan paralel gerçekleme literatürde sıklıkla kullanılan test fonksiyonlarında test edilerek seri gerçekleme ile kıyaslanmıştır. Paralel yöntemin seri yönteme göre çok daha hızlı sonuca ulaştığı görülmüştür.

1.3.Tezin Organizasyonu

Bu tez çalışması 5 bölümden oluşmaktadır.

Birinci bölümde yapılan çalışma ile ilgili genel bilgiler verilmiş, tanıtılmış, amacı ve önemi anlatılmış ve literatüre katkısından bahsedilmiştir.

İkinci bölümde tez çalışmasında kullanılan Ağaç-Tohum algoritmasının ait olduğu sınıf olan doğa esinli sezgisel algoritmalar ile CUDA tabanlı yapılan çalışmaların literatür özetleri verilmiştir.

Üçüncü bölümde paralel hesaplama, paralel programlama, CUDA, Optimizasyon, Ağaç-Tohum Algoritması, çalışmada kullanılan kıyas fonksiyonlarının seri ve paralel kodlanmış halleri ayrıntılı olarak anlatılmıştır.

Dördüncü bölümde her bir kıyas fonksiyonu için elde edilen sonuçlar ayrı ayrı verilerek, her fonksiyonun sonuçları değerlendirilmiştir.

Son bölümde sonuçlara genel bakış ve ileride yapılacak çalışmalar için önerilere yer verilmiştir.

2. KAYNAK ARAŞTIRMASI

Mevcut problemlerin çözümleri için önerilmiş ve geliştirilmiş birçok doğa esinli sezgisel algoritma mevcuttur. Önerilen yöntemlerin performanslarının artırılması için bu yöntemlerin zamanla geliştirilmiş versiyonları literatüre kazandırılmıştır. Ayrıca mevcut doğa esinli sezgisel yöntemlerin bir arada kullanılarak hibrit ve hiyerarşik yöntemler de geliştirilmiştir. Çalışmamızın amacı CUDA mimarisi ile paralel programlama yaklaşımı olduğundan özellikle son yıllarda bu alanda yapılmış çalışmalar incelenmiş, bu alana nasıl yeni bir katkı sağlayacağımız düşünülmüştür.

2.1.Doğa Esinli Sezgisel Algoritmalar Kullanılarak Yapılmış CUDA Tabanlı Çalışmalar

Mussi ve ark. (2011) PSO algoritmasını paralelleştirerek CUDA tabanlı uygulamasını geliştirmiştir. Ayrıca gelişmiş sürücü yönetim sistemi problemlerinden yol sinyal algılama için bir uygulama yapmışlardır.

Solomon ve ark. (2011) çalışmalarında görev eşleştirme(Task Matching) problemini PSO algoritması ile CUDA tabanlı uygulamalar ile 37 kat daha hızlı çözmüştür.

Kneusel (2014) çalışmasında PSO algoritması destekli CUDA tabanlı bir uygulama ile 2 boyutlu resimlerin işlenmesi üzerine bir çalışma yapmıştır.

Zhang ve Seah (2012) hareket yakalama üzerine yaptıkları çalışmalarında PSO destekli stokastik arama algoritması ile CUDA tabanlı bir uygulama gerçekleştirmişlerdir.

Platos ve ark. (2012) CUDA ile PSO tabanlı Doküman Sınıflandırma Algoritmasını hızlandırmışlardır.

Janousešek ve ark. (2014) sınıflandırma problemlerinin çözümünde Yapay Arı Kolonisi algoritmasını paralelleştirerek CUDA altyapısı ile uygulama geliştirmişlerdir. Geliştirilen uygulamanın 86.25 kat daha hızlı çalıştığı gözlemlenmiştir.

Rymut ve Kwolek (2015) gerçek zamanlı çoklu görünüm insan pozu izleme için hızlandırılmış PSO algoritmasını GPU üzerinde CUDA tabanlı bir yaklaşımla gerçekleştirmeye çalışmıştır. Çalışmada 1000 parçacıklı ve 10 iterasyon gerçekleştirilen uygulamanın CPU'ya göre 12 kat daha hızlı olduğu ispatlanmıştır.

Yuan ve ark. (2015) doğrusal olmayan sürtünme ile motorlarda bulunan kanatlı disklerin düzenleme optimizasyonunu Benzetilmiş Tavlama, Genetik Algoritma ve Tabu Arama algoritmaları hibritlenerek sunulan Paralel Tavlama Tabu Evrimsel Algoritma ile yapmış, CUDA'nın işleme zamanını kısaltmasının yanı sıra, kanatlı disk sistemi tasarımı için önemli olan titreşim çokluğu azaltılarak yeni ürünler için değerlendirilebilecek bir sonuç ortaya koymuştur.

Bukharov ve Bogolyubov (2015) Sinir ağları ve Genetik Algoritma ile bir karar destek sisteminin geliştirilmesinde CUDA'dan yararlanmışlardır. Karar destek sistemi (decision support system) bir işletmede yöneticilerin ve profesyonel çalışanların karar vermesine yardımcı olarak kullanılan, karar verme sürecinde kullanıcıların sistemle karşılıklı olarak etkileşimde bulunduğu, bilgisayar tabanlı bir bilişim sistemidir. Kararın hızlı verilmesi önemli olduğundan GPGPU ile hızlandırma yapılması düşünülmüş ve kullanılan karar verme parametrelerine bağlı olarak yaklaşık 10 kat hızlanma sağlamışlardır. Çalışmalarında verdikleri bilgiye göre örneğin ağ sayısı 3 iken seri uygulama 1,66 saniyede çözüme ulaşmışken, paralel uygulama 0,58 saniyede çözüme ulaşmıştır. Yine ağ sayısı artırıldıkça gerçeklenmiş seri uygulama süresinin katlamalı olarak arttığından bahsedilerek paralel uygulamanın hız başarısı ortaya çıkarılmıştır.

Wang ve ark. (2015) Bilimsel İş Akışı Zamanlama problemi için GPU tabanlı Paralel Karınca Kolonisi Algoritması önermişlerdir. NVIDIA Tesla M2070 GPU üzerinde çalıştırılan paralel uygulama 1000 görev düğümünü 5 saniyede çözerken, Intel Xeon X5650 CPU üzerinde çalıştırılan seri uygulama 104 saniyede çözüme ulaşmıştır. Çalışmada 20.7 kat hızlanma sağlanmıştır.

Kai ve ark. (2015) CUDA altyapısını kullanarak Paralel Genetik Algoritma ile Grafik Boyama Sorunu'nu çözmeye çalışmışlardır. Grafik boyama sorunu grafik alanında bilinen NP-zor kombinatoryal optimizasyon problemidir. Hızlı bir şekilde uygulanabilir bir çözüm bulmak önemli olduğundan CUDA ile paralelleştirme yapılmış, hesaplama süresinde şimdiye kadar önerilen uygulamalardan çok iyi olduğu ortaya koyulmuştur. Ayrıca grafik boyutu büyüdükçe ortaya tartışmasız bir üstünlük çıktığı da belirtilmiştir.

Kalivarapu ve Winer (2015), dijital feromon ile parçacık sürü optimizasyonunu grafik donanımlar kullanarak hızlandırma çalışması yapmıştır. Çalışmalarında CUDA'yı sadece OpenCL derlemesi yapmak için kullanmışlardır. Ackley(D=10),Ackley(D=20),Sum of Squares(D=30) ve Griewank(D=50) kıyas fonksiyonları ile yapılan testlerde 2 ile 5 kat arasında hızlanma elde edildiği ortaya koyulmuştur. Hızlanmanın yetersiz gibi görünmesinin sebebinin kullanmış oldukları grafik kartların(Quadro 4600 ve 5800) eskiliğine ve kapasitesine dayandığını açıklayan araştırmacılar, daha güçlü kartlar ile çok daha hızlı sonuçlar üretilebileceğini bildirmişlerdir.

Silva ve Bastos Filho (2015), düşük gecikmeli bellek kullanan GPU'lar üzerinde PSO’yu daha verimli hale getirme uygulaması yapmıştır. GPU’nun paylaşımlı(shared) belleğini kullanarak PSO’daki her bir parçacığı bir thread olarak değerlendirip her block’taki threadleri alt sürü(sub-swarm) olarak ayarlayıp çalışma gerçekleştirilmiştir. 32 parçacıklı ve 32 boyutlu 8 alt sürü tanımlanmıştır. Bu şekilde seri uygulamaya göre 100 kat hızlanma sağlanmıştır.

Akgün ve Erdoğmuş (2015), genetik algoritmalar kullanarak görüntü evrişim filtresi ağırlıklarının eğitimini CUDA ile hızlandırmışlardır. 256x256, 512x512,1024x1024 boyutlu resimler kullanarak seri, direk paralel metod, popülasyon tabanlı paralel metod, blok tabanlı paralel metod ve alt resimler tabanlı paralel metod olarak ayrı ayrı çalıştırılmıştır. 55 ila 90 kat hızlanma elde edilmiştir.

Zarrabi ve ark. (2015), yerçekimsel arama algoritması kullanarak CUDA ile yüksek performanslı metasezgisellere yönelik örnek olay çalışması yapmışlardır. Sphere, Rastrigin, Rosenbrock ve Griewank kıyas fonksiyonları ile farklı paralel yaklaşımlar test edilmiş yaklaşık 3 ila 28 kat arasında hızlanma elde edilmiştir.

Tsuchida ve Yoshioka (2015), sinir ağı öğrenme için bir paralelizasyon yöntemi önermiştir. Çalışmada sinir ağı öğrenme, akıllı sinyal işlemenin bir biçimi şeklinde uygulanmıştır. Sinir ağının öğrenme süreci paralel uygulama ile hızlandırılmıştır. Normal metodla kıyaslandığında parametrelere göre 3 ila 6 kat hızlanma sağlanmıştır.

Ouyang ve ark. (2015), bir boyutlu ısı iletim denklemi için CUDA ile paralel hibrid PSO algoritmasını önermiştir. İki farklı GPU’da yapılan testler sonucunda 12,12 ve 13,06 kat hızlanma elde edilmiştir.

Bukata ve ark. (2015), CUDA platformu için tasarladıkları paralel Tabu Arama algoritmasını kullanarak kaynak kısıtlı proje çizelgeleme problemini çözmüşlerdir. Çözüm esnasında CPU’daki çekirdekleri de işin içine katarak problemin çözümü hızlandırılmaya çalışılmış böylece 4 kat hızlanma elde edilmiştir. GPU ile hızlandırma işleminde ise 50 kat hız elde edilmiştir.

Peker ve ark. (2015), anestezik derinlik düzeylerinin hızlı ve otomatik sınıflandırılması için sinir ağları kullanarak GPU tabanlı parallelleştirme yapmışlardır. Anestezik derinlik düzeyinin belirlenmesi için kullanılması gereken çok büyük ve kompleks bir veri seti olduğundan dolayı CUDA ile yapılan işlemler hızlandırılmıştır.

Lastra ve ark. (2015), aşırı yüksek boyutlu optimizasyon problemleri için yüksek performanslı memetik algoritmayı CUDA ile paralleştirerek 100000, 500000, 1000000, 1500000, 3000000 boyutlu problemlere çözüm aramışlardır. 18 günde çözülebilen 3000000 boyutlu problem 2,5 saatte başarıyla çözülmüştür.

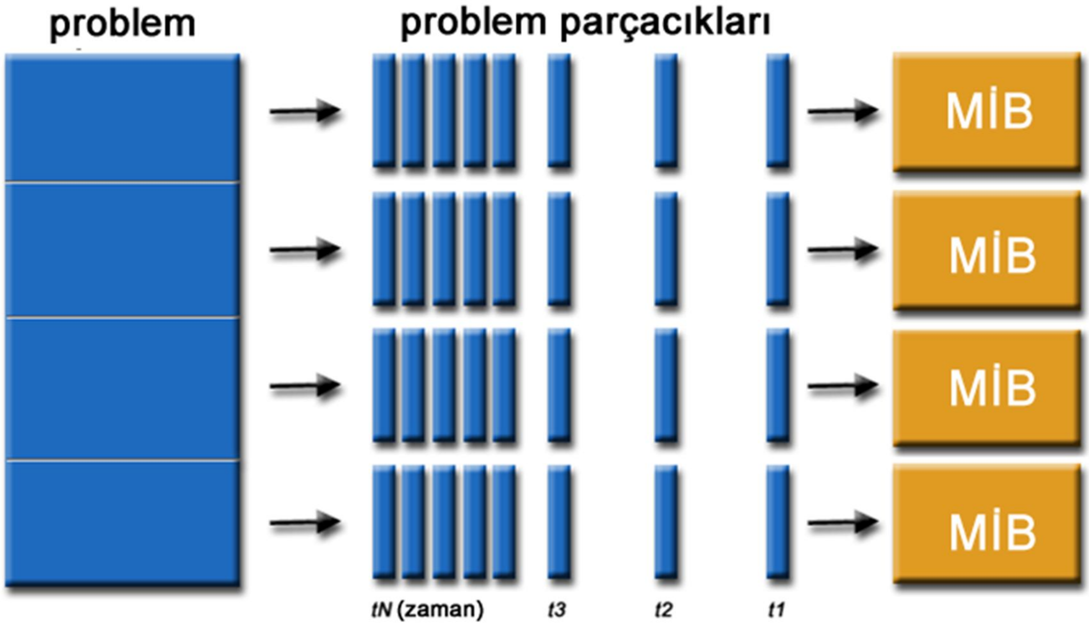
Kıran (2016), Ağaç-Tohum algoritmasını kısıtlı optimizasyon problemlerinin iyi bilinenlerinden olan basınç tankı tasarımı için uygulayarak elde ettiği sonuçları Yapay Arı Kolonisi ve Parçacık Sürü Optimizasyonu ile karşılaştırmış, sonuç olarak uygun ve karşılaştırılabilir sonuçlar üretildiğini göstermiştir.

3.MATERYAL VE YÖNTEM

3.1.Paralel Hesaplama

3.1.1.Paralel hesaplamaya giriş

Paralel hesaplama, daha hızlı sonuç almak için bir uygulamaya ait program parçalarının birden fazla işlemcide aynı anda çalıştırılmasıdır. Hesaplama ihtiyaçları, gün geçtikçe artmaktadır. Daha yüksek frekanslı sensörler, görselleştirme kalitesinin artması, dağıtık veri tabanları üzerinde yapılan işlem yoğunluğu buna birer örnektir. Diğer taraftan işlemci teknolojisi fiziksel limitlerine (termodinamik, ışık hızı, CMOS transistörler) yaklaşmaktadır. Bundan dolayı paralel hesaplama ve araçlarının popüleritesi artmaktadır ve ağ teknolojilerindeki hızlı gelişmeler(dağıtık bellekli sistemlerin iletişimi için) paralel hesaplama için kolay edinilebilir ve ulaşılabilir donanımlara izin vermektedir(Temirci, 2009).Şekil 3.1'de paralel hesaplama benzetimi gösterilmiştir.



Şekil 3.1. Paralel Hesaplama Benzetimi

Paralel hesaplama işlemlerin zamansal maliyetini azaltırken bazı ek yüklerin yazılımcı veya sistem için oluşmasına neden olmaktadır. Bunlardan bazıları aşağıda sıralanmıştır(Sarıman, 2012).

- İşlemcilerde fazladan geçen süre
- İletişim ek yükü

- Senkronizasyon ek yükü
- Programın paralel olmayan/olamayan parçaları

Paralel programlamada ek yük ve çalışma zamanı hızlanma ve verimlilik ile ifade edilir(Sarıman, 2012).

Örneğin iki sayının toplanması;

Seri (Tek işlemcili)

=1 + 2 + 3 + 4 + 5 + 6 + 7 + 8

=3 + 3 + 4 + 5 + 6 + 7 + 8

=6 + 4 + 5 + 6 + 7 + 8

=10 + 5 + 6 + 7 + 8

=15 + 6 + 7 + 8

=21 + 7 + 8

=28 + 8

=36

Paralel (2 işlemcili)

=1 + 2 + 3 + 4 + 5 + 6 + 7 + 8

=3 + 3 + 4 + 5 + 6 + 15

=6 + 4 + 5 + 21

=10 + 26

=36

7/4=1,75 kat hızlanma sağlanır. Çok fazla hesaplama gerektiren, bitirilmesinde zaman kısıtları olan ve özellikle n adet parçacığa bölünebilen görevler için paralel hesaplama mükemmel bir çözüm olarak göze çarpmaktadır(Altıntaş ve Yegenoğlu, 2011).

Eğer bir insan bir çukuru bir dakikada kazıyorsa, 60 insanın bir çukuru bir saniyede kazması gerekir. Fakat işler bu şekilde yürümemektedir. Pratikte lineer hızlanmayı (işlemci sayısıyla orantılı) başarmak çok zordur. Bunun nedeni, doğada birçok algoritma aslında sıralıdır (Amdahl yasası bunu bilimsel olarak açıklamıştır)(Sayar ve Ergün, 2014). Ekstra işlemciler eklendikçe, bazı iş yükleri, boru hattı (pipeline) paralellik kullanarak belli bir noktaya kadar fayda sağlar. Bu sistem, bir fabrika montaj hattı yaklaşımı kullanarak işleri parçalar. Eğer iş n aşamaya bölünebiliyorsa ve bir ayrık değişken bir aşamadan diğer birine iletilebiliyorsa, en fazla n adet işlemci kullanılabilir. Bununla birlikte, en yavaş aşama diğer aşamaları da tutacaktır ve n işlemciyi tam performansta kullanmak pek mümkün olmayacaktır.

Pek çok algoritma, paralel donanımın kullanımını daha verimli yapmak için tekrar tasarlanmalıdır. Tek işlemcili sistemlerde iyi çalışan programlar, paralel sistemlerde aynı performansı vermeyebilir. Aynı programın çoklu kopyaları, birbirlerini etkileyebilirler (aynı anda aynı hafıza adresine yazma/okuma yapma). Bu yüzden paralel sistemlerde algoritmalar iyi analiz edilmeli ve dikkatli programlama yapılması gerekir.

1965 yılında Gordon Moore tarafından formüle edilen ve Moore Yasası olarak literatüre geçen yasaya göre devre karmaşıklığı her 18 ayda(Moore, 1975 yılında degerlendirmesini bekleme zamanını 18 aydan 24 aya çıkararak düzeltti) iki kat artar. Bu aynı artışın işlemci performansı için de geçerli olduğu anlamına gelmektedir. İşlemci performansını ister IPS (Saniyedeki Komut Sayısı – Instructions Per Second) ile ölçelim, ister FLOPS (Saniyede Gerçekleştirilen Kayan Nokta Sayılı İşlem Sayısı – Floating-Point Operations Per Second) ya da işlemcilerin performansını gerçek uygulamalar üzerinde ölçmeye çalışan güçlü benchmark yazılımlarını kullanalım Moore Yasasının geçerli olduğu şimdiye kadar görülmüştür(Parhami, 2006). Buna rağmen Moore Yasası'nın önümüzdeki 10 yıl içinde tarihe karışacağı ile ilgili öngörülerde bulunmaktadır.

Yüksek performanslı süper bilgisayarların savunma sanayi, uzay araştırmaları ve iklim modelleme uygulamalarında kullanıldığı bilinmektedir. Bunların yanında araç kaza benzetim uygulamaları, ilaç tasarım uygulamaları, tümleşik devre tasarım uygulamaları, bilimsel görselleştirme ve çoklu medya uygulamaları örnek olarak verilebilir. Var olan süper bilgisayarların performansının yetmediği bir uçağın aerodinamik benzetimini yapan uygulamalar, küresel klimayı onlarca yıl modelleyen uygulamalar ve gelişmiş maddelerin atomik yapılarını inceleyen uygulamalar örnek olarak sayılabilir(Parhami, 2006).

Genel olarak bir uygulama şu üç sebepten dolayı birden daha fazla hesaplama gücüne ihtiyaç duyar(William ve ark., 2003):

1. Gerçek zamanlı kısıtlar: Buna örnek olarak hava tahmini programları verilebilir. Pazartesi günü ile ilgili tahmini salıya yetiştirmenin herhangi bir pratik faydası yoktur. Bir diğer örnek olarak bir deneyde üretilen verilerin en azından üretildiği hızda işlenmesi (veya saklanması) gerekmektedir ki daha sonraki analizlerde bu verilerin ve dolayısıyla deneyin bir anlamı olmuş olsun.

2. Verimlilik: Verimlilik (throughput) belirli bir zaman periyodunda bir bilgisayarın yapabileceği iş miktarıdır. Bazı simülasyonların hesaplama gücüne o kadar çok ihtiyacı vardır ki hesaplamayı bir bilgisayar ile bitirmeye çalışmak günler hatta aylar sürebilir.
3. Bellek: Bazı simülasyonların bir bilgisayarın sınırlarını kat be kat aşan bellek ihtiyaçları vardır.

Daha yüksek performanslı donanımlar kullanarak, problemleri çözen algoritmaları optimize ederek ve/veya paralel hesaplama yöntemleri kullanmaya imkân sağlayan paralel bilgisayarlar kurarak daha fazla işlem gücü elde edilebilir.

Dağıtık bellekli paralel hesaplama bir bilgisayarın kabiliyetlerini aşan karmaşık bir problemi çözmek için birden çok hesaplama kaynağının aynı anda kullanılmasını kapsayan bir hesaplama biçimidir. Hesaplama kaynakları,

- birden çok işlemciye sahip bir bilgisayarı,
- birbirleri ile bağlı birden çok bilgisayarları (paralel bilgisayar) veya
- her ikisinin birleşimini içerebilir.

3.1.2.Paralel hesaplama yöntemleri

3.1.2.1.Çok çekirdekli hesaplama (Multicore computing)

Günümüzde de yaygınlaşan çok çekirdekli işlemcilerin içerisinde birden fazla işlemi (komut setini) aynı anda yapmaya izin veren birden fazla denetim birimi (çekirdek) vardır. Bu denetim birimleri ortak bellekleri kullanırlar (Temirci, 2009).

Çok çekirdekli işlemcilerdeki avantajı kullanabilecek bir işletim sistemi ve uygun kodlarla her bir çekirdek ayrı bir işlemci gibi kullanılabilir ve paralel hesaplama yapılabilir. Ancak bu sistemin bazı dezavantajları da mevcuttur. İşletim sistemi ve bu işletim sisteminde kullanılan yazılımlar, bu işlemcilerden maksimum verim alabilmek için, çok çekirdek mimarisini destekleyecek şekilde olmalıdır (Temirci, 2009).

3.1.2.2.Simetrik çoklu işleme (Symmetric multiprocessing)

Bu yöntemde, bir bilgisayar sisteminde veri yolları ile bağlı bir hafızayı paylaşan (shared memory) birden çok özdeş işlemci kullanımı esastır. Bugün yaygın olarak kullanılan SMP mimarisidir. SMP sistemleri işlem için gerekli olan verinin hafızanın neresinde olduğuna bakmaksızın herhangi bir anda yapılmasına imkân sunar. Fakat bu yöntemde birtakım veri yolu problemleri (bus contention) sebebiyle genellikle işlemci sayısı bakımından sınırlıdır. Ölçeklenebilirliği yüksek değildir. Çok yüksek hızlara ihtiyaç duyan uygulamalarda yetersiz kalırlar.(Temirci, 2009)

3.1.2.3.Dağıtık hesaplama (Distributed computing)

Dağıtık hesaplama yönteminde ise işlem parçaları, ağ (network) üzerinde birbirine bağlı işleme ünitelerine dağıtılır. Bağlantı bu şekilde ağ üzerinde olduğundan son derece ölçeklenebilir bir sistemdir. Dağıtık hesaplama sistemleri de kendi içinde üç ana kısma ayrılır: (Temirci, 2009)

- Cluster Computing (Bilgisayar Kümeleri)
- Massive Parallel Processing (Güçlü Paralel İşleme)
- Grid Computing (Izgara Hesaplama)(Temirci, 2009)

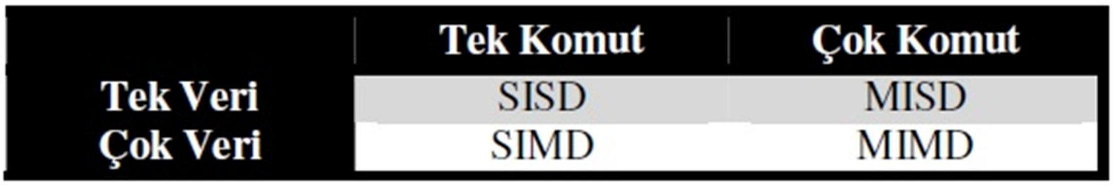
Paralel hesaplamada süreç(process) kavramı önemlidir. Süreç bir fiziksel işlemci üzerinde çalışan özerk bir program veya altprogramdır. Bir sürecin eriştiği yerel bir depolama alanı vardır. Bir program çalışma zamanının herhangi bir anında birden çok süreci kapsar hale geliyorsa söz konusu program paralel bir programdır(Pacheco, 1997).

Verimli bir paralel hesaplama için hem donanım hem de yazılım kabiliyetlerine ihtiyaç vardır. Süreçleri belirleyen, oluşturan ve yok eden yollar olmalıdır. Süreçler arasındaki iletişimi belirleyen protokoller sağlanmalıdır. Süreçler arasındaki karşılıklı bağlantılar bu süreçler arasındaki iletişimi hızlı bir şekilde yerine getirmelidir.

Paralel hesaplamayı gerçekleştirmek için çok çeşitli donanım mimarisi ve bu mimarileri destekleyen yazılım modelleri mevcuttur. Donanım mimarilerini mantıksal bir sıraya sokmak zor olabilir. Ama yine de bir yerden başlamak gerekirse bilgisayar mimarilerini sınıflamak için Michael J. Flynn tarafından 1966 yılında ortaya konan ve Flynn Taksonomisi olarak bilinen sınıflamadan başlanabilir.

3.1.3.Flynn taksonomisi

Bu taksonomide bilgisayar mimarileri komut ve veri olmak üzere iki bağımsız boyut üzerinden sınıflanırlar. Bu boyutlardan her biri tek veya çok olmak üzere iki olası durumda olabilir. Dolayısıyla ortaya dört durum daha doğrusu dört sınıf çıkmaktadır (Kaçka, 2011). Şekil 3,2'de Flynn taksonomisi görülmektedir.



	Tek Komut	Çok Komut
Tek Veri	SISD	MISD
Çok Veri	SIMD	MIMD

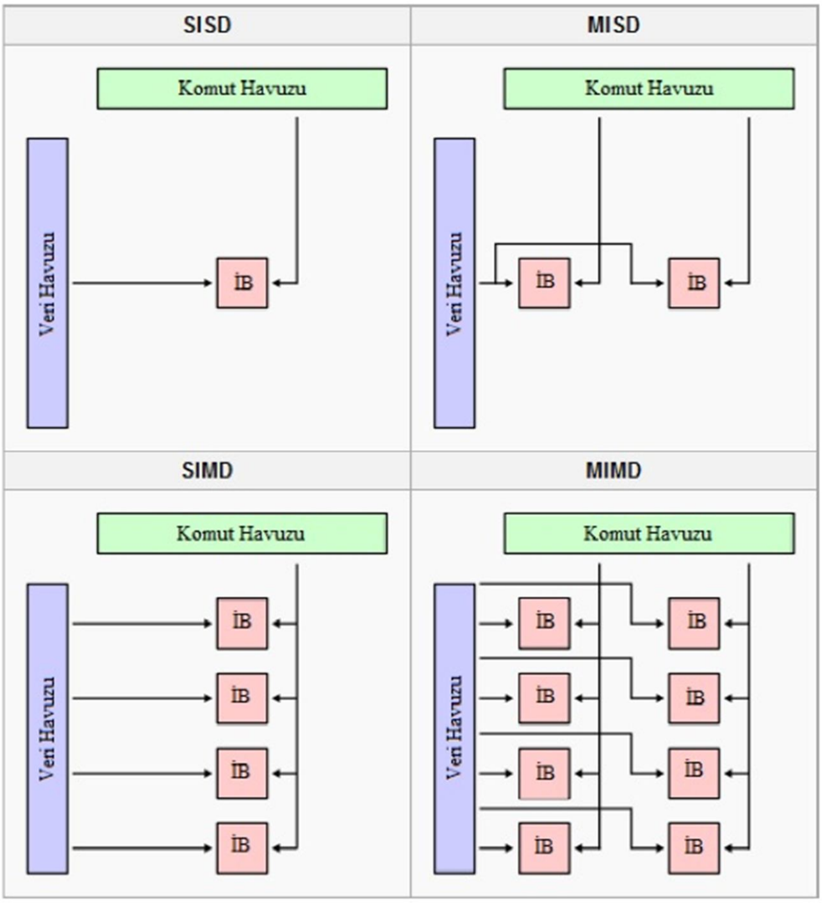
Şekil 3.2. Flynn taksonomisi(Kaçka, 2011)

Bu taksonomideki en basit mimari SISD (Single-Instruction Single-Data) mimarisidir. Buna örnek olarak klasik von Neumann makinası gösterilebilir (Kaçka, 2011).

MISD (Multiple-Instruction Single-Data) sistemleri genellikle hata toleranslı uygulamalarda kullanılırlar. Bu sistemlerde heterojen sistemler aynı veri üzerinde çalışırlar. Buna örnek olarak uzay mekiği uçuş kontrolü bilgisayarı verilebilir (Kaçka, 2011).

SIMD (Single-Instruction Multiple-Data) sistemlerde ise tüm işlem birimleri (processing unit) herhangi bir saat döngüsünde senkron bir şekilde aynı komutu işlerler. Bütün işlem birimleri bir kontrol birimi tarafından kontrol edilirler(Ananth ve ark., 2003). Her işlem birimi farklı veri elemanını işleyebilir. Bu makinelerin iki çeşidi vardır: işlemci dizileri (Connection Machine CM-2, ILLIAC IV) ve vektör iş hatları (IBM 9000, Cray X-MP, ETA 100). Grafiksel işlemci birimi (GPU) olan çoğu bilgisayar bu mimariyi içinde barındırır (Barney, 2010).

MIMD (Multiple-Instruction Multiple-Data) mimarisi kendi verileri ile işlem yapan özerk işlemcilerden oluşur. Dağıtık sistemler genelde bu sınıfa girerler. SIMD sistemleri eşzamanlı çalışırken MIMD sistemleri asenkron bir yapıya sahiptirler (Kaçka, 2011). Şekil 3.3'de Flynn taksonomisinin işlem birimleri ve veri havuzu/komut havuzu açısından gösterimi görülmektedir.



Şekil 3.3. Flynn taksonomisinin işlem birimleri ve veri havuzu/komut havuzu açısından gösterimi(Kaçka, 2011)

3.1.4.Amdahl yasası

Amdahl Yasası verilen bir algoritma paralelleştirildiğinde teorik olarak en fazla ne kadar performans kazancı elde edilebileceğini tanımlayan ve büyük ihtimalle en çok bilinen ve başvurulan kanunlardan bir tanesidir. Bu yasa 1967 yılında Gene Amdahl tarafından paralel hesaplamanın tartışıldığı bir konferansta paralel hesaplamaya karşı delil olarak sunulmuştur. Bugün paralel hesaplamayı açıklamak için kullanılsa da delil ortaya koyulduktan 40 sene sonrasına kadar seri performans artışı devam ettirebildiği için zamanında haklı olduğu söylenebilir (Sayar ve Ergün, 2014).

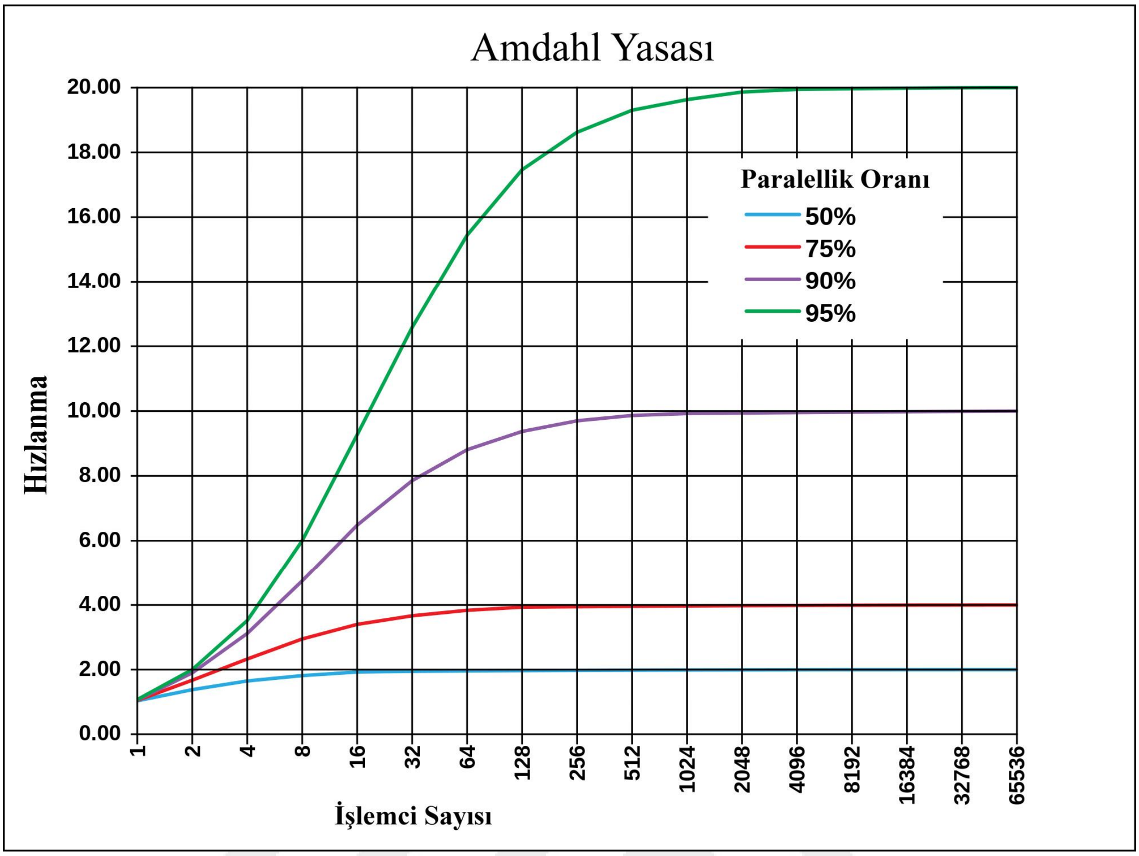
Eğer f kadarı seri olan bir problem n tane işlemcide çalışıyorsa maksimum performans kazancı pk

$pk = 1 / (f + (1-f) / n)$

olacaktır. Hatta bu sistemde sonsuz adet işlemci olduğunu düşünürsek

$pk = 1 / f$

olacaktır. Yasaya göre performans artışının nasıl değiştiği Şekil 3.4 'de görülebilir (Sayar ve Ergün, 2014).



Şekil 3.4. Amdahl yasasına göre performans artışı

Ancak asimetrik paralel mimariler kullanıldığında Amdahl yasasının öngördüğünden daha iyi bir performans artışı sağlanabilmektedir. Örneğin 64 tane aynı işlemciden kullanmak yerine 60 tane aynı işlemci kullanmak ve bir tane bu işlemcilerden iki katı hızlı bir işlemci kullanıp, hızlı işlemciye programın seri kısmını işletmek daha performanslı olacaktır(Vajda, 2011).

Bu durum uygulamamızı geliştirdiğimiz NVIDIA CUDA destekli aygıtlara ve mimariye işaret etmektedir.

Amdahl yasası genel olarak yanlış olmamakla beraber Amdahl'ın paralel programlamanın anlamsız olması sonucunu çıkarırken yaptığı iki hatalı varsayım vardır. Birincisi o zamanda paralel bilgisayarlar mevcut olmadığı için programcılar programları paralelleştirmek için çaba harcamamışlardır. Bu tip bilgisayarlar ortaya çıkınca programların paralel kısımlarını arttıracak yöntemler geliştirmeye başlamışlardır. İkincisi ise problemin büyüklüğü değiştikçe paralel ve seri kısımların toplam işletme hızları eşit hızlı büyümemektedir. Bu nedenle problemin boyutu büyüdükçe f azalacak bu sayede performans yükselecektir. Amdahl yasası problemlerin boyutunu sabit kabul ederek yanlış bir varsayımda bulunmuştur. Amdahl asıl olarak f'i 0.25 ila 0.4 olarak tahmin etmiştir günümüz f bu değerin çok çok altındadır. Maksimum performans kazancı genelde problemin büyüklüğü ile orantılıdır (Sayar ve Ergün, 2014).

3.1.5.Paralel program tasarımı

Paralel programları tasarlamak ve geliştirmek genelde programcı kontrollü bir süreç şeklinde ilerler. Paralelleştirmeyi belirlemek ve gerçekleştirmek genelde programcının sorumluluğundadır. Bu süreç zaman alıcı, karmaşık, hataya yatkın ve tekrarlı bir süreçtir. Bunun yanında zaman içinde programcının işini kolaylaştıran bazı araçlar geliştirilmiştir (Barney, 2010).

Paralel program geliştirirken ihtiyaçlardan biri hesaplama algoritmalarının ve giriş verilerinin eş zamanlı çalışabilecek altprogramlara nasıl ayrıştırılabileceğini belirlemektir. Bu sürece problem ayrıştırması (problem decomposition) denir.

İki çeşit ayrıştırma söz konusudur:

1. Alan ayrıştırması
2. Fonksiyon ayrıştırması

Çoğu program bu iki çeşidin bir karışımından ibarettir (Pacheco, 1997) .

3.1.5.1.Alan ayrıştırması

Alan ayrıştırmasında (domain decomposition) veriler aşağı yukarı aynı parçalara ayrılırlar ve süreçler ile eşleştirilirler. Her süreç sadece kendisine atanmış veri kısmı ile çalışır. Her süreç genelde veri üzerinde aynı komutları çalıştırır. Bu yaklaşım veri paralelleştirilmesi (data parallelism) olarak da bilinir. Süreçler belli aralıklarda birbirleri ile haberleşip veri alışverişinde bulunurlar (Kaçka, 2011).

SPMD (single program, multiple data) yaklaşımı alan ayrıştırmasına bir örnektir. Burada tüm süreçlerde aynı kod çalışır. Bu tarz yaklaşımlar işlemcilerin geniş veri kısımlarında birbirinden bağımsız çalışabileceği sonlu fark algoritmalarında kullanılırlar (Kaçka, 2011).

3.1.5.2.Fonksiyon ayrıştırması

Bazı karmaşık problemlerde tek başına alan ayrıştırması verimli bir strateji olmayabilir. Örneğin, süreçlere atanmış farklı veri altkümeleri için farklı zamanlar gerektirmesi durumunda kodun performansı en yavaş süreç ile sınırlıdır. Bu durumda fonksiyon ayrıştırması (functional decomposition) faydalı olur. Görev paralelleştirilmesi (task parallellism) olarak da bilinen bu yaklaşımda problem çok sayıda alt göreve (fonksiyona) bölünür ve hangi süreç müsait olursa ona atanır. Yönetici/işçi (manager/worker) olarak da bilinen bu yaklaşım bir istemci sunucu örneği şeklinde uyarlanır. Burada bir ana süreç (efendi veya yönetici süreç) işi parçalara böler ve bu parçalar işçi (istemci) süreçlere atanırlar (Kaçka, 2011).

3.1.6.Paralel hesaplamanın avantaj ve dezavantajları

Paralel hesaplamanın en önemli avantajı şüphesiz problemin çözüm zamanını azaltarak çözüme daha hızlı bir şekilde ulaşılmasını sağlamaktır. Problem bir parça olarak değil parçalar halinde ve her bir parçanın belirli zaman aralıklarına bölünmüş olmasıyla daha kolay ve hızlı çözüm elde edilmesi sağlanır (Akçay ve Erdem, 2010).

Paralel hesaplamanın avantajları;

- Boşta olan kaynakları kullanarak işlerin yürütülebilirliğini arttırmak
- Hızlı uygulamalar sayesinde işlemler hızlanır ve sonuçları daha hızlı elde etmek
- Yeni ve daha fazla işe yarayan uygulamaların geliştirilmesini hızlandırmak
- İşbirliği ve üretkenlik kapasitelerinde artışlar sağlamak
- Kullanıcıya güçlü tek makine kullanıyormuş gibi bir ara yüz sağlanarak kullanımının kolaylaşmasını sağlamak.
- Benzer konuda çalışan araştırmacıların sanal organizasyonlarda bir araya gelmesini sağlamak.

Paralel hesaplamanın en önemli dezavantajı ise parçalara ayrılan problemlerin her birinin ayrı bilgisayarlarda veya CPU’larda çalıştırıldığında parçaların herhangi birinde oluşabilecek problemlerin tüm sistemin başarıya ulaşmasını engellemesi olarak belirtilebilir (Akçay ve Erdem, 2010).

3.2. Paralel Programlama

Paralel bilgisayarlarda programlama için işletim sistemi seviyesinde ve programlama dili seviyesinde pek çok yazılım sistemi geliştirilmiştir. Bu sistemler, problemin parçalara bölümlenmesini ve işlemcilere atanmasını sağlayan çeşitli mekanizmalar içermektedir. Dolaylı paralellik (implicit parallelism) derleyici ya da diğer programın problemi bölümlemesi ve işlemcilere otomatik olarak atamasıdır. Doğrudan paralellik (explicit parallelism) ise programcının problemin nasıl bölümleneceğini bildirmesidir. Yük dengeleme, ağır yük ile çalışan işlemciden bazı görevleri daha hafif yük ile çalışan işlemcilere taşıyarak hepsinin aynı meşguliyette olmasını sağlar.

Bazı insanlar paralel programlamayı, eş zamanlı (concurrent) programlama ile eş anlamlı kabul ederler. Bazıları da kesin bir çizgiyle ayırırlar. Paralel programlama, işlemler arasında iyi tanımlanmış iletişim yapıları kullanan ve verimi artırmaya yönelik işlemlerinin paralel işlenmesini sağlayan bir yapıdır. Eş zamanlı (concurrent) programlama, performanstan ziyade başka nedenlerden dolayı eş zamanlı işlemler arasında yeni iletişim tekniklerine dayanan bir yapıdadır. İşlemler arası iletişim genelde paylaşımlı hafıza veya mesaj geçirme tekniği ile yapılır.

3.2.1.Paralellik türleri

Paralellik işlemci içinde ve işlemci dışında gerçekleştirilebilir. Bilgisayar sistemlerinde paralellik, bit düzeyinde paralellik (Bit-level parallelism), komut düzeyinde paralellik (Instruction-level parallelism), veri paralelliği (Data parallelism) ve görev paralelliği (Task parallelism) olmak üzere 4 sınıfa ayrılmaktadır. İşlemci içindeki paralellik bir komutun evreleri veya komutlar arası ölçekte olmaktadır. Bu yüzden bit ve komut düzeyi paralellik işlemci içinde gerçekleşmektedir. Veri ve görev paylaşımlı paralellik ise işlemci dışında gerçekleşmektedir. İşlemci içi paralellik maliyet ve kullanıcı müdahale sınırlaması vardır. Esnek ve ekonomik değildir (Akçay ve ark., 2011).

3.2.1.1.Bit düzeyinde paralellik (Bit level parallelism)

İşlemci içindeki bir saat dalgasında işlenebilen bit miktarı bit düzeyindeki paralellik düzeyini göstermektedir. Eski nesil işlemcilerde bir saat dalgasında 4 bit veri işlenebiliyorken günümüzde ise 512 bit veriyi aynı anda işleyebilen grafik işlemciler(Nvidia GTX280, GTX285, Quadro FX 5800) mevcuttur.

3.2.1.2.Komut düzeyinde paralellik (Instruction-level parallelism)

Bir bilgisayar programı, özünde, bir işlemci tarafından yürütülen komut evresi akışıdır. Bu akış şekliyle programın sonucu değiştirmeden komutlar, paralel olarak yürütülebilir. Komut düzeyinde paralellik aynı anda birden fazla komutun işlenebilmesi temeline dayanır. Bu işlemci içinde iş hattı (pipeline) teknolojisiyle gerçekleştirilir. (Akçay ve ark., 2011)

3.2.1.3.Veri düzeyinde paralellik (Data parallelism)

Veri paralelliği paralel olarak işlenecek farklı işlem düğümleri arasında veri dağıtımını odaklanan bir yöntemdir. Veri düzeyinde paralellik program döngülerin yapısında karşımıza çıkar. Her işlemci dağıtılmış verilerin farklı parçaları üzerinde aynı görev yaptığında veri paralelliği sağlanır. (Akçay ve ark., 2011)

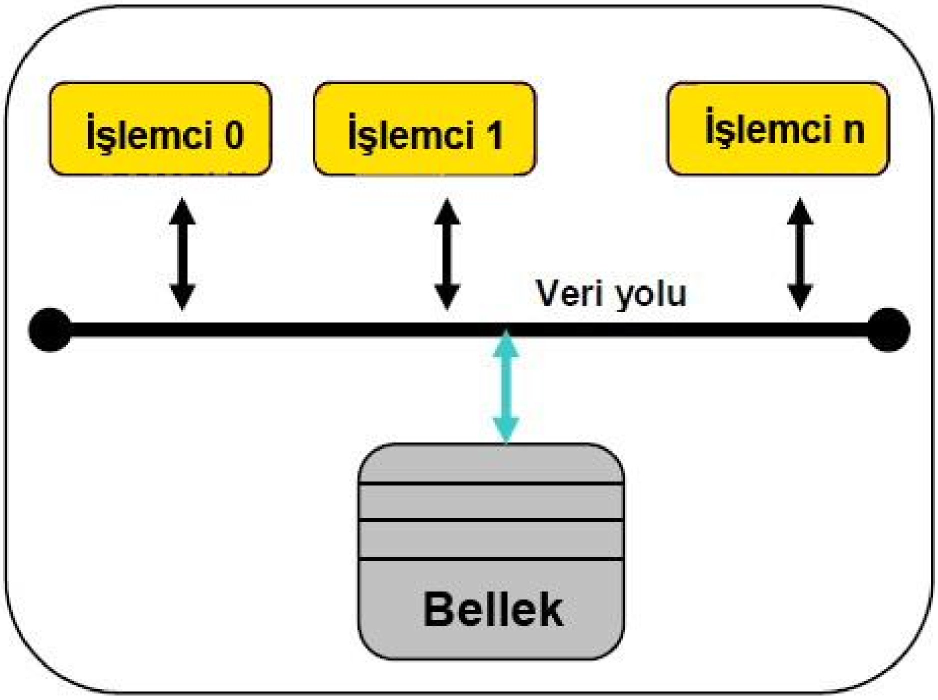
3.2.1.4.Görev paylaşımlı paralellik (Task parallelism)

Görev paylaşımlı paralel sistemlerde her fonksiyon farklı işlemciler üzerinde çalışır. Bu yüzden görev paylaşımlı paralel sistemler ortak bellekli (shared memory) ve dağıtık bellekli (distributed memory) olmak üzere iki ana sınıfa ayrılmaktadır. Görev paylaşımlı paralel sistemler CPU ve GPU üzerinde denenebilme yetisine sahiptir. (Akçay ve ark., 2011)

3.2.2.Paralel sistemlerin sınıflandırılması

3.2.2.1.Ortak bellekli sistemler

Paylaşımlı bellekli sistemler birbirinin tamamıyla aynı olan çok sayıda işlemciden oluşur. Bu işlemciler aynı saat frekansı ile yürütülürler ve aynı belleği paylaşırlar. Şekil 3.5'de paylaşımlı bellekli paralel sistem mimarisi görülmektedir.



Şekil 3.5. Paylaşımlı bellekli paralel sistem mimarisi

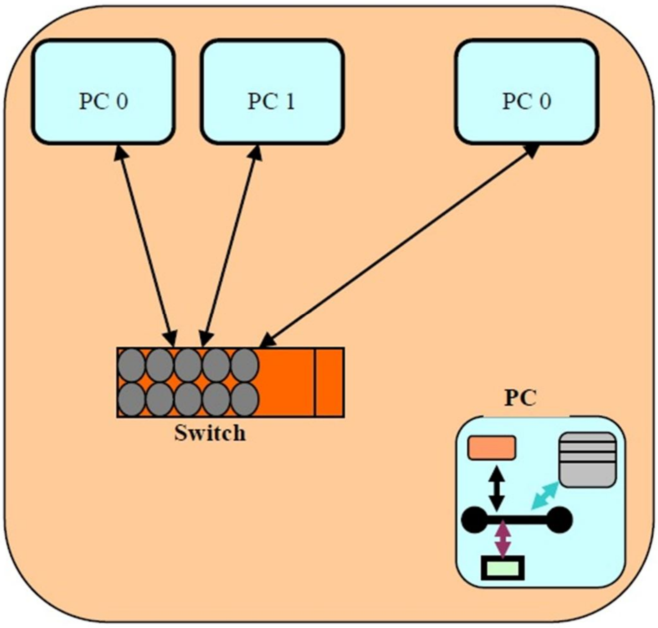
İşlemciler arasında iletişim paylaşılan ortak bellek üzerinden olduğundan veri paylaşımı oldukça hızlıdır. Ortak bellekli paralel sistemler hem CPU üzerinde hem de GPU üzerinde gerçekleştirilebilmektedir. CPU üzerine kurulu sistemlerde tek işletim sistemi tüm işlemcileri ve belleği yönetir ve bilgisayar sistemi genelde tek bir kasa içinde bulunur. Bir sistemin ortak bellekli yapı olup olmağı donanımsal farklılığı kullanıcı (tek işlemcili sistemlere göre) pek hissetmez. Uygulama yazılımlarının paralelleştirilmesi donanım, derleyici ve işletim sistemi yardımıyla gerçekleştirilir ve kullanıcıya sadece ihtiyaç duyduğu işlemci sayısını belirlemek kalır. GPU üzerinde ortak bellekli sistem gerçekleştirimi ise üretici firmanın paralel programlama desteği veren sürücü (driver) yazılımının kurulumuyla gerçekleştirilir. Bu tür sistemleri kullanmak, bu tür sistemler için yazılım geliştirmek diğer paralel sistemlere göre daha kolaydır. Bu tür sistemlerde programlamaiş parçacığı(thread) temeline dayanmaktadır. Ama bu mimarideki paralel sistemler genelde sınırlı miktarda imal edildiklerinden pahalı, farklı kullanım alanlarına uyumu zor olabilmektedir. CPU üzerinde çalışan ve yaygın olarak kullanılan ortak bellekli programlama yazılımları OpenMP, POSIX Threads ve Win 32/64 Threads olarak gösterebiliriz. GPU için yaygın olarak kullanılan ise OpenMPI, OpenCL, NVIDIA tarafından geliştirilen CUDA mimarisidir (Akçay ve ark., 2011).

3.2.2.2.Dağıtık bellekli sistemler

Dağıtık bellekli paralel sistemler hali hazırda kullanılmakta olan bilgisayarların, örneğin herhangi bir kuruma ait olan bilgisayarların, güçlü bir ağ (Ethernet, anahtar cihazı, kablo) donanımıyla birleştirilmesiyle oluşturulmaktadır. Dağıtık bellekli yapıda, ağa dâhil edilecek bilgisayarların, donanım veya yazılım özelliklerinin birbirleriyle aynı olma gereksinimi yoktur (Akçay ve ark., 2011).

Ağ donanımlarıyla haberleştirilen bu bilgisayarlara paralel yazılımlar (PVM, MPI vb.) kurularak, içlerinden seçilecek bir bilgisayar üzerinde ortam yapılandırılması sağlanmaktadır. Birbirlerine ağ donanımıyla bağlanmış bilgisayarlar ve bu bilgisayarlara yüklenmiş yazılımlarla, süper bilgisayar denilen yüksek performanslı bilgisayar ortamı gerçekleştirilmektedir (Akçay ve ark., 2011).

Dağıtık bellekli bilgisayarlar yüksek iletişim olanağı sağlayan Ethernet kartı, tip A-B kablolama ve güçlü anahtar (switch) cihazıyla birleştirilmektedir (Akçay ve ark., 2011). Şekil 3.6’da dağıtık bellekli paralel sistem mimarisi görülmektedir.



Şekil 3.6. Dağıtık bellekli paralel sistem mimarisi.(Akçay ve ark., 2011)

Dağıtık bellekli sistemlerde ortak bellekli sistemlerden farklı olarak komutlar arası paralellik değil de komut grupları arası paralellik mevcuttur. Kullanıcı isterse komutlarla paralel ortama müdahale edebilmektedir. Yüksek performans elde etmek isteyen araştırma kurumları, üniversiteler ve değişik ölçekteki kurumlar ekonomik olduğundan dolayı dağıtık bellekli paralel sistemleri tercih etmektedir. Bu sistemlerin temeli eski bilgisayarlarda (tek işlemcili bilgisayarlarda) kullanıldığından, önceden oluşturulmuş dağıtık mimarilerde hala yaygın olarak kullanılmaktadır (Çelik ve Özmen, 2009).

Dağıtık sistemlerin en büyük avantajı, ağa bağlı ve zaten kullanılmakta olan bilgisayarlardan ihtiyaç duyulan kadarını yazılım yoluyla yapılandırıp tek bir hızlı bilgisayara (süper bilgisayara) dönüştürebilmesidir. Ayrıca, sisteme ilave edilen bilgisayarların, mimarileri ya da işletim sistemleri farklı olabilmektedir. Dağıtık sisteme dâhil edilen makinenin iş yükü dağılımı doğru bir şekilde yapıldığı takdirde kullanıcısının farkına varmadan bir makine dağıtık sistem üyesi olabilir ve üzerinde iş koşturulabilmektedir. Bu tür sistemler ekonomik çözümler sunduğundan eğitim kurumlarında tercih edilmektedir. Dağıtık sistemlerden istenen performans elde etmek için sistem performansı gözlemleme monitörleri (XPVM, XMPI, vs.) tarafından izlenerek değerlendirilmesi yapılmalıdır (Çelik ve Özmen, 2009).

Dağıtık sistemlerde icra süresi (işlem süresi) ilk işlemin icraya başlamasından son işlemin icrayı sona erdirmesine kadar geçen süredir. Paralel bir uygulamanın iletişimde harcadığı sürenin, hesaplamada (işlemcilerde) harcanan süreye oranı ne kadar küçük olursa, paralellik o oranda başarılı olmuş olur. Bu oran büyüdükçe verim düşer, performans tek işlemcili sistemden bile daha kötü olabilir. Amaç, hesaplama süresi fazla uygulamalar gerçekleyebilmek olmalıdır (Çelik ve Özmen, 2009).

3.2.3.Paralel programlama modelleri

Bir paralel programlama modeli, paralel algoritmaları açıklayan bir yazılım teknolojileri kümesidir. Bu model, uygulamalar, diller, derleyiciler, kütüphaneler, iletişim sistemleri ve paralel giriş/çıkış alanlarını kapsar. Programcılar, kendileri ve uygulamaları için uygun bir model veya karma bir model seçip, uygulamalarını geliştirirler.

Paralel modeller çok farklı şekillerde uyarlanırlar: klasik sıralı dillerden çağrılan kütüphaneler şeklinde, dil uzantıları şeklinde ya da tamamen yeni işleme modelleriyle. Bu modeller kabaca ikiye ayrılırlar: paylaşımlı hafıza sistemleri ve dağıtık hafıza sistemleri. Günümüzde bu iki sistem arasındaki çizgi oldukça bulanıklaşmıştır.

Kullanılan paralel programlama modelleri şöyledir: Ateji PX, POSIX Threads, OpenMP, OpenHMPP, OpenACC, PVM, MPI, UPC, TBB, Boost Thread, Global Arrays, Charm++, Cilk/Cilk Plus, Coarray Fortran, OpenCL, CUDA, Dryad, C++ AMP, PLINQ, TPL, HPF, SHMEM, Occam, Linda.

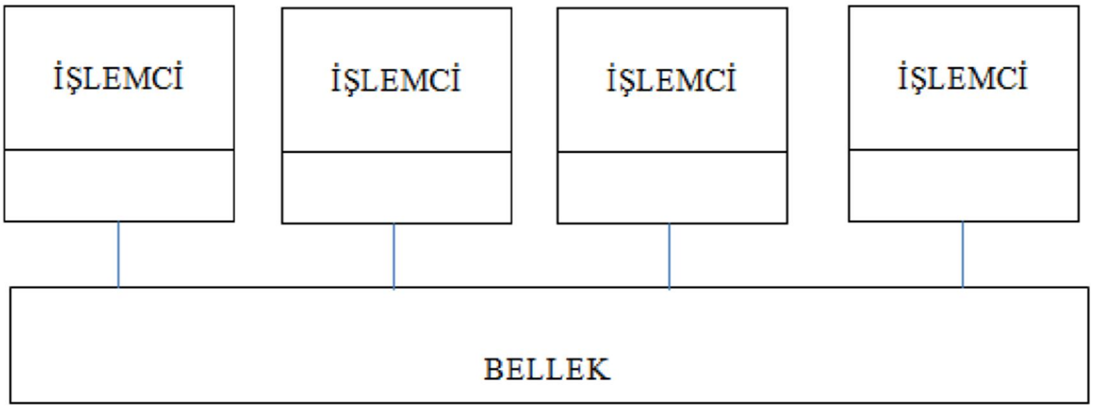
Çalışmamızda kullandığımız CUDA detaylı olarak anlatılacaktır.

3.2.4.Paralel bilgisayarların bellek mimarileri

Paralel bilgisayar yapıları; paylaşımlı bellek, dağıtık bellek ve dağıtık paylaşımlı bellek olmak üzere 3 gruba ayrılır. Problemin büyüklüğü ve hâlihazırdaki donanıma göre uygun mimariler tercih edilmektedir(Güneş, 2011).

3.2.4.1.Paylaşımlı bellek

Bu mimariye sahip bilgisayarlarda tüm işlemciler ortak kullanılan bellek alanına global adres uzayı üzerinden erişim sağlarlar. Tüm işlemciler birbirinden bağımsız olarak işlem yapabilirler ancak aynı bellek alanını paylaşırlar. Hafıza üzerindeki bir adres bölgesinde yapılan değişiklikler diğer işlemcilerin yürüttüğü işlemleri de etkilemektedir. Ortak bir bellek alanı kullanıldığı için işlemciler arası haberleşme hızlıdır. Bu modelde aktif olan bir süreç içerisindeki iş parçacıklarının sayısı artırılarak iş yükü dağıtılır. Böylece aynı adres uzayında işlem yapan iş parçacığı sayısı ile orantılı olarak programın hızı artacaktır. Paylaşımlı bellek mimarisi Şekil 3.7'de görselleştirilmiştir. Bu mimariye sahip bilgisayarlar UMA ve NUMA olmak üzere iki gruba ayrılmaktadır(Güneş 2011).



Şekil 3.7. Paylaşımlı bellekli mimari

UMA(Tek düze bellek erişimi)

Bu tür makinelerde yer alan işlemciler eşit miktarda bellek bölgelerini kullanırlar ve belleğe erişim süreleri eşittir. Genelde bu makineler CC-UMA(Uyumlu Ön Bellek Tek Düze Erişim-Cache Coherent Uniform Memory Access) olarak adlandırılırlar. Uyumlu ön bellek yapısı sayesinde, bir işlemci hafızanın belirli bir bölgesindeki veri üzerinde değişiklik yaptığında, diğer işlemcilerde bu değişiklikten haberdar edilmektedir. Bu tür makinelere örnek olarak Simetrik Çoklu İşlemcilere(SMP) sahip olan bilgisayarlar verilebilir(Güneş, 2011).

NUMA(Tek düze olmayan bellek erişimi)

İki ya da daha fazla SMP makinenin fiziksel olarak birbirlerine bağlanmasıyla oluşur. Bir SMP makine direkt olarak bir başka SMP makinenin hafızasına erişim sağlayabilir. Tüm işlemcilerin belleğe erişim süreleri ve bellek kullanım miktarları farklıdır (Güneş, 2011)

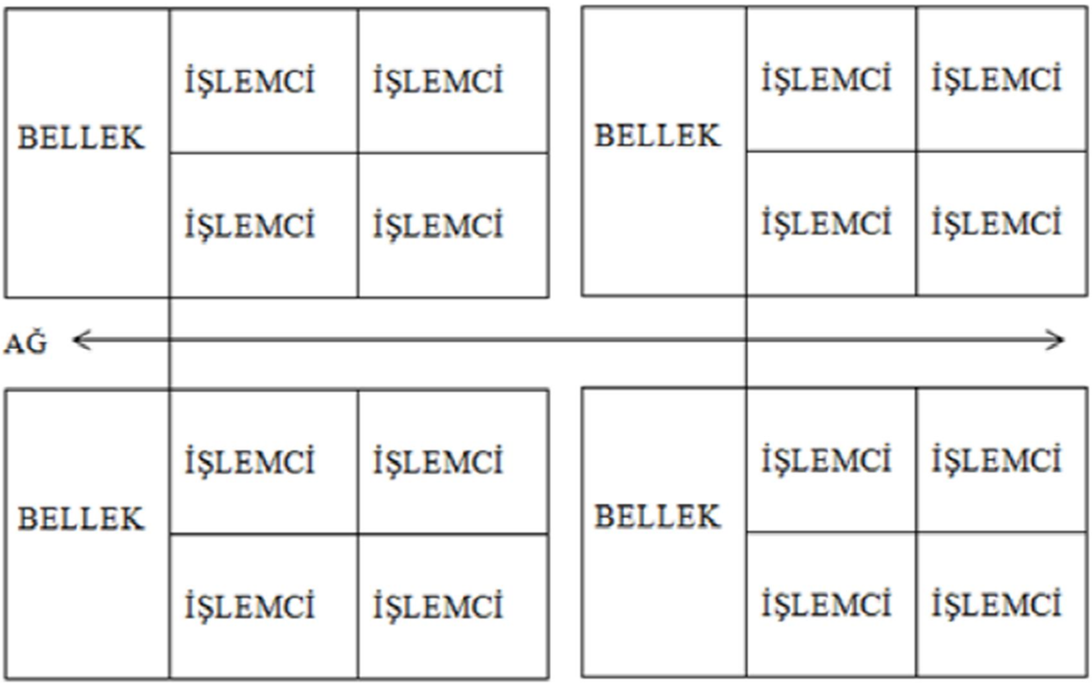
3.2.4.2.Dağıtık bellek

Bu modelde bilgisayarlar bir ağ hattı üzerinden mesajlaşarak haberleşirler. Her işlemci yer aldığı makinenin yerel belleğini kullanır. İşlemciler farklı bellek birimlerini kullandıkları için bellek adres uzayları tamamen farklıdır. İşlemciler birbirlerinden bağımsız olarak çalışırlar ve bir işlemcinin kendi hafıza bölgesinde yaptığı değişiklikler diğer işlemcilere yansımaz. İşlemciler veri ve iş paylaşımı için belirli kurallar çerçevesinde ağ hattı üzerinden mesajlaşırlar. Bu haberleşmeler aslında süreçler arasında gerçekleştirilir. Süreçler sistem kaynaklarını kullanabildikleri için, bir makinede aktif olan bir süreç, ihtiyaç dâhilinde diğer bir makinedeki sürece ağ hattı üzerinden mesaj göndererek iş yükünü dağıtabilir. Makineler arası iletişimin detayları programcı tarafından belirlenir. Bu sistemde çoklu süreç yaklaşımı söz konusudur. Dağıtık bellekli mimari Şekil 3.8'de görselleştirilmiştir.

Şekil 3.8. Dağıtık bellekli mimari

3.2.4.3.Dağıtık paylaşımlı bellek

Bu model, genelde paylaşımlı bellekli mimariye sahip olan SMP makinelerin bir ağ hattı üzerinden birleştirilmesiyle oluşturulur. Genelde çok büyük ölçekteki problemlerin çözümünde kullanılmaktadır. Makineler kendi içerisinde paylaşımlı bellekli mimaride olduğu gibi yerel bellekleri kullanırlar ve bir başka SMP makineyle iş yükünü paylaşmak için dağıtık bellekte olduğu gibi ağ hattı üzerinden mesajlaşırlar. Yani sistem kendi içerisinde çoklu iş parçacıklarını aynı anda işlerken, ağ ortamında da süreçler arası haberleşmeler söz konusudur. Şekil 3.9'da bu durum görselleştirilmiştir.



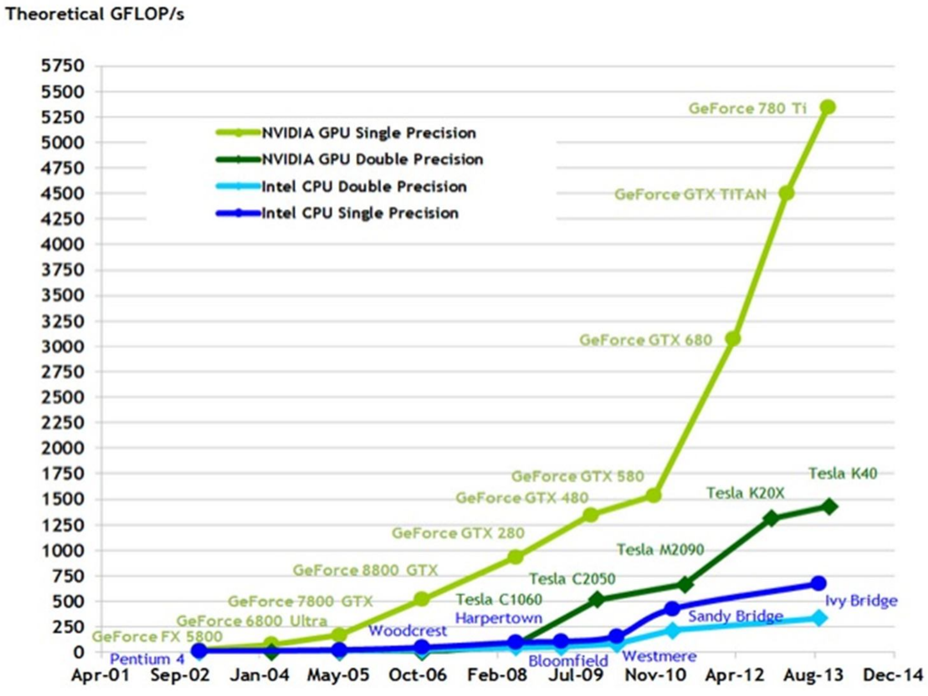
Şekil 3.9. Dağıtık paylaşımlı bellek

3.3.CUDA

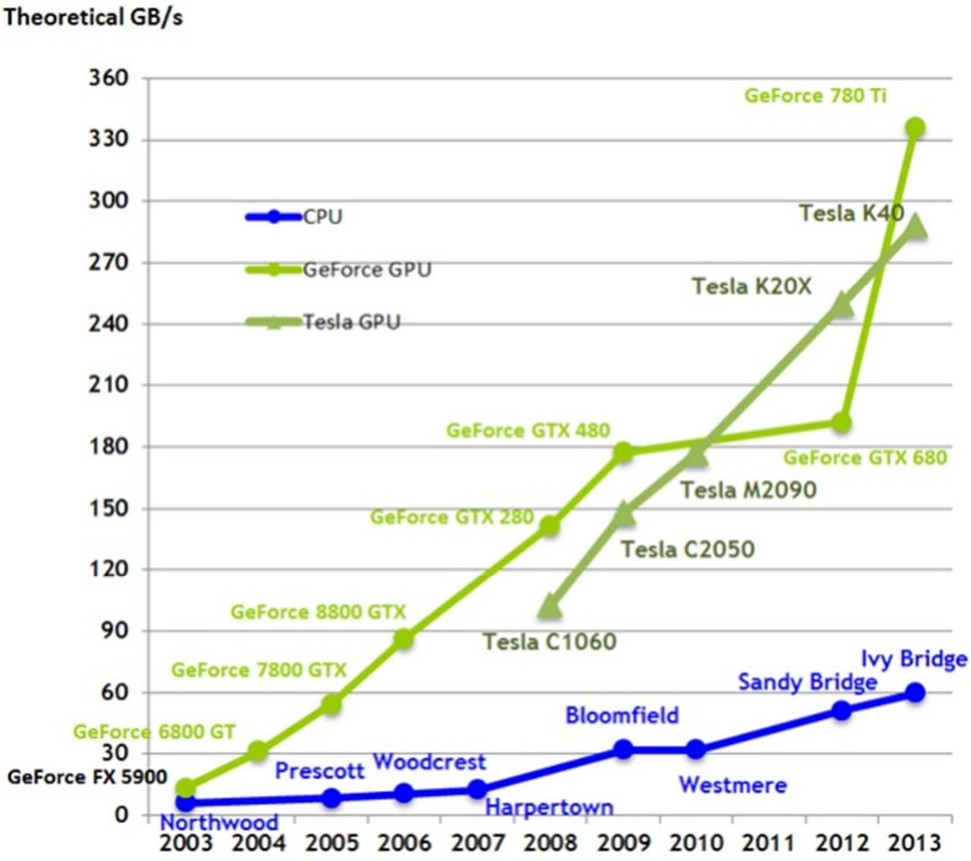
3.3.1.CUDA’ya genel bakış

1980’li yıllarda IBM’in ve Intel’in hâkimiyetindeki GPU geliştirme çalışmaları, 1990’lı yılların başından itibaren sadece grafik kartı geliştirmeye yönelmiş olan S3 Graphics, NVIDIA ve ATI gibi firmaların kontrolünde devam etti. Bu dönemde 2B donanımsal hızlandırma yaygınlaştı. Yine aynı dönemde OpenGL grafik programlama kütüphanesinin kullanıma sunulmasıyla GPU dünyasında yeni bir dönem başladı.

2000’lı yıllarda ise GPU geliştirme çalışmaları, GPU hesaplama gücünün kat ve kat artmasıyla sonuçlanmıştır. Yıllara göre GFLOPS (Giga Floating-Point Operations Per Second) ve yıllara göre bellek bant genişliği bazında CPU-GPU kıyaslaması aşağıda yer almaktadır. Single Precision tek hassasiyetli anlamına gelmekte olup 4 byte yer kaplayan sayı formatıdır. Double Precision çift hassasiyetli anlamına gelmekte olup 8 byte yer kaplayan sayı formatıdır. Yapılacak işe göre donanım seçimi için bu özelliklerin bilinmesi önemlidir. Şekil 3.10’da CPU ve GPU’ların yıllara göre teorik GFLOPS değişimi, Şekil 3.11’de ise CPU ve GPU’ların yıllara göre bellek bant genişliği görülmektedir.



Şekil 3.10. Yıllara göre GFLOPS (Giga Floating-Point Operations Per Second)(Nvidia, 2015)



Şekil 3.11. Yıllara göre bellek bant genişliği (Nvidia, 2015)

Dağıtık sistemlerin kurulması ve PVM, MPI gibi paralel programlama yazılımlarının kullanımı oldukça zordur. Ayrıca mevcut ortak bellekli paralel yazılım programları olan OpenMP, POSIX Threads ve Win 32/64 Threads ile tek bir programın birden çok iş parçacığına dağıtılarak çalıştırılmak üzere hazırlanması da oldukça zordur.

Bahsedilen ortak bellekli mimarilerde sadece CPU üzerinde programlama yapabilmektedir. Bu durumda bir ortak bellekli paralel yazılım olan CUDA, hem GPU üzerinde kullanılabilen ortak bellekli bir yazılım olarak hem de kolay iş parçacığı yönetimi yapısıyla oldukça kullanışlı ve yararlı olan bir yazılımdır.

CUDA grafik işlemcilerde çalışmak üzere oluşturulmuş bir mimari, bir geliştirme ortamıdır. CUDA kolay ve güçlü bir yazılımdır. CUDA, C temelli bir paralel programlama dili olduğundan, CPU için C programlama dili ile yazılmış olan uygulamaları, grafik işlemci üzerinde çoklu iş parçacığı (Multi-thread) kullanılarak koşturulmasına olanak sağlar.

CUDA teknolojisine donanımsal olarak bakıldığında iş parçacıkları SP (streaming processor) üzerinde çalışır. X adet SP işlemcisinin oluşturduğu yapıya ise

SM (Streaming Multiprocessor) olarak adlandırılır. SM'ler ise ekran kartı donanımını oluşturmaktadır.

CUDA manuel olarak paralellik oluşturduğu için bütün zorlukları ortadan kaldırır. CUDA'da yazılmış bir program, aslında "kernel" (Çekirdek) adı verilen seri bir programdır. GPU bu kernelin istenildiği kadar kopyasını çalıştırarak onu paralel hale getirir. CUDA, C dilinin bir uzantısı olduğu için genellikle programları CUDA'ya yönlendirmek veya onları multi-thread hale getirmek için mimarilerini değiştirmeye gerek yoktur.

Program çalıştırıldığında CPU kodun kendisine ait olan seri kısmını, GPU ise ağır hesaplamalar gerektiren paralel CUDA kodunu çalıştırır. Kodun GPU kısmına "kernel" adı verilir. Kernel, belirli bir veri kümesine uygulanacak olan işlemleri tanımlar. GPU, veri kümesinin her unsuru için ayrı bir kernel kopyası oluşturur. Bu kernel kopyalarına iş parçacığı(thread) adı verilir. 512 veya 1024 adet(cihazın özelliğine bağlı olarak) iş parçacığının birleşimiyle blok(block), 65536 adet bloğun birleşmesiyle oluşan gruplara ise grid adı verilir.

Her bir iş parçacığı kendine ait program sayacı(program counter), kaydedici (register) ve durum(state) bilgilerini barındırır. Görüntü veya veri işleme gibi geniş veri kümelerinde bir seferde milyonlarca iş parçacığı oluşturulur ve paralel şekilde çalıştırılır. CUDA teknolojisinin temeli birçok iş parçacığı dizisi tarafından GPU'nun çalıştırılması temeline dayanır.

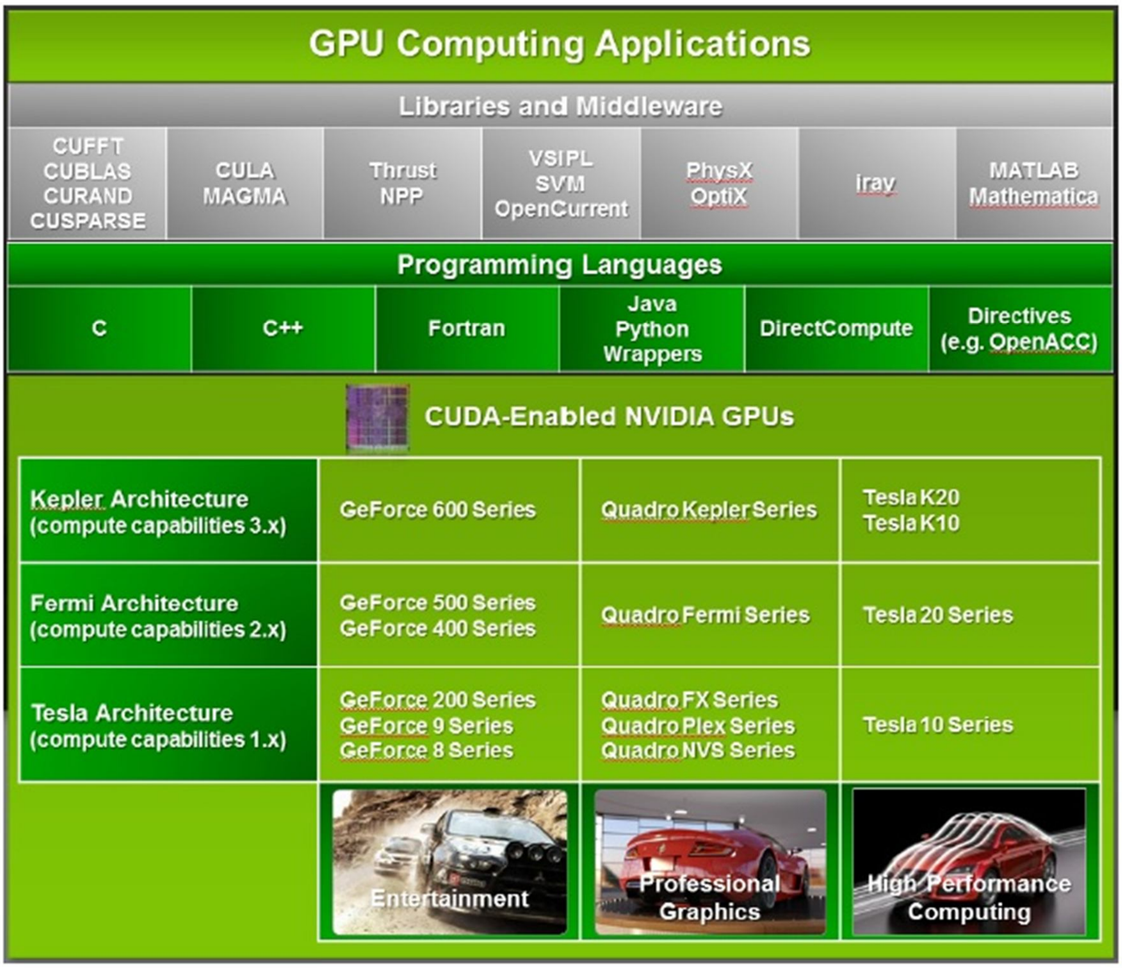
Bütün iş parçacıkları aynı kodu çalıştırabilir ve her bir iş parçacığının bir benzersiz numarası vardır. Aynı zamanda her bir iş parçacığı paylaşımlı(shared) bellek alanı kullanabilir.

CUDA mimarisi üzerinde yapılan örnek çalışmalarla performansın CPU üzerindeki uygulamalara göre daha yüksek olduğu netleşmiştir. Aslında bu fark temel olarak GPU mimarisinin, grafik işleme gibi işlem yoğunluğu olan ve yüksek derecelerde paralellik gerektiren işlemler için geliştirilmiş olmasından kaynaklanmaktadır.

CPU'daki gibi akış kontrolü gerektiren işlemler yerine, aritmetiğin yoğun olduğu ve her bir piksel için tekrar eden işlemlerden oluşan resim/görüntü işleme, 3 boyutlu görselleştirme(rendering) ve sinyal işleme türündeki uygulamaların GPU'nun hedef uygulamaları olması, önbellek ve akış kontrol mekanizmalarının yerine veri işlemeye yönelik transistörlerin ağırlıkta olduğu bir mimarinin ortaya çıkmasını sağlamıştır (Nvidia, 2015). Şekil 3.12'de bu durum görselleştirilmiştir.

Şekil 3.12. GPU veri işleme için daha fazla transistor tahsis eder. (Nvidia, 2015)

CUDA, yazılım geliştiricilerin yüksek seviyeli bir dil olan C ile yazılım geliştirmelerine izin veren bir ortamla birlikte gelir. Şekil 3.13'te CUDA platformu tarafından desteklenen diller ve program uygulama ara yüzleri görülmektedir.



Şekil 3.13. CUDA çeşitli dilleri ve uygulama ara yüzlerini desteklemektedir.(Nvidia, 2015)

CUDA mimarisi yazılımcıyı GPU çekirdekleri ile uğraşmak zorunda bırakmaz.

3.3.2.CUDA programlama modeli

GPU mimarisi donanıma göre farklı özellikler göstermesine ve CUDA ile geliştirilen uygulamaları etkilemesine rağmen, programlama modeli açısından temel çalışma prensipleri aynıdır.

CUDA; Fortran, OpenCL ve DirectCompute gibi farklı dil ve programlama arabirimlerini belirli ek komut ve kısıtlamalarla desteklemektedir. CUDA mimarisini programlamada kullanılabilecek dillerden biri de Cuda C'dir. CUDA C ile programlamayı kolaylaştırmak için kullanılabilecek hazır kütüphaneler geliştirilmiştir. Ayrıca Nawata ve Suda, C kodunu hiçbir direktif olmadan Cuda C koduna çeviren, Auto Parallelizing Translator from C to CUDA(APTCC) çalışması ile karmaşık GPU mimarisini gözardı ederek algoritmaya odaklanmıştır.

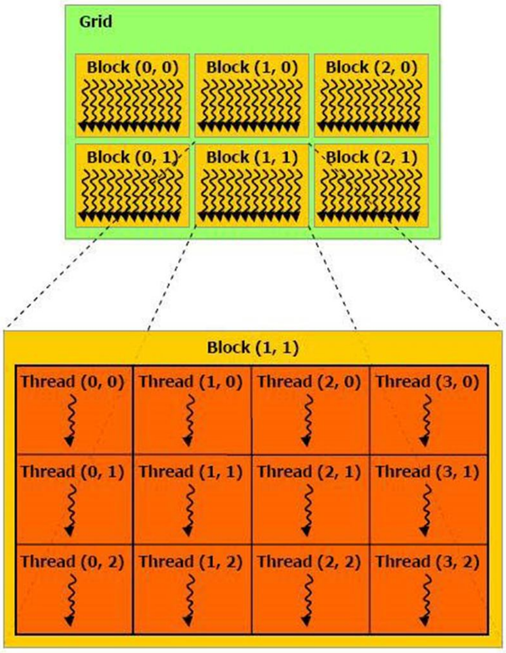
CUDA C, C programlama dilinden türetilmiştir. Programcının C fonksiyonlarına benzer biçimde kodlamasına olanak sağlanmaktadır. Paralel şekilde çalıştırılan Cuda iş parçacıkları oluşturularak uygulamada gerekli yerlerde çağrılmaktadır.

Uygulama, CPU'nun kontrolünde çalıştırılır. CUDA paralel programlama modelinde, ana işlemci (CPU) host olarak isimlendirilirken, ana işlemciye yardımcı olan GPU device olarak adlandırılır. İşlenecek verinin host-device, device-host aktarımı ve GPU'ların yönetimi CPU tarafından gerçekleştirilir. CUDA uygulamaların CPU ve GPU arasında karışık olarak çalıştırılmasına imkân sağlamaktadır.

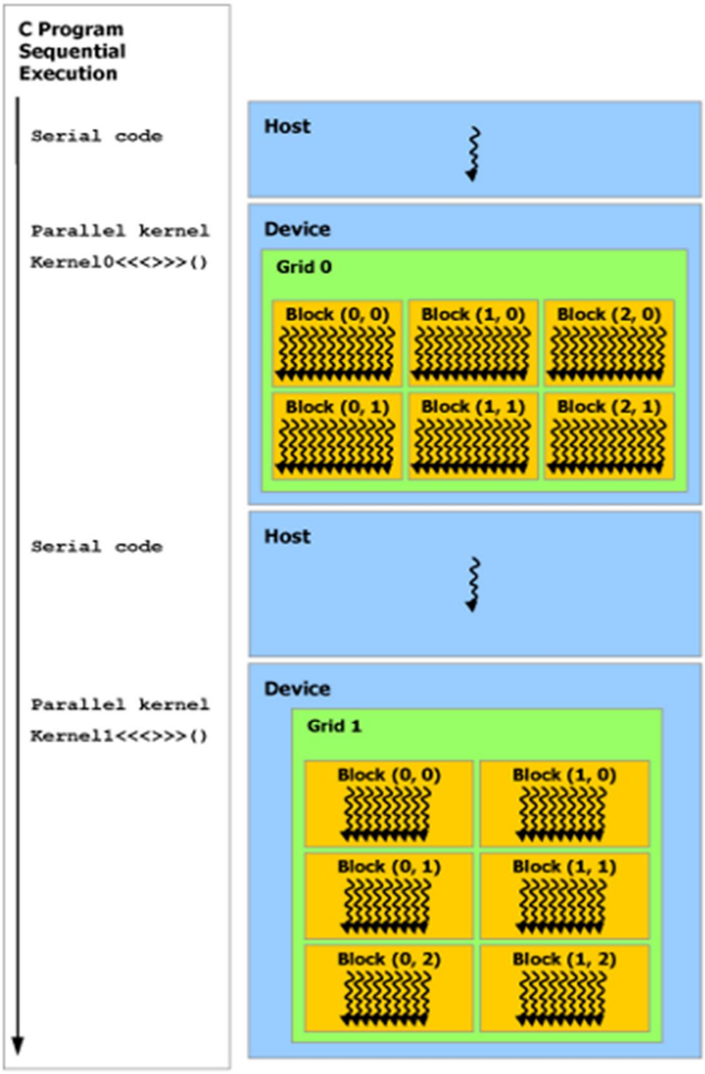
Koşut işlem yapılması istendiğinde, kernel olarak isimlendirilen, özel tanımlanmış bir C işlevi ile aygıta bildirilmektedir.

KernelName <<<gridsize,blocksize>>>(…) şeklinde koşut işlev çağrılarak, çalıştırılacak topoloji belirtilmektedir.

Paralel programlama için işlevler, ızgara (grid), blok (block) ve iş parçacığı (thread) olarak isimlendirilmiş üç yapılı bir sistem ile gerçekleştirilmektedir. Şekil 3.14'de CUDA iş parçacıklarına ait diyagram gösterilmektedir. İş parçacıkları 1, 2 veya 3 boyutlu diziler şeklinde blokları oluşturur. Bir veya iki boyutlu blok dizileri ise ızgarayı meydana getirmektedir. Şekil 3.15'de CPU ve GPU üzerinde karışık programlanmış bir uygulamanın çalışma şeması gösterilmektedir.



Şekil 3.14. CUDA Grid-Block-Thread yapısı (Nvidia, 2015)



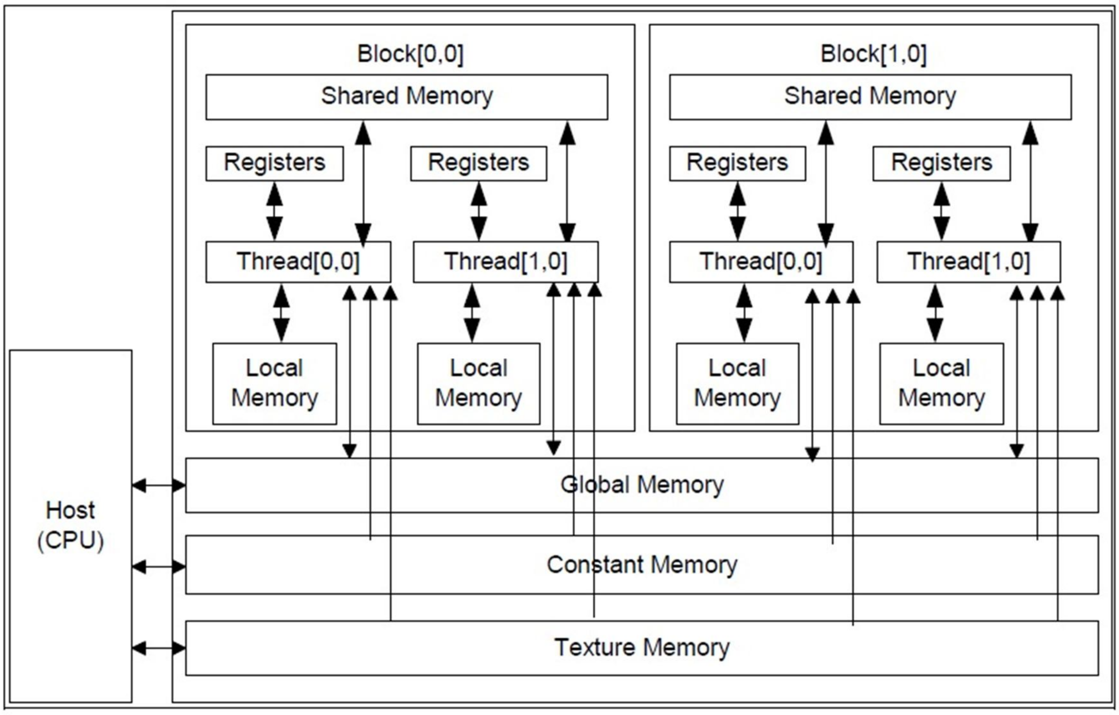
Şekil 3.15. CPU ve GPU üzerinde karışık programlanmış bir uygulamanın çalışma şeması (Nvidia, 2015)

3.3.3.CUDA bellek yapısı

NVIDIA ekran kartlarında genel amaçlı programlama için global, paylaşımlı(shared) ve yerel(local) bellek olmak üzere üç farklı bellek türü tanımlanmıştır. Şekil 3.16'da host, device, blok ve iş parçacıklarının belleklere erişim modeli görülmektedir.

Global bellek, ekran kartının RAM'idir. GPU'daki tüm iş parçacıkları tarafından erişilebilmektedir. Erişim hızı nispeten yavaştır. Her bloğun okuma ve yazma yapılabildiği bu bellek host ile device arasında veri aktarılmasında ve bloklar arası iletişimde kullanılmaktadır.

Paylaşımlı bellek, global bellekten hızlıdır fakat sadece aynı blok içindeki iş parçacıkları tarafından erişilebilmektedir. Gerektiğinde blok içinde iletişimde kullanılabilmektedir.



Şekil 3.16. CUDA aygıtının bellek yapısı (Nvidia, 2015)

CUDA programlama modelinde, her iş parçacığı için sadece kendisine ait bir yerel bellek vardır. Ana belleğe erişim hızı yavaş olduğu için ekran kartlarında hızlı erişim yapılabilen çok sayıda yazmaç (register) bulunmaktadır.

Ekran kartlarında genel amaçlı programlama dışında grafik uygulamalarında kullanılmak üzere sabit (constant) ve doku (texture) bellek olarak isimlendirilen iki çeşit bellek daha bulunmaktadır. CPU constant and texture belleğe yazabilir ve okuyabilirken GPU'daki iş parçacıklarının sadece okuma izni bulunmaktadır.

Değişmez bellek alanı mevcut NVIDIA ürünlerinde 64KB veya daha azdır. Bu nedenle etkin kullanımı zordur.

Doku bellek, GPU tarafından donanım olarak desteklenen ve global bellek üzerindeki işlemlere göre daha yüksek performans değerlerini destekleyen bir teknolojidir.

Uygulama ara yüzü için sadece okuma işlemlerine izin verir. Kullanımında ise temel olarak önce bir veri kaynağı ile bağlantı yapılır. Daha sonra CUDA tarafından sağlanan fonksiyonlar ile veri yönetilir.

3.3.4.CUDA C dili

Cuda C ile yazılan işlevler için işlevin host ya da device da çalışacağını belirtmek üzere işlev niteleyicileri kullanılmaktadır.

__device__ ,önüne getirildiği işlevin device da yani ekran kartı üzerinde çalışacağını ve sadece device da işlenen yordamlar tarafından çağırılabileceğini belirtmektedir.

__global__ , önüne getirildiği işlevin device üzerinde çalışacağını ve ancak host yani işlemcide çalışan yordamlar tarafından çağırılabileceğini belirtmektedir.

__host__, önüne getirildiği işlevin işlemci üzerinde çalışacağını ve işlemcide çalışan yordamlar tarafından çağırılabileceğini belirtmektedir.

Bir işlev için niteleyici tanımlanmadığında CUDA işlevin **__host__** niteleyicisine sahip olduğunu varsayar ve işlemci üzerinde çalıştırır.

ThreadID bir bloktaki thread'in indeksini verir. 1, 2 veya 3 boyutlu olabilir. Tek boyutlu bir blokta threadID dizinin indeksi iken, iki boyutlu bir blok (Dx, Dy) için, thread indexi (x + y Dx) şeklinde; üç boyutlu blok(Dx, Dy, Dz) içinse, thread indeksi (x + y Dx + z Dx Dy) şeklinde hesaplanabilir.

Bir grid'deki blok indeksi ise blockID ile ifade edilir. 1 veya 2 boyutlu olabilir. blockIdx.x ve blockIdx.y şeklinde ilgili değerlere ulaşılabilir. Bir bloktaki thread sayısı blockDim ile 1, 2 veya 3 boyutlu ifade edilebilir. Compute Capability 1.x olan GPU'lar için bu değer maksimum 512 iken Compute Capability 2.x olan GPU'lar için maksimum 1024’tür. Bir grid’deki blok sayısı ise gridDim terimiyle 1 veya 2 boyutlu olarak ifade edilebilir. Grid’de maksimum 65535 blok bulunabilir.

Host aynı zamanda GPU belleğini de yönetir. Device tarafında kullanılacak veriler için host, önce alan ayırarak verileri device belleğine kopyalar.

Host, grid ve blok boyut bilgileri belirlenerek kernel’in kaç kez çalışacağı ayarlanmalıdır. Kernel çalışmayı bitirdiğinde host, device belleğindeki veriyi host belleğine aktararak, device üzerinde ayrılan alanı serbest bırakmalıdır.

Host tarafından bir kez KernelName <<<gridsize,blocksize>>>(…) ifadesi şeklinde çağrılan kernel, device üzerinde paralel işlenen bir dizi thread tarafından çalıştırılır. Bütün threadler aynı kodu çalıştırmasına rağmen kendilerine ait benzersiz iş parçacığı belirleyicisi sayesinde alınacak kararlar ve yapılacak hesaplamalarda erişilecek bellek adresi kontrolü sağlanmış olur.

3.3.4.1.Kerneller

CUDA C, standart C dilini genişletir ve kullanıcılara kernel denilen C fonksiyonlarını tanımlama imkânı sağlar. Kernel fonksiyonları, normal C fonksiyonlarından farklı olarak N kere çağrıldıklarında, N tane ayrı kanalda paralel olarak çalışırlar.

Kernel fonksiyonları __global__ ifadesi ile tanımlanır. Kerneli çalıştıracak kanalların sayısı <<<...>>> ifadesi ile belirtilir. Kerneli çalıştıran her bir kanala özel bir anahtar değer (ID) verilir. Bu değere “threadIDx” değişkeni üzerinden ulaşılır.

Aşağıda bir kod parçacığının hem seri hem de paralel hali örnek olarak verilmiştir.

x ve y vektörleri için αx+y işleminin sıralı hali:

```
void saxpy(int n, float a, float *x, float *y)
{
        for (int i = 0; i < n; ++i)
            y[i] = a*x[i] + y[i];
}
saxpy(1<<20, 2.0, x, y);
```

x ve y vektörleri için αx+y işleminin CUDA destekli paralel hali:

```
__global__ void saxpy(int n, float a, float *x, float *y)
        {
```

```
      int i = blockIdx.x*blockDim.x + threadIdx.x;
      if (i < n) y[i] = a*x[i] + y[i];
     }
     ...
     int N = 1<<20;
     cudaMemcpy(d_x, x, N, cudaMemcpyHostToDevice);
     cudaMemcpy(d_y, y, N, cudaMemcpyHostToDevice);

     // SAXPY işlemi 1 milyon eleman için yapılmaktadır
     saxpy<<<4096,256>>>(N, 2.0, x, y);

cudaMemcpy(y, d_y, N, cudaMemcpyDeviceToHost);
```

Gösterimde saxpy kerneli <<<B,T>>> türünde bir konfigürasyon ile <<<4096,256>>> şeklinde çağırılmıştır. Burada B blok sayısını ve T de her bir blok içindeki kanal sayısını ifade eder.

3.3.4.2.Kanal hiyerarşisi

Kanalların ID değerlerine ulaşmamızı sağlayan threadIDx değişkeni 3 elementli bir elemandır. Bu yapısı ile vektör, matris veya 3 boyutlu veri kümeleri üzerinde işlemlerin kolayca yapılmasını sağlar.

Kanal indeks değeri ve kanal ID değerleri arasındaki ilişki, data bloğunun boyutlarına göre değişir. Tek boyutlu bir data bloğu için kanal indeks ve ID değerleri bir birine eşittir. İki boyutlu, boyutları Dx ve Dy olan bir data bloğu için, x ve y koordinatlarındaki kanal için indeks (Kin) değeri

“Kin=x+y Dy”
x: Kanalın, kanallardan oluşan matris içerisindeki sütun numarası
y: Kanalın, kanallardan oluşan matris içerisindeki satır numarası
Dx: Kanal matrisinin sütun sayısı
Dy: Kanal matrisinin satır sayısı
Olarak hesaplanır. Üç boyutlu veri kümelerinde ise
Kin=x+y Dx+z Dx Dy
x: Kanalın, kanal kümesindeki koordinatının x değeri
y: Kanalın, kanal kümesindeki koordinatının y değeri

z: Kanalın, kanal kümesindeki koordinatının z değeri

Dx: Kanal kümesinin x ekseni değeri

Dy: Kanal kümesinin y ekseni değeri

Dz: Kanal kümesinin z ekseni değeri

formülü geçerlidir.

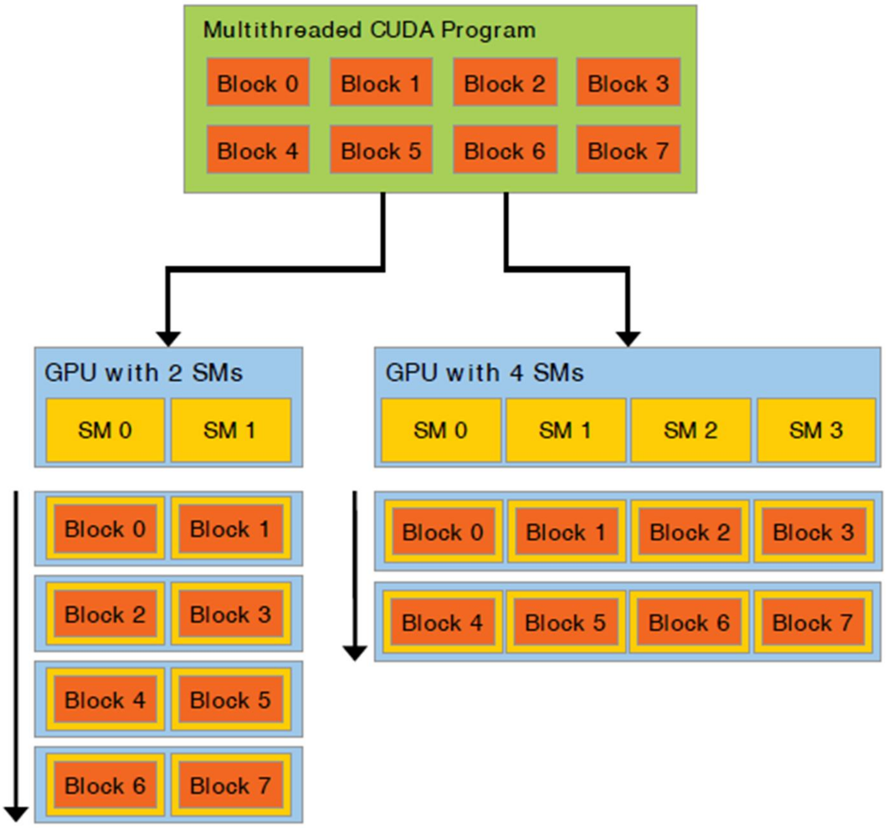
Bir blokta açılabilecek kanal sayısı donanıma göre değişiklik göstermektedir. Mevcut ürünlerde bu sayı 1024'e kadar çıkabilir. İşlenecek verinin tam kapsanması için eşit şekillendirilmiş bloklardan yararlanılabilir. Böylece toplam kanal sayısı, bir bloktaki kanal sayısının toplam blok sayısı ile çarpılmasıyla elde edilir.

Bloklar bir boyutlu veya iki boyutlu kümeler halinde olabilir. Bu yapılara grid denir.

Şekil 3.15'te 6 bloktan oluşan bir kanal yapısı görülmektedir. Grid içerisindeki herhangi bir blok, bir boyutlu veya iki boyutlu bir indeks ile ifade edilebilir. Bu indeks değeri blockIdx olarak adlandırılır. "blockDim" elamanı üzerinden söz konusu bloğun boyutları elde edilebilir.

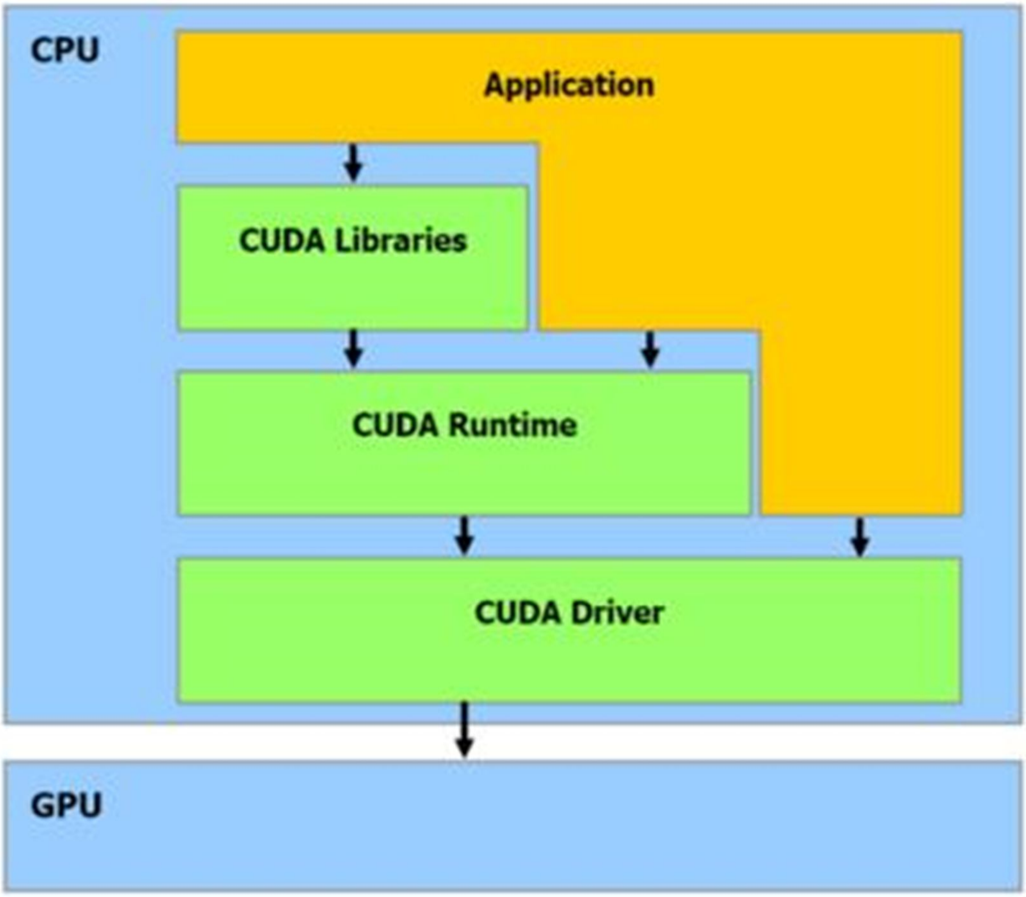
Blok içerisindeki kanallar etkileşimli olarak çalışabilirler. Ayrıca erişim hızının çok yüksek olduğu ortak paylaşımlı bellek yapısı mevcuttur. Bu yapıya erişimleri düzenleyen ve blok içerisindeki kanalların senkronizasyonunu sağlayan mekanizmalar da mevcuttur.

CUDA C ile yazılmış bir uygulama, bir kerneli çağırdıktan sonra bu uygulamanın çekirdekler arasında nasıl yönetileceği konusunda bir işlem yapmak zorunda değildir. CUDA mimarisi bunu kullanıcıdan soyutlar ve yönetimini gerçekleştirir. 8 bloktan oluşan bir kernelin farklı sayıda çekirdekleri olan iki mimarideki paralel çalışma modeli Şekil 3.17'de görülmektedir.



Sekil 3.17. Değişen çekirdek sayısına göre blokların paylaşımı (Nvidia, 2015)

Hesaplama kapasitesi büyük versiyon ve küçük versiyon numaralarıyla ifade edilir. Büyük versiyon numaraları aynı olan ürünler aynı çekirdek mimarisine sahiptirler. Küçük versiyon numarası ise mimari üzerinde yapılan güncellemelere göre değişir. Örneğin Fermi mimarisine sahip ürünlerin hesaplama kapasitesi 2.x ile ifade edilir. Fermi mimarisinden önceki CUDA mimarili ürünlerin hesaplama kapasiteleri ise 1.x ile ifade edilir. 3.x versiyonunu Kepler, 5.x versiyonu ise Maxwell olarak adlandırılmaktadır. Günümüzden 5.3 versiyonlu Tegra X1 son üretilen CUDA destekli ürün olarak bilinmektedir. Hesaplama kapasitesine göre çoklu-işlemcilerin yetenek ve özellikleri farklıdır.

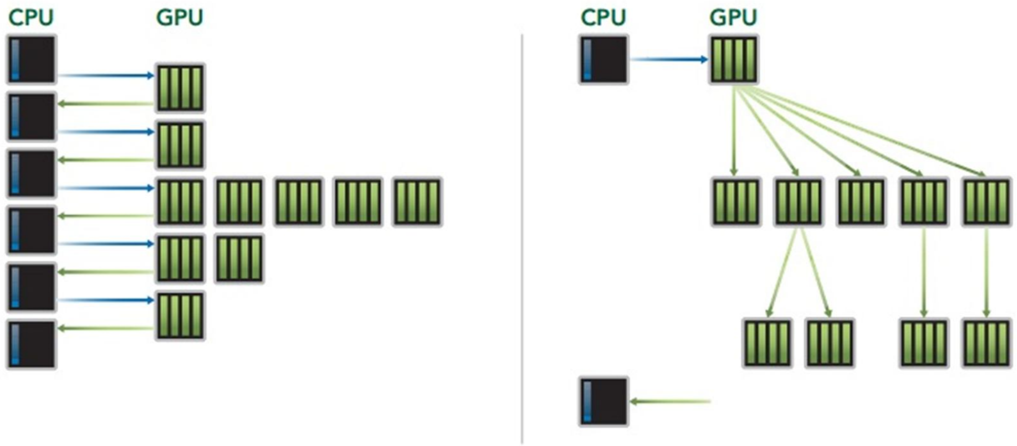


Sekil 3.18. CUDA yazılım yığını ve elemanları (Nvidia, 2015)

CUDA yazılım yığını bir aygıt sürücüsü, bir uygulama programlama ara yüzü ve yürütüm yazılımı, iki adet genel kullanım için yüksek seviye matematik kütüphanesinden oluşmaktadır. Şekil 3.18'de CUDA yazılım yığını ve elemanları görülmektedir.

3.3.4.3.Dinamik paralellik(Dynamic parallelism) nedir?

Kepler mimarisiyle gelen en önemli özelliklerden birisidir. GPU içerisinde kernellerin birbirleriyle iletişim kurmasını, işlem yapmasını sağlar. Böylece her adımda CPU ile iletişim kurmaya gerek kalmaz. Özyineli ve veri bağımlı yürütme desenleri (recursive and data-dependent execution patterns) kodlarken çok faydalı olmaktadır. Şekil 3.19'da dinamik paralellik modeli görülmektedir.



Şekil 3.19. CUDA'da dinamik paralellik (Dynamic parallelism) (Nvidia, 2015)

3.3.5.Cuda'nın artıları ve eksileri

3.3.5.1.CUDA'nın artıları

- Farklı işletim sistemlerinde çalışabilmektedir. (Linux, Windows)
- Birbirinden bağımsız büyük verileri paralel olarak hızlı bir şekilde işlemeye olanak sağlamaktadır. (GTX580 modeli için teorik maksimum işlem performansı 1581.1 GFLOPS olarak belirtilmektedir.)
- CPU işlemci hızı ile sınırlı kalırken GPU çekirdeklerde paralel işleme yeteneği ile performansı arttırmaktadır.
- GPU üzerinde C/C++ programlama dilleri başta olmak üzere her geçen gün artan programlama dilleri ve bazı ufak CUDA eklentileri ile programlama yapabilme olanağı sağlamaktadır.
- Heterojen seri ve paralel programlama modeline sahiptir. (Algoritmalar için paralel CUDA kodu ile seri C kodu heterojen bir şekilde gerçeklenebilmektedir.)
- Farklı serbest ve optimize edilmiş yazılım kütüphaneleri bulunmaktadır. CUFFT ve CUBLAS kütüphaneleri FFT ve temel lineer cebir fonksiyonlarını içermektedir.

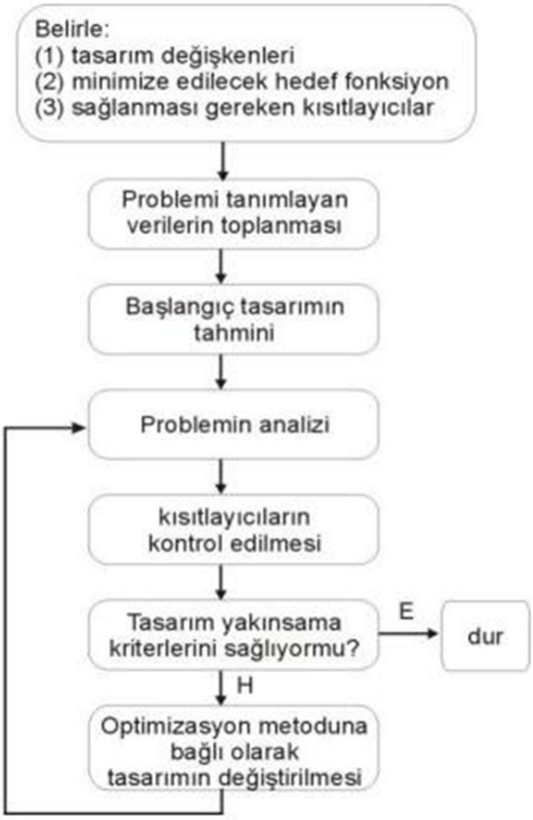
3.3.5.2.CUDA'nın eksileri

- Veriyolu (bus) bant genişliği ve bekleme süresi (latency) CPU ve GPU arasında darboğaza sebep olabilmektedir.
- Sadece NVIDIA firmasının ürettiği donanımlarla çalışabilmektedir.

3.4.Optimizasyon

3.4.1.Klasik optimizasyon

Kelime anlamı olarak "mümkün olan en iyiyi bulma" ve "daha iyiyi yapma" anlamına gelen optimizasyon, genel anlamda eldeki kısıtlı imkanları en optimum şekilde kullanabilmek olarak tanımlanır. Ortakcı (2011) ise optimizasyonu, bir uygunluk (amaç, değerlendirme) fonksiyonuna göre belirli aralıktaki sayısal değerlerin en uygununu seçerek kompleks problemleri çözmek olarak tanımlamıştır. Matematikte ise optimizasyon, belirlenen şartlar altında fonksiyonların minimum ve maksimum değerlerinin tespiti ile ilgilenen bir disiplindir. Her hangi bir alana optimizasyon işlemi uygulanarak maksimum performans ve minimum maliyeti sağlayan çözüm elde edilmeye çalışılır. Örneğin; 100kg şeker pancarında maksimum şeker elde etmeye çalışma işlemi maksimizasyon, 100 kg şekeri minimum şeker pancarından elde etme işlemi ise minimizasyon olarak adlandırılabilir. İki işlem için de amaç maliyeti düşürmek ve verimi artırmaktır. Belirli kısıtları olan, bir problemin sonucunu etkileyen parametre değerlerinin bulunarak en kârlı sonucun minimum maliyetlerle belirlenmesini hedeflemek, problemin optimize edilmesi anlamını taşır. Her bir gerçek dünya probleminde gerekli çaba, sermaye, malzeme ve işçiliğin minimum seviyede belirlenmesi ve kazancın maksimum düzeyde olması en gerçekçi amaç olmuştur. Optimizasyon işleminde problemin çözümünü belirleyen karar değişkenlerinin belirlenmesi, sonrasında ise bu karar verici parametreler ışığında minimize edilecek maliyet fonksiyonu ya da maksimize edilecek kâr fonksiyonları tanımlanmalıdır (Amaç fonksiyonu). Bunların tanımlanmasında problemi sınırlayan, karar değişkenlerinin alabileceği değer ya da değer aralıklarını ifade eden sınırlamaların belirtilmesi gerekmektedir. Probleme göre bazı kısıtlamalar eşitsizlik, bazıları ise eşitlikler şeklinde olabilmektedir (kısıtlayıcılar). Şekil 3.20'de bir sistem tasarlanırken optimizasyona ait yukarıda açıklanan adımlar şematize edilmiştir.



Şekil 3.20. Bir tasarımın optimizasyonu(Eldem, 2014)

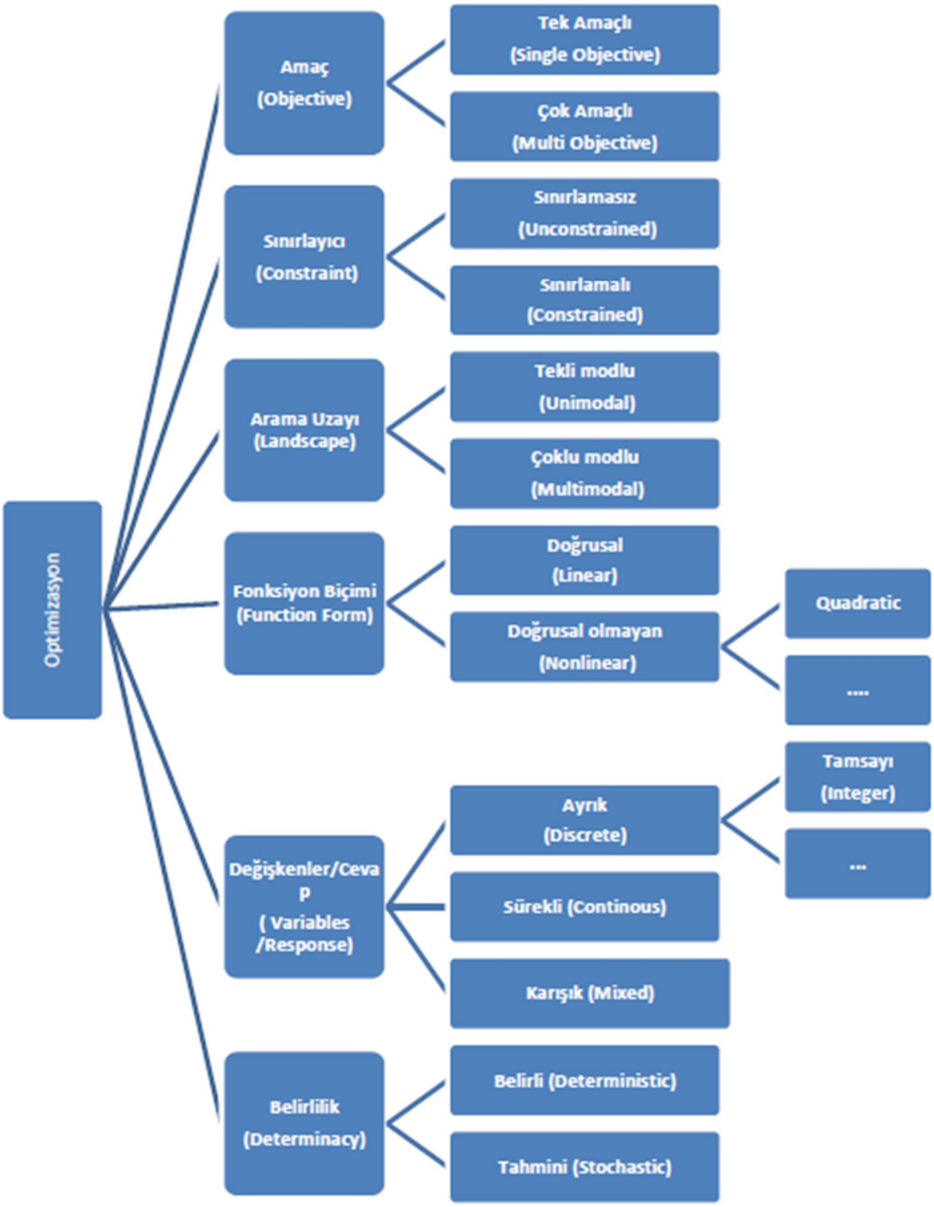
Optimizasyon, geniş bir yelpazede ele alınacak problemlerin kesin optimum sonuçlarının araştırılması amaçlarını içermektedir. Bu yüzden optimizasyon problemlerinin çok farklı isimlendirilmeleri ve sınıflandırılmaları mevcuttur. Genellikle optimizasyon teknikleri problemden probleme önemli ölçüde değişebilmektedir. Her bir optimizasyon probleminin çözümü için tek bir yaklaşım söz konusu olamamaktadır. Çünkü her bir problemin karmaşıklığı, önemli oranda kısıtlayıcıları ve amaç fonksiyonlarına bağlı olduğundan çok farklılık gösterebilmektedir.

Optimizasyon problemlerindeki sınıflandırma, hedef sayısı, kısıtlayıcıların sayısı, fonksiyonların yapısı, hedef fonksiyonunun yapısı, tasarım değişkenlerinin türü, değerlerdeki belirsizlik ve hesaplama zorluğuna göre olmaktadır (Eldem, 2014).

Gerçek dünya problemlerinin bir çoğu çok amaçlı optimizasyon problemleri olarak değerlendirilmektedir. Örneğin bir araba motoru tasarlanmasında yakıt verimliliğini maksimize etmek, karbon dioksit emisyonunu en aza indirmek ve gürültü seviyesini düşürmek gibi çok kıstas göz önüne alınmalıdır.

Fonksiyonların yapısı bakımından optimizasyon problemleri lineer ve lineer olmayan (nonlineer) şekilde ikiye ayrılmaktadır. Optimizasyon probleminin kısıtlayıcıları lineer ise problem lineer (doğrusal) kısıtlı olmaktadır. Amaç fonksiyonu ve kısıtlayıcıların hepsi doğrusal ise doğrusal optimizasyon, bunlardan herhangi birinin doğrusal olmaması durumunda ise problem doğrusal olmayan optimizasyon problemi (nonlinear optimization problem) olarak adlandırılmaktadır.

Tasarım değişkenlerinin değer türü de sınıflandırma için kullanılabilir. Tüm tasarım değişkenlerinin değerleri ayrık ise, optimizasyon problemi ayrık optimizasyon problemi (discrete optimization problem) olarak isimlendirilir. Tasarım değişkenlerinin sürekli değer yerine tam sayı değerlerin aldığı ayrık optimizasyon problemlerine; graf teorisi ve rotalama kapsamında ele alınan GSP, minimum yayılan ağaç problemi (minimum spanning tree problem), araç rotalama (vehicle routing), havayolu planlama (airline scheduling) ve sırt çantası problemi (knapsack problem) örnek olarak verilebilir. Eğer tasarım değişkenleri sürekli ise optimizasyon, sürekli optimizasyon problemi (continuous optimization problem) olarak isimlendirilmektedir. Karışık optimizasyon problemleri ise (mixed optimization problems) tasarım değişkenlerinin hem ayrık hem de sürekli olduğunda bir sınıflandırma olarak nitelendirilebilir (Eldem, 2014). Şekil 3.21’ de optimizasyon problemlerinin sınıflandırılması gösterilmektedir.



Şekil 3.21. Optimizasyon problemlerinin sınıflandırılması (Eldem, 2014).

Optimizasyon problemlerinin çözümünde kullanılan algoritmalar iki kategoride değerlendirilebilir: Bunlardan ilki belirli bir prosedürü takip eden, takip edilen yolun, tasarım değişkenleri ve fonksiyon değerlerinin tekrarlanabildiği deterministik (rastgele olmayan, belirli) algoritmalardır. Yani algoritma ne zaman çalıştırılırsa çalıştırılsın aynı girdiler için aynı sonuçları her zaman üretebilen algoritmalar deterministik yaklaşımla açıklanmaktadır. Diğeri ise rastsallığı içinde barındıran stokastik (olasılıksal) algoritmalardır. Ayrıca, deterministik ve stokastik algoritmaların bir arada kullanıldığı hibrit yaklaşımlarda üçüncü bir tür optimizasyon algoritması olarak ele alınmaktadır. Optimizasyon problemlerinin çözümünde genellikle melez yaklaşımlar sıklıkla tercih edilebilmektedir. Şekil 3.22'de optimizasyon problemlerinin çözümünde kullanılan algoritmaların sınıflandırılması gösterilmektedir.

Şekil 3.22. Optimizasyon algoritmalarının sınıflandırılması (Eldem, 2014).

Tez kapsamında ele alınan yöntem olasılıksal algoritmalar sınıfında değerlendirildiğinden stokastik algoritmalar alt başlığı kısaca açıklanmıştır.

Stokastik algoritmalar genellikle aralarında küçük farklar olan sezgisel (heuristic) ve metasezgisel (metaheuristic) olarak iki sınıfta incelenir. Sezgisel (heuristic) kavramı deneme yanılma yoluyla çözümün bulunması olarak ifade edilebilir. Optimizasyon problemlerinde deterministik algoritmalara nazaran çok daha makul zamanlarda kaliteli çözümleri bulabilen sezgisel algoritmalar, her zaman en uygun çözümü bulma garantisi verememektedirler.

3.4.2.Sezgisel ve Metasezgisel optimizasyon

Sezgisel optimizasyon, herhangi bir problemin optimizasyonunda alternatif çözüm yollarından en optimum sonuçlara ulaşabilmek için doğal fenomenlerden ilham alan optimizasyon teknikleridir. Sezgisel optimizasyon çözümünde kullanılan algoritmalar, yakınsama özelliğine sahip olup, her zaman optimum çözümü garanti edememektedirler. Yakınsama özelliğinden dolayı optimum sonuca yakın sonuçlar üretebilmektedirler.

Sezgisel algoritmaların değerlendirilmesinde kullanılan kriterler aşağıdaki gibi olmalıdır (Karaboğa, 2011):

Çözüm Kalitesi ve Hesaplama Zamanı: Çözüm kalitesi ve hesaplama zamanı bir algoritmanın etkinliğinin değerlendirilmesi için önemli kriterlerdir. Bundan dolayı bir algoritma ayarlanabilir parametreler setine sahip olmalı ve bu parametreler kullanıcıya etkinlik açısında hesaplama maliyeti ile çözüm kalitesi arasında bir vurgulamanın yapılabilmesine imkân vermelidir. Diğer bir deyişle çözüm kalitesi ile hesap zamanı arasındaki ilişki kontrol edilebilmelidir.

Algoritma Basitliği ve Gerçeklenebilirlik: Algoritma prensipleri basit olmalı ve genel olarak uygulanabilir olmalıdır. Bu durum problem yapısı ile ilgili başlangıçta çok az bilgiye sahip olunması halinde bile algoritmanın yeni alanlara kolaylıkla uygulanabilmesini sağlar.

Esneklik: Algoritmalar modelde, sınırlamalarda ve amaç fonksiyonlarında yapılacak değişiklikleri kolayca karşılayabilmelidir.

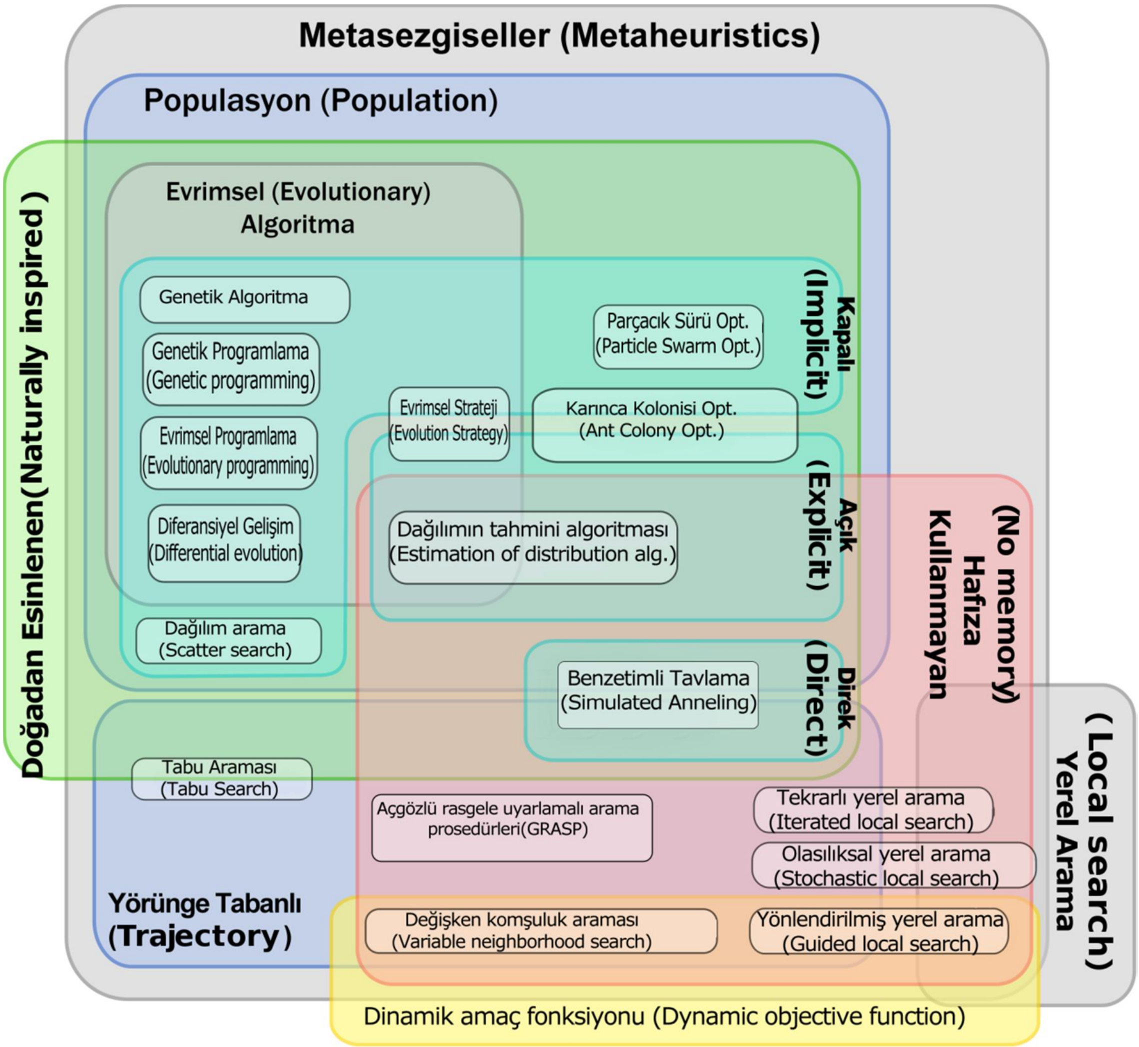
Dinçlik: Yöntem başlangıç çözümünün seçimine sahip olmaksızın her zaman yüksek kaliteli, kabul edilebilir çözümleri üretebilme kabiliyetine sahip olmalıdır.

Basitlik ve Analiz Edilebilirlik: Karmaşık algoritmalar, esneklik ve çözüm kalitesi açısından basit algoritmalardan daha zor analiz edilebilmektedir. Algoritma kolayca analiz edilebilir olmalıdır.

Etkileşimli Hesaplama ve Teknoloji Değişimleri: Algoritma içinde insan-makine etkileşimini kullanma fikri çoğu sistemde yaygın olarak gerçekleştirilmektedir. Herkesçe bilindiği gibi iyi bir kullanıcı arayüzü herhangi bir bilgisayar sistemini veya algoritmayı daha çekici yapmaktadır. Bunun en önemli avantajı çözümleri grafiksel olarak sergilenebilmesidir.

Metasezgisel algoritmalar, sezgisel optimizasyon algoritmalarının gelişmişi olarak ifade edilir. Temel sezgisel yöntemlerin bir arada kullanılmasıyla ortaya çıkmışlardır. Yüksek seviye anlamında kullanılan meta ifadesi ile bu algoritmalar basit sezgisel algoritmalardan daha iyi performans sergilemektedirler. Ayrıca tüm metasezgisel algoritmalar, rastgelelik ve yerel aramayı değişimli olarak kullanmaktadırlar. Rastgelelik, optimizasyon algoritmalarının çözümlerini yerel aramalardan kurtararak küresel ölçekli aramalar yaparak iyileştirmeyi sağlamaktadırlar (Eldem, 2014). Şekil 3.23’te metasezgisel algoritmaların sınıflandırılması gösterilmektedir.

Çoğu sezgisel algoritmalar probleme bağımlı algoritmalardır. Bir problem için en iyi performansı gösterirken diğer bir problem için aynı şekilde başarılı olmayabilir. Metasezgisellerin tüm problemleri kapsayıcı olması konusunda yeni yöntemler geliştirilmektedir. Bu yaklaşımlar sosyal, biyoloji, zooloji, fizik, bilgisayar ve karar verme gibi bilimler temel alınarak türetilmiştir. Bundan dolayı bu tür yaklaşımlara modern sezgisel yaklaşımlar ya da yapay zekâ yaklaşımları adı verilmektedir (Eldem, 2014).



Şekil 3.23. Metasezgisel algoritmaların cinslerine göre sınıflandırılması (Eldem, 2014)

Metasezgisel algoritmaların karakteristik özellikleri aşağıdaki gibi sıralanabilir (Blum ve Roli, 2003):

- Metasezgiseller arama süreçlerine rehberlik eden stratejilerdir.
- Arama uzayındaki keşiflerle en iyi ya da ona en yakın çözümlerin bulunması amacını taşır.

- Metasezgisel algoritmaları oluşturan teknikler, basit yerel arama prosedüründen karmaşık öğrenim süreçleri aralığında bir değişim gösterir.
- Metasezgisel algoritmalar tahmine dayalı rastgelelik içerirler.
- Bulunan bir çözüme takılıp kalmayı önleyici mekanizmalar barındırırlar.
- Metasezgiseller probleme özgü değildir.
- Farklı yöntemlerin kullanılmasıyla arama uzayını keşfeden yüksek seviyeli stratejilerdir.

Bu tez çalışması kapsamında ele alınan TSA algoritması doğa esinli-popülasyon tabanlı metasezgiseller sınıfında değerlendirilmektedir.

3.4.3.Doğa esinli sezgiseller

Doğa Esinli Sezgiseller (Nature Inspired Heuristics) (DES), gerçek doğadaki bazı olaylar ve onların isleyişinden esinlenilmiş mekanizmalardır. DES'lerin son yıllarda popülerleşmesinin sebebi, bir çok optimizasyon probleminin oldukça karmaşık olmasıdır. Farklı DES'ler sürekli olarak yayınlanmakta ve bunların bazıları da oldukça yüksek kalitede sonuçlar vermektedirler (Gazioğlu, 2015).

Literatürde bulunan birçok DES iki sınıf altında toplanabilir:

1. Tek noktadan başlayan arama teknikleri
2. Birçok noktadan başlayan arama teknikleri

Birinci grup içerisindeki en çok kullanılan metotlar arasında Kirkpatrick (1984) tarafından önerilen Isıl İşlem Algoritması (Simulated Annealing), Glover (1986) tarafından ortaya atılan ve Hansen (1986) tarafından geliştirilen Tabu Araması (Tabu Search) ve Feo ve Resende (1995) tarafından geliştirilen Açgözlü Rastgele Adaptif Arama Prosedürü (Greedy Randomized Adaptive Search Procedure) algoritmaları gelir.

İkinci grupta ise Kıran (2015) tarafından önerilen Ağaç-Tohum Algoritması(Tree-Seed Algorithm), Sürü Zekası Algoritmaları (Dorigo ve ark. (1991) tarafından önerilen Karınca Kolonisi Optimizasyonu (KKO) (Ant Colony Optimization), Eberhart ve Kennedy (1995) tarafından geliştirilmiş Parçacık Sürü Optimizasyonu (PSO) (Particle Swarm Optimizasyonu), Karaboga (2005) tarafından geliştirilen Yapay Arı Kolonisi Algoritması(Artificial Bee Colony Algorithm) vs.) ve Farmer ve ark. (1986) tarafından önerilen Yapay Bağışıklık Sistemi (Artificial Immune System) teknikleri bulunur.

Kuş sürülerinin havada süzülmeleri ve farklı şekiller almaları, karıncaların yiyecek aramaları, balık sürülerinin beraberce yüzmeleri ve kaçışmaları, bal arılarının danslarla haberleşmeleri gibi sürü halinde gerçekleştirilen davranışları keşfedilmesiyle biyologlar ve bilgisayar uzmanları bu sürülerin davranışları ve davranışlarının modellenmesi arasındaki iletişim mantığının üzerine çalışmalar yapmaya başlamışlardır. Bu çalışmalar, sürülerin davranışlarının sistem ve modellere uyarlandığı yaklaşımların gelişmesine yol açmıştır. Sürü zekası yaklaşımları adı verilen bu yaklaşımlar optimizasyon problemlerinde, robotik konularında ve askeri uygulamalarda gösterdikleri başarılar ile fazlaca ilgi çekmekte ve bu konu üzerindeki çalışmalar gün geçtikçe artmaktadır. Sürü zekâsı, kısaca özerk yapıdaki basit bireyler grubunun üst seviye de kolektif bir zekâ geliştirmesi şeklinde tanımlanabilir.

Sürü zekâsı doğada karınca, arı, kuş ve balık gibi canlıların bireysel olarak gerçekleştiremediği ancak grup halinde hareket ederek gerçekleştirebildikleri faaliyetleri örnek alır. Sürü zekâsı temelli algoritmaların en popüler olanlarına Dorigo ve ark. (1991) tarafından karıncaların yiyecek arama da gösterdikleri zeki davranışı modelleyen karınca koloni optimizasyon (ant colony optimization) algoritması, kuş ve balık sürülerinin davranışlarını temel alan Eberhart ve Kennedy (1995) tarafından geliştirilmiş parçacık sürü optimizasyon (particle swarm optimization) algoritması, doğal bağışıklık sisteminden esinlenerek Farmer ve ark. (1986) tarafından önerilen yapay bağışıklık algoritması (artificial immune system) ve bakterilerin davranışlarının Passino (2002) tarafından modellendiği bakteri optimizasyon algoritması (bacterial foraging optimization) gösterilebilir.

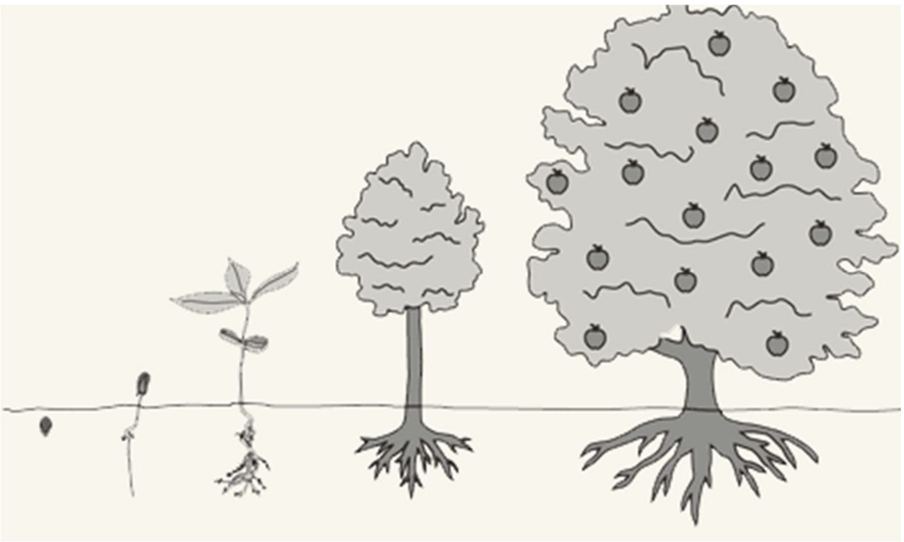
Arıların davranışlarına dayalı sürü zekâsı yaklaşımları 2000’li yılların başlarında başlamıştır. Bilim insanları arıların kraliçe arı benzetim modellerine, dans davranışlarına, görev paylaşımlarına, yuva yeri seçimlerine, üreme süreçlerine, navigasyon davranışlarına ve yiyecek kaynakları araştırma davranışlarına dayalı modeller geliştirmişlerdir(Akay, 2009). Bunlardan arıların yiyecek kaynağı aramalarındaki zeki davranışları modelleyen algoritmalardan en yaygın kullanılanlarına Lucic ve Teodorovic (2001)’in arı sistemi (Bee System) adını verdikleri yaklaşım, Teodorović ve Dell’Orco (2005)’nun gezinti eşleme probleminin çözümü için önerdikleri arı koloni optimizasyonu (Bee Colony Optimization System), (Yang, 2005)iki boyutlu nümerik fonksiyonların çözümüne yönelik sunduğu sanal arı algoritması (virtual bee algorithm) ve Pham ve ark. (2005) arıların yiyecek arama davranışlarını simule ederek geliştirdikleri arılar algoritması (bees algorithm) örnekleri verilebilir.

Sürü zekâsı tanımı ilk kez robotik alanında Beni ve Wang (1993) tarafından dile getirilmiştir. Sürü zekâsı, merkezi bir noktadan yönetilmeyen ve kendi kendine hareket eden bireylerin oluşturduğu toplulukların zeki bir şekilde hareket etmesini tarif eder. Her bir birey kendi yerel çevresiyle iletişime geçerek tüm sürü içerisinde bir haberleşme ağı oluşturur ve bu haberleşme ağı sayesinde sürü zeki bir şekilde hareket eden tek bir parça gibi görünür. Sürü zekâsı tanımı biyolojik canlı topluluklarında görülür. Balıklar, kuşlar, arılar, bakteriler, karıncalar ve daha birçok biyolojik grup sürü halinde bir zekâ göstermektedir. Bu biyolojik sistemlerin hareketleri incelenerek bilgisayar bilimleri içerisinde birçok sürü zekâsı algoritması geliştirilmiştir(Şahin, 2014). Eberhart ve Kennedy (1995) tarafından geliştirilmiş Parçacık Sürü Optimizasyonu (Particle Swarm Optimization), Dorigo ve ark. (1991) tarafından geliştirilmiş Karınca Kolonisi Optimizasyonu (Ant Colony Optimization), Karaboga (2005) tarafından geliştirilmiş Yapay Arı Kolonisi Algoritması (Artifical Bee Colony Algorithm) gibi birçok sürü zekâsı algoritması geliştirilmiş ve eniyileme problemlerinde sıklıkla kullanılır hale gelmişlerdir.

3.5.Ağaç-Tohum Algoritması(Tree-Seed Algorithm)

Ağaç-Tohum algoritması bir doğa esinli zeki optimizasyon algoritmasıdır. Ağaç-Tohum algoritması sürekli optimizasyon problemlerinin çözümü için ağaçlar ile tohumlar arasındaki ilişkiden yola çıkılarak önerilmiştir.

Tohumlu bir bitki olan ağaçların üreme ve yayılma organı tohumdur. Tohumlar yapılarına uygun olarak taşınır. Bazı tohumlar kuşların tüylerine, hayvanların kürklerine ve giysilere tutunarak farklı yerlere taşınır. Böylece taşınılan noktada uygun yetişme ortamı bulan tohum büyüyerek ağaç olur. Şekil 3.24'te doğadaki tohumdan ağaca dönüşüm sürecinin zamana bağlı gösterimi görülmektedir.

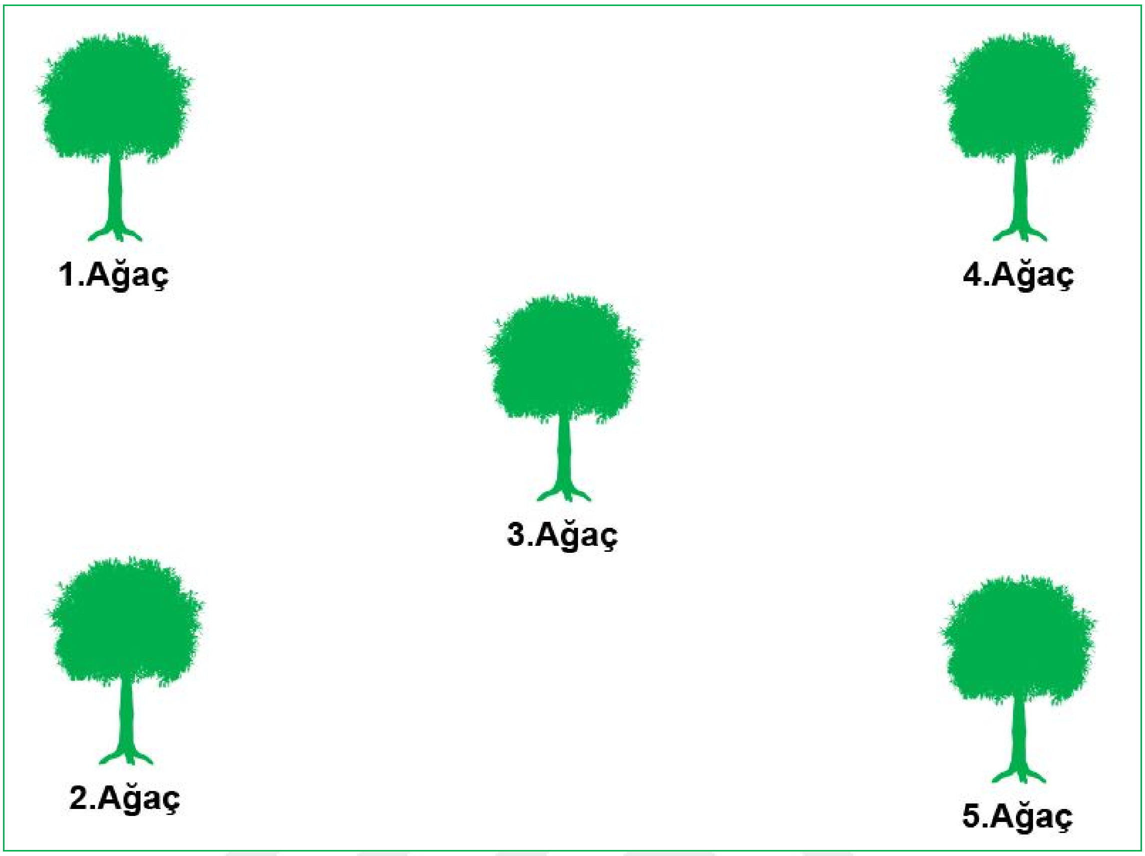


Şekil 3.24. Doğadaki tohumdan ağaca dönüşüm sürecinin zamana bağlı gösterimi

Optimizasyon problemlerini çözmek için ilk üretilen sonuçları ağaçlar olarak düşünürsek, daha sonra her ağacın kendi tohumları arasında açgözlü yaklaşım uygulanır ve en iyi tohum belirlenir. En iyi tohum, kendi ağacından daha iyiyse artık ağaç yok olur ve en iyi tohum o ağacın yerini alır.

Algoritmanın iki ana unsuru vardır, keşfetme ve faydalanma. Keşif aşamasında arama uzayına dağılmış rastgele noktalarda yerleşmiş ağaçlar bulunmaktadır, faydalanma aşamasında ise ağaçlar ile aynı özellikteki tohumlar kullanılmaktadır.

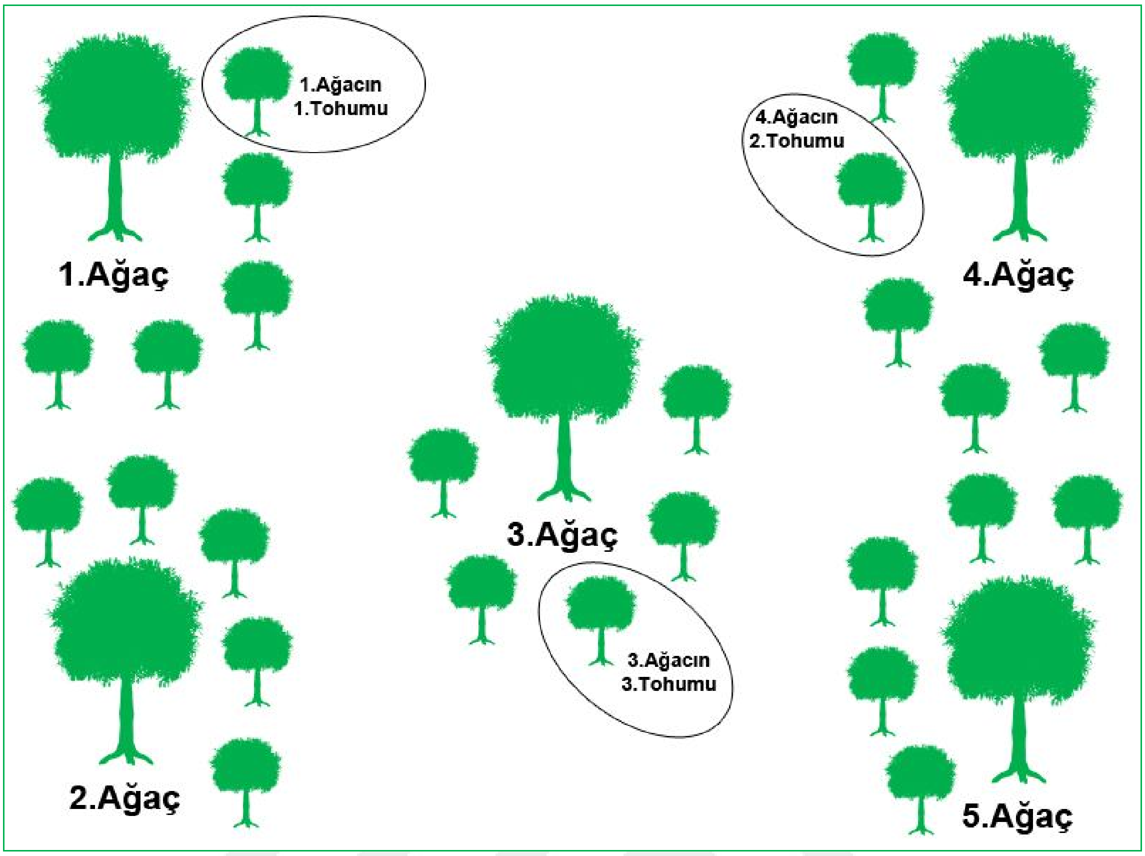
Şekil 3.25'te arama uzayına rastgele dağılmış temsili ağaçlar(Ağaç sayısı=5 olduğunda) görülmektedir. Şekil 3.26'da 5 ağaç için 5'er tohum üretildiğinde arama uzayının durumu görülmektedir. Şekil 3.27'de birinci iterasyon sonucunda en iyi tohumların ağaç olması sonrası arama uzayının durumu görülmektedir.



Şekil 3.25. Arama uzayına rastgele dağılmış temsili ağaçlar(Ağaç sayısı=5 olduğunda)

Her ağacın tohum sayısı rastgele olarak belirlenmektedir. Araştırma eğilimi(search tendency, ST) parametresi ise tohum oluşumu sırasında en iyi ağaç ile rastgele seçilecek ağaç arasında seçim yapmak için kullanılmaktadır. Araştırma eğilimi parametresi ile algoritmanın yerel bir yoğunlaşma içine girmesi amaçlanmış uygun değer veya uygun değere yakınsama yapması sağlanmıştır.

Araştırmanın popülasyondan seçilen rastgele seçilecek ağaca doğru mu yoksa ağaç topluluğundaki(ormandaki) en iyi çözüm değerine sahip ağaca doğru mu yapılacağına kısacası tohumun lokasyonunu belirlemede en iyi ve rastgele seçilen ağaçların etkilerini ölçmek için kullanılır.



Şekil 3.26. 5 ağaç için 5'er tohum üretildiğinde arama uzayının durumu

Tohum oluşumu sırasında araştırma eğilimi parametresine göre 3.1 ve 3.2 formülleri kullanılır. Araştırma eğilimi parametresi 0-1 arasında bir sayı olarak belirlenir. 0-1 arasında üretilen rastgele bir sayı eğer araştırma eğilimi parametresinden küçükse formül 3.1, büyükse formül 3.2 uygulanır. Araştırma eğilimi parametresi büyüdükçe faydalanma(exploitation) ve hızlı yakınsama sağlanırken, küçüldükçe yavaş yakınsama ve keşfetme(exploration) sağlanmış olur. Algoritmanın keşif ve faydalanma kapasitesi araştırma eğilimi parametresi ile kontrol edilebilmektedir.

$$S_{i,j} = T_{i,j} + \alpha_{i,j} x (B_j - T_{r,j}) \qquad (3.1)$$

$$S_{i,j} = T_{i,j} + \alpha_{i,j} x (T_{i,j} - T_{r,j}) \qquad (3.2)$$

$S_{i,j}$: i. tohumun j.boyutu (i. ağaç için üretilmektedir)

$T_{i,j}$: i.ağacın, j.boyutu

$\alpha_{i,j}$: -1,1 arasında üretilen rastgele bir değer

B_j: Şimdiye kadar elde edilen en iyi ağacın j.boyutu

$T_{r,j}$: r.ağacın j.boyutu; r rastgele seçilen i'den farklı bir ağaç olmalıdır

Başlangıçta çözülecek optimizasyon probleminin sınırları içerisinde rastgele ağaçlar oluşturulur.

$$T_{i,j} = L_{j,min} + r_{i,j}\left(H_{j,max} - L_{j,min}\right) \quad (3.3)$$

$L_{j,min}$: Arama uzayının alt sınırı(j.parametre için)

$H_{j,max}$:Arama uzayının üst sınırı

$r_{i,j}$: 0-1 arasında üretilen rastgele sayı(i.ağacın j.parametresi için)

Minimizasyon problemleri için en iyi çözüm tüm ağaçların fonksiyon değerleri hesaplanarak bulunur.

$$B = ArgMin\{f(\vec{T_i})\}\ i = 1,2,\dots,N \quad (3.4)$$

Her ağaca ait tohum sayısının 1'den az olmamak kaydıyla popülasyon boyutunun yüzde 10'u ile yüzde 25'i arasında bir değer olması uygun görülmüştür(Kıran, 2015).

Yapılan testlerde araştırma eğilimi parametresi 0.1 olarak önerilmiştir. Problemin boyutu büyüdükçe algoritmanın iyi sonuçlar üretemediği de açıklanmıştır. Bunun sebebinin boyutun artmasına bağlı olarak arama uzayının üstel artması gösterilmiştir. Ağaç-Tohum algoritması unimodal(tek optimum değeri olan) fonksiyonları çözmekte, multimodal(çok yerel optimum değeri olan) fonksiyonları çözmeye göre daha başarılıdır. Ağaç-Tohum algoritması literatürde yaygın olarak kullanılan 24 kıyas fonksiyonunu ve eşikleme problemini çözmede kabul edilebilir ve karşılaştırılabilir düzeyde sonuçlar üreterek başarısını ortaya koymuştur(Kıran, 2015).

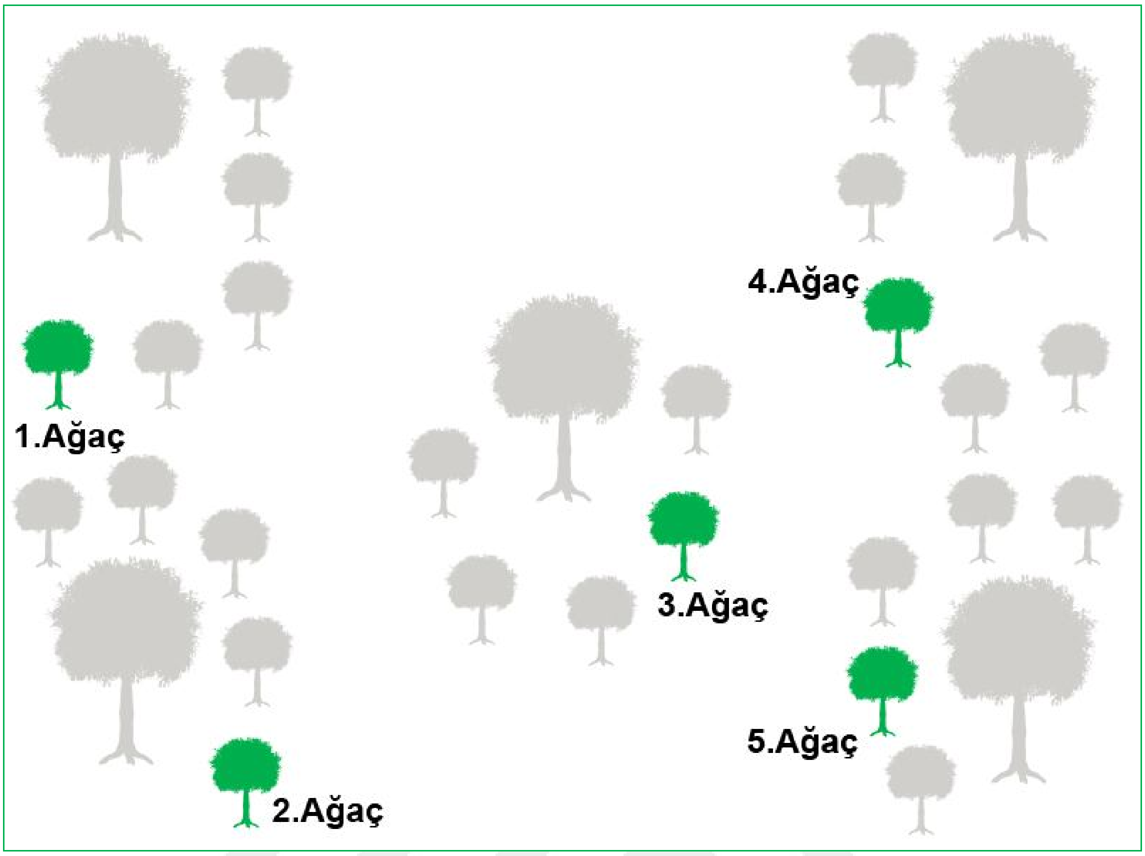
3.5.1.Ağaç-Tohum algoritmasının algoritmik çerçevesi

1. Algoritmanın Başlatılması
    a. Popülasyon sayısı belirlenir(N).
    b. Araştırma eğilimi parametresi belirlenir(ST).
    c. Problemin boyutu belirlenir(D).
    d. Sonlandırma koşulu belirlenir.
    e. Araştırma uzayına D boyutlu N adet ağaç sınırlara uygun olarak dağıtılır.
    f. Üretilen ağaçların kalitesi hesaplanır.
    g. En iyi ağaç seçilir, parametreleri alınır.
2. Tohumlarla arama işlemi

FOR tüm ağaçlar için,

```
        İlgili ağaç için kaç tohum üretileceği rastgele olarak belirlenir(x)
        FOR ilgili ağacın x adet tohumu için
            FOR ilgili tohumun i=1…D boyutu için
                IF (rand<ST)
                İlgili boyut 3.1.formüle göre hesaplanır
                ELSE
                İlgili boyut 3.2.formüle göre hesaplanır
                END IF
            END FOR
        END FOR
        En iyi tohum seçilir ve kendi ağacıyla karşılaştırılır.
        Eğer en iyi tohum, ağacından daha iyise tohum ağaç olur.
    END FOR
```

3. En iyi çözümün seçilmesi
    a. Popülasyonun en iyi çözümü seçilir
    b. Eğer mevcut en iyi çözüm, şimdiye kadar ki en iyi çözümden daha iyiyse yer değiştirir ve parametreleri aktarılır.
4. Sonlandırma kıstasının kontrol edilmesi
    a. Sonlandırma kıstasına erişilmediyse 2.adıma geri dönülür.
5. Raporlama
    a. En iyi çözüm yazdırılır.(Kıran, 2015)



Şekil 3.27. Birinci iterasyon sonucunda en iyi tohumların ağaç olması sonrası arama uzayının durumu

3.5.2.Ağaç-Tohum algoritmasının parametreleri

Ağaç Sayısı: Popülasyon sayısı olarakta adlandırılabilir. Çözüme ulaşmak için ilk etapta oluşturulacak aday çözüm sayısını simgeler. Herhangi bir sınırlaması yoktur. Algoritmanın tanıtımı yapılırken 10,20,30,40,50 alınarak testler yapılmıştır.

Ağaç Boyutu: Problemin tipine göre değişiklik gösterir, kaç boyutlu bir problem optimize edilecekse ona ayarlanır.

Tohum Sayısı: Ağaç sayısının yüzde 10'undan az, yüzde 25'inden fazla olmaması önerilmektedir. Her ağacın tohum sayısı rastgele olarak farklı farklı seçilir. Örneğin ağaç sayısı 10 alındığı durumda en az 1 en fazla 3 tohum önerilmektedir.

Tohum Boyutu: Ağaç Boyutu ile aynı olmalıdır.

Araştırma Eğilimi(Search Tendency): Araştırma eğilimi parametresi 0-1 arasında bir sayı olarak belirlenir. Araştırma eğilimi parametresi büyüdükçe güçlü yerel arama ve hızlı yakınsama sağlanırken, küçüldükçe yavaş yakınsama ve global arama sağlanmış olur. Algoritmanın keşif ve faydalanma kapasitesi arama eğilimi parametresi ile kontrol edilmektedir. Arama eğilimi değeri olarak 0.1 önerilmektedir. Araştırmanın popülasyondan seçilen rastgele seçilecek ağaca doğru mu yoksa ağaç topluluğundaki(ormandaki) en iyi çözüm değerine sahip ağaca doğru mu yapılacağına kısacası tohumun lokasyonunu belirlemede en iyi ve rastgele seçilen ağaçların etkilerini ölçmek için kullanılır.

Sonlandırma Koşulu: Algoritmanın kaç adım çalıştırılacağını belirleyen kriterdir. Yapılan işleme göre değişiklik gösterebilir.

3.6.Kıyas Fonksiyonları

Geliştirilen algoritmayı test etmek için literatürde sıklıkla kullanılan kıyas fonksiyonlarının özellikle hız testi yapacağımızdan çok boyutlu olarak çalıştırılabilecek durumda olanları kullanılmıştır. Seçtiğimiz kıyas fonksiyonları Sphere, Ackley, Griewank, Rastrigin, Rosenbrock, Schwefel, Styblinski Tang,Sum of Different Powers, Sum Squares, Zakharov'dur.

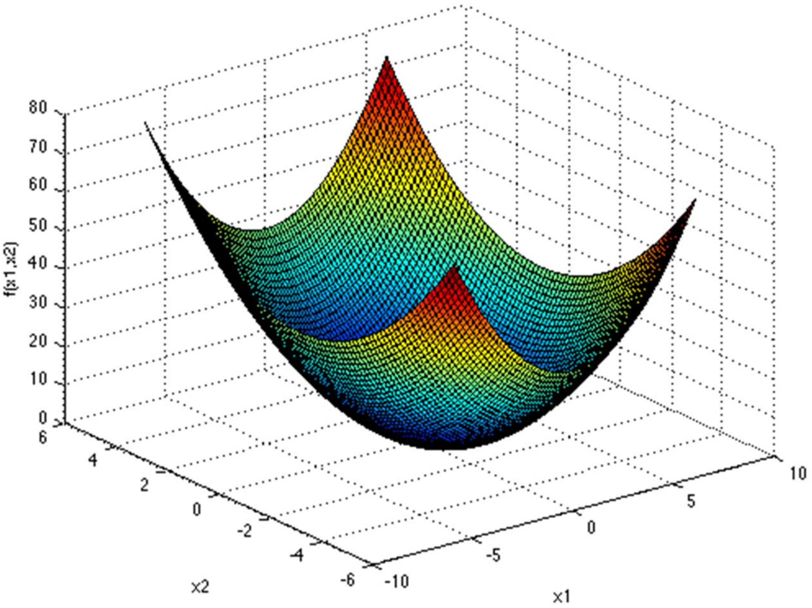
Çalışmamızda kullandığımız kıyas fonksiyonlarını olabildiğince farklı şekilli olanlar seçmeye çalıştık. Ackley, Griewank, Rastrigin ve Schwefel bir çok yerel minimumu olan fonksiyonlardır. Sphere, Sum of Different Powers, Sum Squares fonksiyonları konveks(dışbükey) kıyas fonksiyonlarındandır. Zakharov fonksiyonu levha(plaka) şekilli bir kıyas fonksiyonudur. Rosenbrock fonksiyonu vadi şekilli bir test fonksiyonudur. Styblinski-Tang fonksiyonu ise farklı şekilli bir kıyas fonksiyonudur.

Kıyas fonksiyonları seri uygulama için Matlab ortamında yazıldı, algoritmadaki ağaç ve tohumlar için aynı Matlab fonksiyon dosyası kullanılmıştır. Paralel uygulama için Ağaç ve Tohumlar için ayrı ayrı fonksiyon dosyaları hazırlanmıştır. Bu hazırlanan dosyalar CUDA derleyicisi ile derlenerek çalışır hale getirilmiş, üretilen PTX dosyaları da programa tanıtılmıştır.

3.6.1.Sphere fonksiyonu

$$f(x) = \sum_{i=1}^{d} x_i^2 \qquad (3.5)$$

En basit kıyas fonksiyonudur. Test bölgesi genellikle $-100<x_i<100$; i=1,…,n sınırları arasındadır. Küresel minimum; i=1,…,n ve $x_i=0$ için f(x)=0 noktasındadır. Fonksiyon küresel olmasından dolayı yerel minimuma takılma olasılığı oldukça düşüktür. Sphere fonksiyonu düzgün, unimodal (tek optimum nokta içeren) ve ileri derecede dışbükey(konveks) bir fonksiyondur. De Jong Fonksiyonu olarak ta bilinmektedir. Şekil 3.28'de Küre fonksiyonu'nun grafiği görülmektedir.



Şekil 3.28. Küre Fonksiyonu'nun Grafiği

Seri Matlab Kodu(Sphere.m):

```
function [y] = Sphere(x)
d = length(x);
total = 0;
for i = 1:d
        total = total + x(i)^2;
end
y = total;
end
```

Üretilen Ağaçlar için kullanılan Paralel C Kodu(Sphere.cu):

```
__global__ void Obj(double * objt, double * nt,int N,int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double total = 0.00;
   for (int j = 0; j < D; j++){
   total=(nt[i+(j*N)]*nt[i+(j*N)]);
   objt[i]=objt[i]+total;
}
}
```

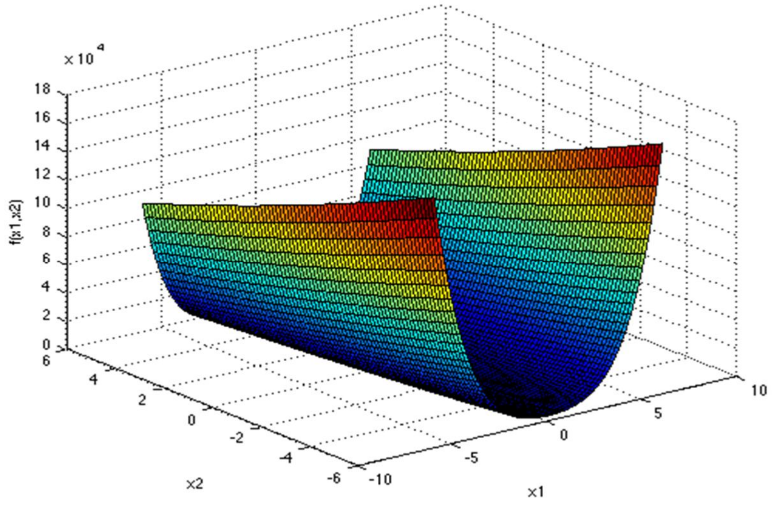
Üretilen Tohumlar için kullanılan Paralel C Kodu(Spheres.cu):

```
__global__ void Objs(double * objs, double * seeds,int N,int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double total = 0.00;
   for (int j = 0; j < D; j++){
  total=seeds[(i/N)+i+(i/N)*((N*(D-1))-1)+(j*N)]*seeds[(i/N)+i+(i/N)*((N*(D-1))-1)+(j*N)];
   objs[i]=objs[i]+total;
}
}
```

3.6.2.Rosenbrock fonksiyonu

$$f(x) = \sum_{i=1}^{d-1}[100(x_{i+1} - x_i^2)^2 - (x_i - 1)^2] \tag{3.6}$$

Rosenbrock vadisi veya muz fonksiyonu olarak da anılır. Küresel uygun değer uzun, geniş, parabolik şekilli vadide yatar. Test bölgesi genellikle $-2.048 \leq x_i \leq 2.048$, i=1, ..., n sınırları arasında aranır. Küresel minimum x_i=1, i=1, ..., n için f(x)=0 noktasında bulunmaktadır. Rosenbrock fonksiyonu gradyan tabanlı optimizasyon algoritmaları için popüler bir test fonksiyonudur. Şekil 3.29'da Rosenbrock fonksiyonu'nun grafiği görülmektedir.



Şekil 3.29. Rosenbrock fonksiyonu'nun grafiği

Seri Matlab Kodu(Rosenbrock.m):

```
function [y] = Rosenbrock(x)
d = length(x);
total = 0;
for i = 1:(d-1)
        total = total + 100*(x(i+1)-x(i)^2)^2 + (x(i)-1)^2;
end
y = total; end
```

Üretilen Ağaçlar için kullanılan Paralel C Kodu(Rosenbrock.cu):

```
__global__ void Obj(double * objt, double * nt, int N, int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double total = 0.00;
   for (int j = 0; j < D-1; j++){
   double bir=nt[i+(j*N)];
   double iki=nt[i+(j*N)+N];
   total=100*((iki-(bir*bir))*((iki-(bir*bir))))+((bir-1)*(bir-1));
   objt[i]=objt[i]+total;
}
}
```

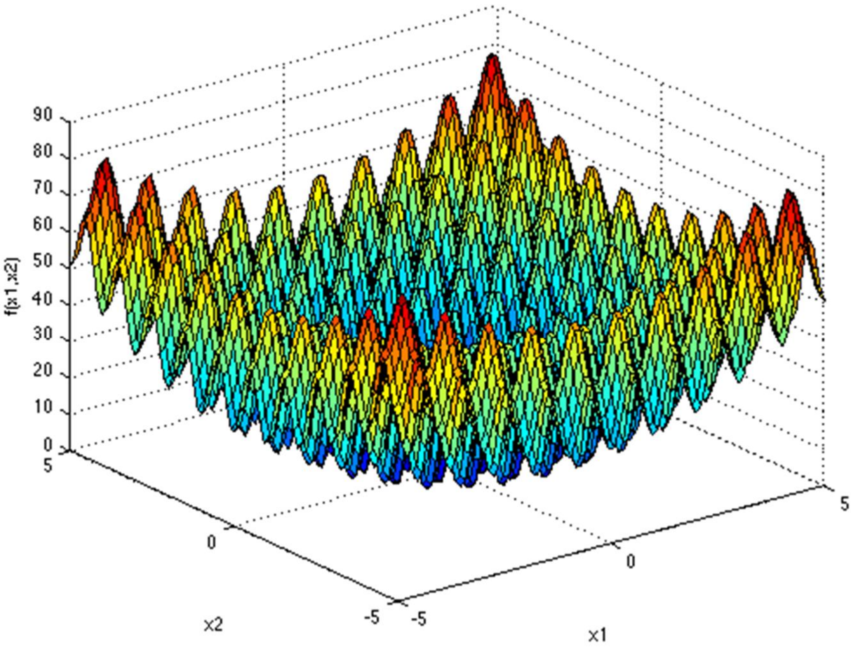
Üretilen Tohumlar için kullanılan Paralel C Kodu(Rosenbrocks.cu):

```
__global__ void Objs(double * objs, double * seeds,int N,int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double total = 0.00;
   for (int j = 0; j < D-1; j++){
   double bir=seeds[i+(i/N)*(D-1)*N+(j*N)];
   double iki=seeds[i+(i/N)*(D-1)*N+(j*N)+N];
   total=100*((iki-(bir*bir))*((iki-(bir*bir))))+((bir-1)*(bir-1));
   objs[i]=objs[i]+total;
}
}
```

3.6.3.Rastrigin fonksiyonu

$$f(x) = 10d + \sum_{i=1}^{d} x_i^2 - 10\cos(2\pi x_i) \qquad (3.7)$$

Çok modelli ve minimum noktaları düzenli dağılmış bir fonksiyondur. Test bölgesi genellikle $-5.12 \leq x_i \leq 5.12$, i=1, ..., n sınırları arasında aranır. Küresel minimum $x_i=0$, i=1, ..., n için f(x)=0 noktasında bulunmaktadır. Doğrusal olmayan multimodal(bir çok yerel minimum ancak bir genel minimum içeren) fonksiyon türlerinin tipik örneklerindendir. Rastrigin fonksiyonu geniş arama uzayı ve birçok yerel minimuma sahip olduğu için zor bir problem türüdür. Fonksiyon ileri derecede multimodal'dır. Şekil 3.30'da Rastrigin fonksiyonu'nun grafiği görülmektedir.



Şekil 3.30. Rastrigin fonksiyonu'nun grafiği

Seri Matlab Kodu(Rastrigin.m):

```
function [y] = Rastrigin(x)
d = length(x);
total = 0;
for i = 1:d
        total = total + (x(i)^2 - 10*cos(2*pi*x(i)));
end
y = 10*d + total;
end
```

Üretilen Ağaçlar için kullanılan Paralel C Kodu(Rastrigin.cu):

```
__global__ void Obj(double * objt, double * nt,int N,int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double M_PI =3.14159265358979323846;
   double total = 0.00;
   for (int j = 0; j < D; j++){
   double bir=nt[i+(j*N)];
   total=((bir*bir)-(10*cos(2*M_PI*bir))+10);
   objt[i]=objt[i]+total;
}
}
```

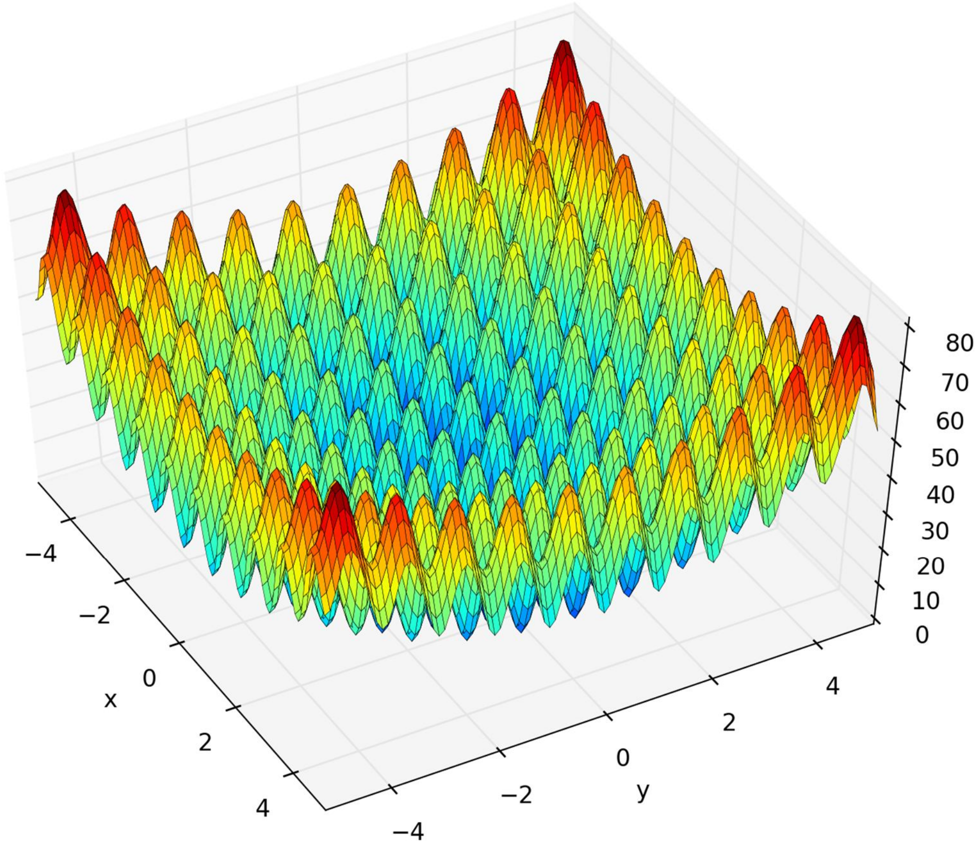
Üretilen Tohumlar için kullanılan Paralel C Kodu(Rastrigins.cu):

```
__global__ void Objs(double * objs, double * seeds,int N,int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double M_PI=3.14159265358979323846;
   double total = 0.00;
   for (int j = 0; j < D; j++){
   double bir=seeds[i+(i/N)*(D-1)*N+(j*N)];
   total=((bir*bir)-(10*cos(2*M_PI*bir))+10);
   objs[i]=objs[i]+total;
}
}
```

3.6.4.Griewank fonksiyonu

$$f(x) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \qquad (3.8)$$

Rastrigin fonksiyonuna benzer bir yapıda ama daha geniş dağılmış minimum noktalara sahip çok modelli bir fonksiyondur. Test bölgesi genellikle -600 ≤ x_i ≤ 600, i=1, ..., n sınırları arasında aranır. Küresel minimum x_i=0, i=1, ..., n için f(x)=0 noktasında bulunmaktadır. Doğrusal olmayan multimodal(bir çok yerel minimum ancak bir genel minimum içeren) fonksiyon türlerinin tipik örneklerindendir. Griewank fonksiyonunda, toplama terimi fonksiyona parabollük özelliği kazandırmaktadır. Bu fonksiyonda yerel minimum derecesi parabollük derecesinden daha üst seviyededir. Çarpım terimi temel alınarak arama uzayının boyutları artırılmakta ve yerel minimumlar azaltılmaktadır. Şekil 3.31'de Griewank fonksiyonu'nun grafiği görülmektedir.



Şekil 3.31. Griewank fonksiyonu'nun grafiği

Seri Matlab Kodu(Griewank.m):

```
function [y] = Griewank(x)
d = length(x);
total1 = 0;
total2 = 1;
for i = 1:d
        total1 = total1 + x(i)^2/4000;
        total2 = total2 * cos(x(i)/sqrt(i));
end
y = total1 - total2 + 1;
end
```

Üretilen Ağaçlar için kullanılan Paralel C Kodu(Griewank.cu):

```
__global__ void Obj(double * objt, double * nt,int N,int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double total1 = 0.00;
   double total2 = 1.00;
   for (int j = 0; j < D; j++){
   double bir=nt[i+(j*N)];
   total1=total1+((bir*bir)/4000);
         total2=total2*(cos(bir/sqrt((double)j+1)));
}
   objt[i]=objt[i]+total1-total2+1;
}
```

Üretilen Tohumlar için kullanılan Paralel C Kodu(Griewanks.cu):

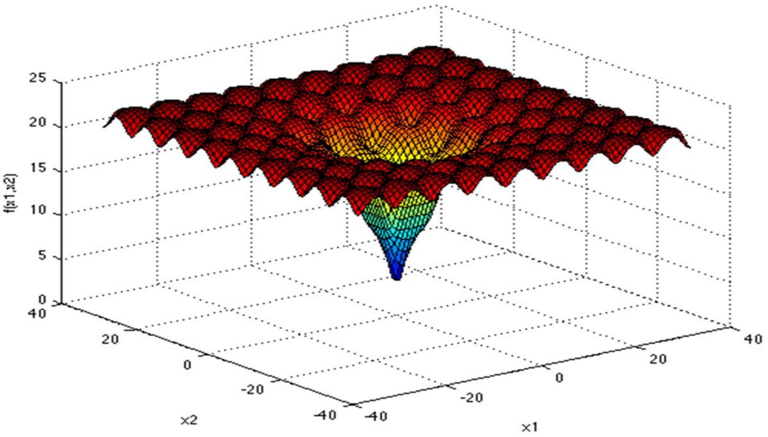
```
__global__ void Objs(double * objs, double * seeds,int N,int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double total1 = 0.00;
   double total2 = 1.00;
   for (int j = 0; j < D; j++){
   double bir=seeds[i+(i/N)*(D-1)*N+(j*N)];
   total1=total1+((bir*bir)/4000);
         total2=total2*(cos(bir/sqrt((double)j+1)));
}
   objs[i]=objs[i]+total1-total2+1;

}
```

3.6.5.Ackley fonksiyonu

$$f(x) = -aexp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1) \quad (3.9)$$

Ackley fonksiyonu sürekli, diferensiyellenebilir, ölçeklenebilir, multimodal(bir çok yerel minimum ancak bir genel minimum içeren), ayrılabilir olmayan bir kıyas fonksiyonudur. Genellikle a=20, b=0.2 ve c=2 olarak önerilmektedir. Test bölgesi $-32.768 \leq x_i \leq 32.768$, i=1, ..., n sınırları arasında aranır. Küresel minimum $x_i=0$, i=1, ..., n için f(x)=0 noktasında bulunmaktadır. Şekil 3.32'de Ackley fonksiyonu'nun grafiği görülmektedir.



Şekil 3.32. Ackley fonksiyonu'nun grafiği

Seri Matlab Kodu(Ackley.m):

```
function [y] = Ackley(x)
a =  20;
b =  0.2;
c =  2*pi;
d = length(x);
total1 = 0;
total2 = 0;
for i = 1:d
        total1 = total1 + x(i)^2;
        total2 = total2 + cos(c*x(i));
end
term1 = -a * exp(-b*sqrt(total1/d));
term2 = -exp(total2/d);
y = term1 + term2 + a + exp(1);
end
```

Üretilen Ağaçlar için kullanılan Paralel C Kodu(Ackley.cu):

```
__global__ void Obj(double * objt, double * nt,int N,int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double total1 = 0.00;
          double total2 = 0.00;
   double M_PI=3.14159265358979323846;

          double a=20.00;
          double b=0.20;
          double c=2*M_PI;
   for (int j = 0; j < D; j++){
   double bir=nt[i+(j*N)];
   total1 = total1 + (bir*bir);
          total2 = total2 + cos(c*bir);
}
double term1 = -a * exp(-b*sqrt(total1/D));
double term2 = -exp(total2/D);
objt[i]=objt[i]+term1 + term2 + a + exp((double)1);
}
```

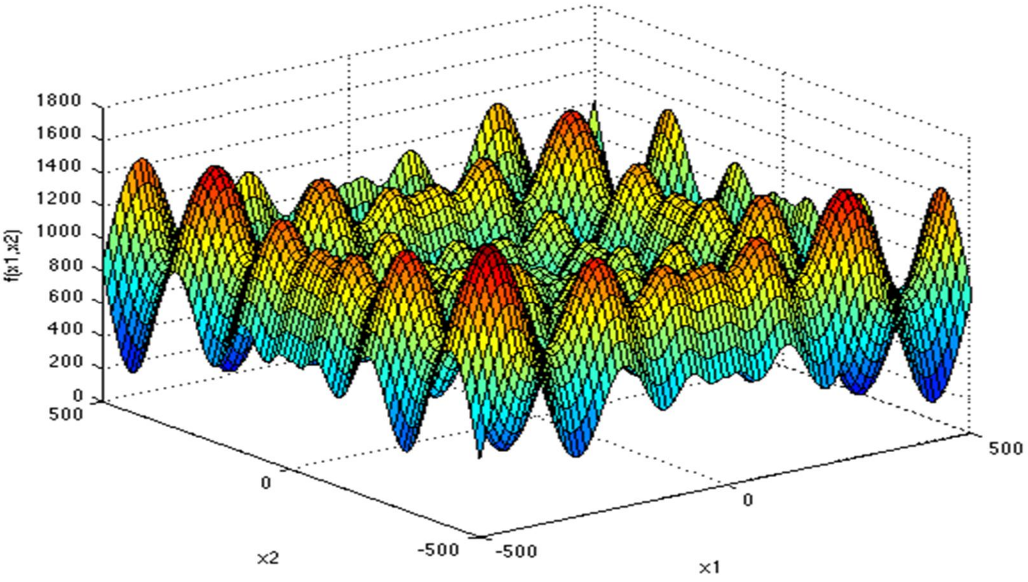
Üretilen Tohumlar için kullanılan Paralel C Kodu(Ackleys.cu):

```
__global__ void Objs(double * objs, double * seeds,int N,int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double total1 = 0.00;
          double total2 = 0.00;
   double M_PI=3.14159265358979323846;
          double a=20.00;
          double b=0.20;
          double c=2*M_PI;
   for (int j = 0; j < D; j++){
   double bir=seeds[i+(i/N)*(D-1)*N+(j*N)];
   total1 = total1 + (bir*bir);
          total2 = total2 + cos(c*bir);
}
double term1 = -a * exp(-b*sqrt(total1/D));
double term2 = -exp(total2/D);
objs[i]=objs[i]+term1 + term2 + a + exp((double)1);
}
```

3.6.6.Schwefel fonksiyonu

$$f(x) = 418.9829d - \sum_{i=1}^{d} x_i \sin(\sqrt{|x_i|}) \tag{3.10}$$

Test bölgesi genellikle -500 ≤ x_i ≤ 500, i=1, ..., n sınırları arasında aranır. Küresel minimum x_i=420.9687, i=1, ..., n için f(x)= - 418.9829 noktasında bulunmaktadır. Doğrusal olmayan multimodal(bir çok yerel minimum ancak bir genel minimum içeren) fonksiyon türlerinin tipik örneklerindendir. Schwefel fonksiyonu Rastrigin fonksiyonundan daha kolay olan ve genel minimumun çok uzağında ikinci en iyi minimuma sahip olan bir fonksiyondur. Şekil 3.33'de Schwefel fonksiyonu'nun grafiği görülmektedir.



Şekil 3.33. Schwefel fonksiyonu'nun grafiği

Seri Matlab Kodu(Schwefel.m):

```
function [y] = Schwefel(x)
d = length(x);
total = 0;
for i = 1:d
        total = total + x(i)*sin(sqrt(abs(x(i))));
end
y = 418.9829*d - total;
end
```

Üretilen Ağaçlar için kullanılan Paralel C Kodu(Schwefel.cu):

```
__global__ void Obj(double * objt, double * nt,int N,int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double total=0.00;
   for (int j = 0; j < D; j++){
   double bir=nt[i+(j*N)];
   total=total+bir*sin(sqrt(abs(bir)));
}
objt[i]=objt[i]+(418.9829*D)-total;
}
```

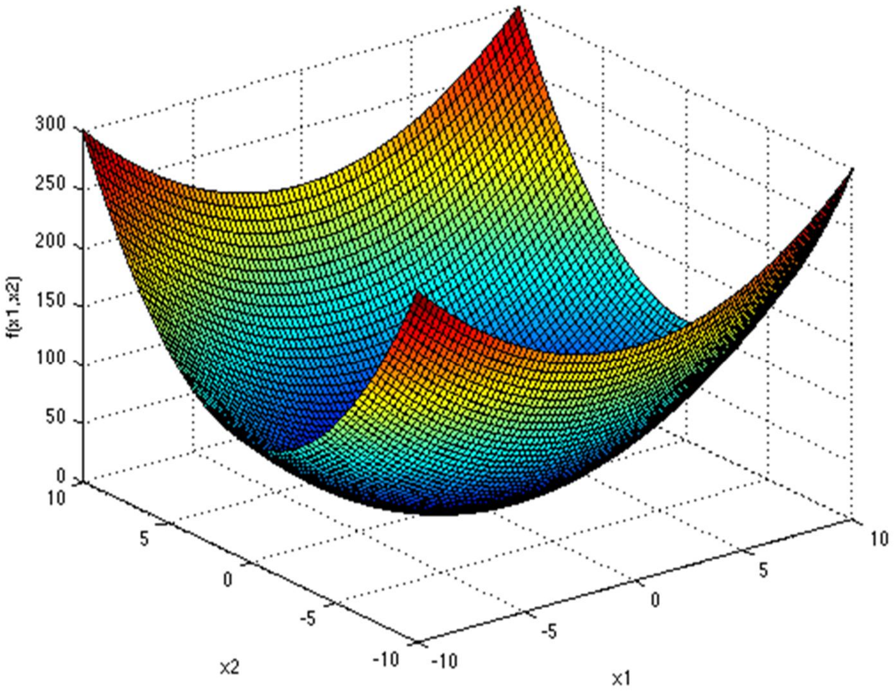
Üretilen Tohumlar için kullanılan Paralel C Kodu(Schwefel s.cu):

```
__global__ void Objs(double * objs, double * seeds,int N,int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double total=0.00;
   for (int j = 0; j < D; j++){
   double bir=seeds[i+(i/N)*(D-1)*N+(j*N)];
   total=total+bir*sin(sqrt(abs(bir)));
}
objs[i]=objs[i]+(418.9829*D)-total;
}
```

3.6.7.Sum Squares fonksiyonu

$$f(x) = \sum_{i=1}^{d} ix_i^2 \qquad (3.11)$$

Karelerin toplamı fonksiyonu, Eksen Paralel Hiper-Elipsoid fonksiyon olarak ta bilinir. Tek global minimumu hariç yerel minimumu yoktur. Sürekli, dışbükey, tek modlu, diferensiyellenebilir, ölçeklenebilir, ayrılabilir bir kıyas fonksiyonudur.Test bölgesi genellikle -10<x_i<10; i=1,…,n sınırları arasındadır. Küresel minimum; i=1,…,n ve x_i=0 için f(x)=0 noktasındadır. Şekil 3.34'de Sum Squares fonksiyonu'nun grafiği görülmektedir.



Şekil 3.34. Sum Squares fonksiyonu’nun grafiği

Seri Matlab Kodu(SumSquares.m):

```
function [y] = SumSquares(x)
d = length(x);
total = 0;
for i = 1:d
        total = total + i*x(i)^2;
end
y = total;
end
```

Üretilen Ağaçlar için kullanılan Paralel C Kodu(SumSquares.cu):

```
__global__ void Obj(double * objt, double * nt,int N,int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double total=0.00;
   for (int j = 0; j < D; j++){
   double bir=nt[i+(j*N)];
   total=total+(j+1)*(bir*bir);
}
objt[i]=objt[i]+total; }
```

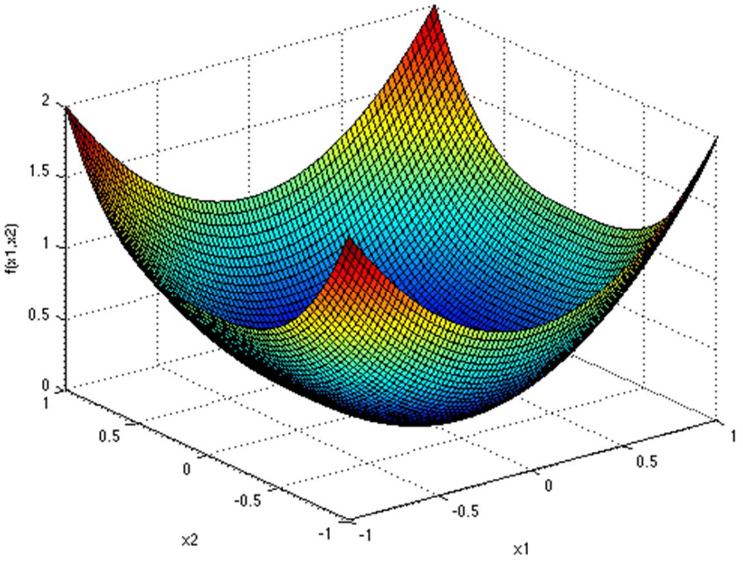
Üretilen Tohumlar için kullanılan Paralel C Kodu(SumSquaress.cu):

```
__global__ void Objs(double * objs, double * seeds,int N,int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double total=0.00;
   for (int j = 0; j < D; j++){
   double bir=seeds[i+(i/N)*(D-1)*N+(j*N)];
   total=total+(j+1)*(bir*bir);
}
objs[i]=objs[i]+total;
}
```

3.6.8.Sum of Different Powers fonksiyonu

$$f(x) = \sum_{i=1}^{d} |x_i|^{i+1} \qquad (3.12)$$

Sum of Different Powers fonksiyonu tek modlu bir fonksiyondur. Test bölgesi genellikle -1<x_i<1; i=1,…,n sınırları arasındadır. Küresel minimum; i=1,…,n ve x_i=0 için f(x)=0 noktasındadır. Şekil 3.35'de farklı kuvvetlerin toplamı fonksiyonu'nun grafiği görülmektedir.



Şekil 3.35. Sum of Different Powers fonksiyonu'nun grafiği

Seri Matlab Kodu(SumofDifferentPowers.m):

```
function [y] = SumofDifferentPowers(x)
d = length(x);
total = 0;
for i = 1:d
   total = total +(abs(x(i)))^(i+1);
end
y = total;
end
```

Üretilen Ağaçlar için kullanılan Paralel C Kodu(SumofDifferentPowers.cu):

```
__global__ void Obj(double * objt, double * nt,int N,int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double total=0.00;
   for (int j = 0; j < D; j++){
   double bir=nt[i+(j*N)];
         total = total +pow(abs(bir),(j+2));
}
objt[i]=objt[i]+total;
}
```

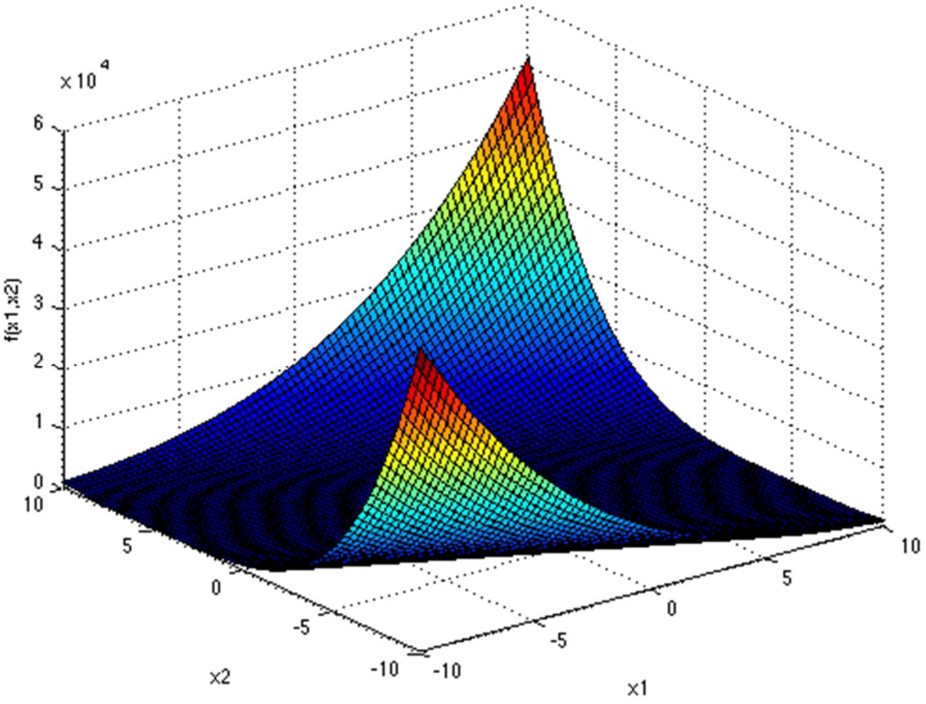
Üretilen Tohumlar için kullanılan Paralel C Kodu(SumofDifferentPowerss.cu):

```
__global__ void Objs(double * objs, double * seeds,int N,int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double total=0.00;
   for (int j = 0; j < D; j++){
   double bir=seeds[i+(i/N)*(D-1)*N+(j*N)];
         total = total +pow(abs(bir),(j+2));
}
objs[i]=objs[i]+total;
}
```

3.6.9.Zakharov fonksiyonu

$$f(x) = \sum_{i=1}^{d} x_i^2 + \left(\sum_{i=1}^{d} 0.5 i x_i\right)^2 + \left(\sum_{i=1}^{d} 0.5 i x_i\right)^4 \qquad (3.13)$$

Zakharov fonksiyonu sürekli, diferensiyellenebilir, ölçeklenebilir, multimodal(birçok yerel minimum ancak bir genel minimum içeren), ayrılabilir olmayan bir kıyas fonksiyonudur. Test bölgesi genellikle -5<x_i<10; i=1,…,n sınırları arasındadır. Küresel minimum; i=1,…,n ve x_i=0 için f(x)=0 noktasındadır. Şekil 3.36'da Zakharov fonksiyonu'nun grafiği görülmektedir.



Şekil 3.36. Zakharov fonksiyonu'nun grafiği

Seri Matlab Kodu(Zakharov.m):

```
function [y] = Zakharov(x)
d = length(x);
total1 = 0;
total2 = 0;
for i = 1:d
        total1 = total1 + x(i)^2;
        total2 = total2 + 0.5*i*x(i);
end
y = total1 + total2^2 + total2^4;
end
```

Üretilen Ağaçlar için kullanılan Paralel C Kodu(Zakharov.cu):

```
__global__ void Obj(double * objt, double * nt,int N,int D)
{
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  double total1 = 0.00;
      double total2 = 0.00;
  for (int j = 0; j < D; j++){
  double bir=nt[i+(j*N)];
  total1 = total1 + bir*bir;
      total2 = total2 + 0.5*(j+1)*bir;
}
objt[i]=objt[i]+total1 + pow(total2,2) + pow(total2,4);
}
```

Üretilen Tohumlar için kullanılan Paralel C Kodu(Zakharovs.cu):

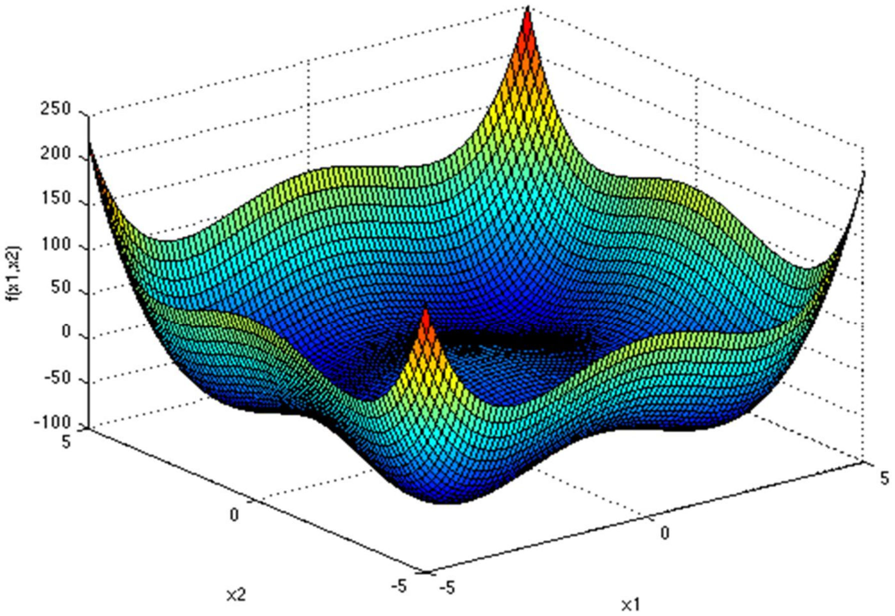
```
__global__ void Objs(double * objs, double * seeds,int N,int D)
{
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  double total1 = 0.00;
      double total2 = 0.00;
  for (int j = 0; j < D; j++){
  double bir=seeds[i+(i/N)*(D-1)*N+(j*N)];
      total1 = total1 + bir*bir;
      total2 = total2 + 0.5*(j+1)*bir;
}
objs[i]=objs[i]+total1 + pow(total2,2) + pow(total2,4);
}
```

3.6.10.Styblinski-Tang fonksiyonu

$$f(x) = \frac{1}{2}\sum_{i=1}^{d}(x_i^4 - 16x_i^2 + 5x_i) \qquad (3.14)$$

Styblinski-Tang fonksiyonu sürekli, diferensiyellenebilir, ölçeklenebilir olmayan, multimodal(birçok yerel minimum ancak bir genel minimum içeren), ayrılabilir olmayan bir kıyas fonksiyonudur. Styblinski-Tang fonksiyonunun test bölgesi genellikle $-5<x_i<5$; i=1,…,n sınırları arasındadır. Küresel minimum; i=1,…,n ve x_i=-

2.903534 için f(x)= -39.16599d noktasındadır. Şekil 3.37'de Styblinski-Tang fonksiyonu'nun grafiği görülmektedir.



Şekil 3.37. Styblinski-Tang fonksiyonu'nun grafiği

Seri Matlab Kodu(StyblinskiTang.m):

```
function [y] = StyblinskiTang(x)
d = length(x);
total = 0;
for i = 1:d
        total = total + x(i)^4 - 16*x(i)^2 + 5*x(i);
end
y = total/2;
end
```

Üretilen Ağaçlar için kullanılan Paralel C Kodu(StyblinskiTang.cu):

```
__global__ void Obj(double * objt, double * nt,int N,int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double total = 0.00;
   for (int j = 0; j < D; j++){
   double bir=nt[i+(j*N)];
   total = total + pow(bir,4) - 16*bir*bir + 5*bir;
}
objt[i]=objt[i]+total/2;}
```

Üretilen Tohumlar için kullanılan Paralel C Kodu(StyblinskiTangs.cu):

```
__global__ void Objs(double * objs, double * seeds,int N,int D)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   double total = 0.00;
   for (int j = 0; j < D; j++){
   double bir=seeds[i+(i/N)*(D-1)*N+(j*N)];
   total = total + pow(bir,4) - 16*bir*bir + 5*bir;
}
objs[i]=objs[i]+total/2;
}
```

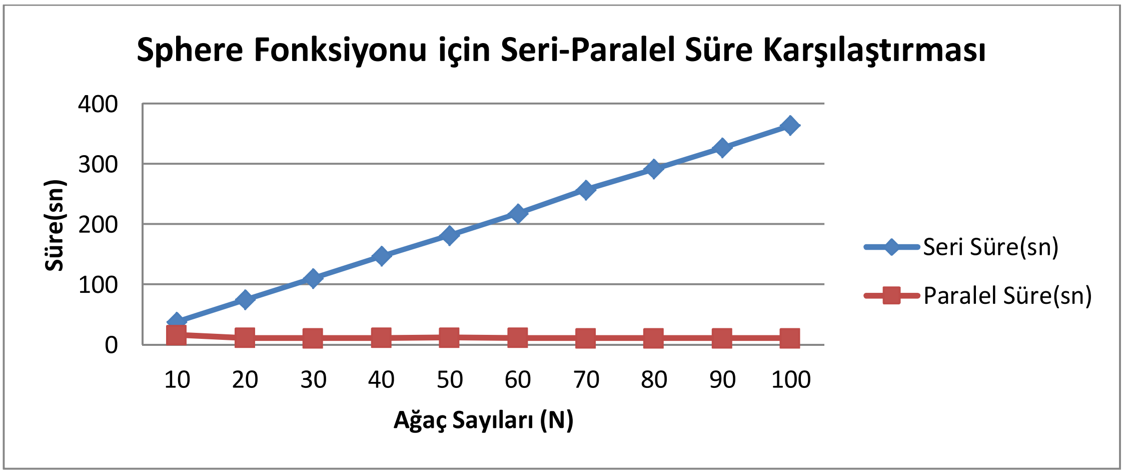
4. ARAŞTIRMA SONUÇLARI VE TARTIŞMA

Uygulama MATLAB R2014a ve Paralel Hesaplama Araç Kutusu(Parallel Computing Toolbox) kullanılarak geliştirilmiş, 2 adet Intel(R) Xeon(R) E5-2670 2.60 GHz işlemci(16x2=32 çekirdekli), 12 GB bellek ve 3072 CUDA çekirdeği olan GeForce GTX TITAN X ekran kartına sahip bir bilgisayarda çalıştırılmıştır.

Ağaç-Tohum Algoritması seri ve paralel olarak sıralı bir biçimde(önce paralel sonra seri) 10 farklı kıyas fonksiyonu(Sphere, Ackley, Griewank, Rastrigin, Rosenbrock, Schwefel, Styblinski Tang, Sum of Different Powers, Sum Squares, Zakharov) için literatürdeki çalışmalarda genel olarak tercih edildiği gibi 30'ar kere D=10, ST=0.1 sabit, ağaç sayıları N=10,20,30,40,50,60,70,80,90,100, yine sırasıyla tohum sayıları Kıran (2015)'ın önerdiği minimum yüzde 10, maksimum yüzde 25 göz önüna alınarak ağaç sayısının yüzde 20'si olan NS=2,4,6,8,10,12,14,16,18,20 alınarak çalıştırılmış ve aşağıdaki sonuçlar elde edilmiştir.

4.1. Sphere Fonksiyonu İçin Elde Edilen Sonuçlar

Küre(Sphere) fonksiyonu için elde edilen sonuçlar(Çizelge 4.3) incelendiğinde ağaç sayısı 10 alındığı durumda paralel çözümün 2,38 kat daha hızlı olduğu görülmektedir. Yine ağaç sayısının 100 alındığı durumda paralel çözüm **34,39 kat daha hızlı** çalışarak ağaç sayısı arttıkça paralel çözümün çok daha hızlı olduğunu ortaya koymuştur. Paralel sonuçlar (Çizelge 4.2) incelendiğinde ağaç sayısı 10 iken çözüm süresi yaklaşık 16 saniye iken, ağaç sayısı 100 olduğu durumda çözüm süresi yaklaşık 11 saniye olarak bulunmuştur. Bu doğrultuda paralel çözümün belirli bir eşiğe kadar ağaç sayısının artmasından olumsuz yönde etkilenmediğini söyleyebiliriz. Seri sonuçlar(Çizelge 4.1) incelendiğinde ağaç sayısına bağlı olarak programın süresinin katlamalı olarak arttığı görülmektedir. Şekil 4.1'de Sphere fonksiyonu için seri-paralel süre karşılaştırması görülmektedir.



Şekil 4.1. Sphere fonksiyonu için seri-paralel süre karşılaştırması

Çizelge 4.1. Seri uygulamada Sphere fonksiyonu için elde edilen sonuçlar

N	Seri Ortalama Sonuç	Seri Sonuç Std. Spm.	Seri Süre(sn)	Seri Süre Std. Spm.
10	0	0	38,23226575	0,276958695
20	0	0	74,8414415	1,016005919
30	0	0	110,4928888	0,157457821
40	0	0	147,2953196	1,564131504
50	0	0	181,8856821	0,40523393
60	0	0	217,8399961	0,495389274
70	1,9611E-303	0	257,0881699	11,32833916
80	2,2571E-284	0	291,9038706	5,654059889
90	8,5883E-268	0	326,8070661	3,463721936
100	4,8054E-253	0	364,1060309	4,17383416

Çizelge 4.2. Paralel uygulamada Sphere fonksiyonu için elde edilen sonuçlar

N	Paralel Ortalama Sonuç	Paralel Sonuç Std. Spm.	Paralel Süre(sn)	Paralel Süre Std. Spm.
10	0	0	16,0321054	1,564223133
20	0	0	10,9254034	0,561572011
30	0	0	10,69297234	0,157457821
40	0	0	11,1871108	0,159377178
50	2,589E-292	0	11,42834872	0,174971102
60	3,9242E-261	0	11,35301047	0,107097195
70	2,3074E-246	0	10,80896656	0,145182042
80	1,8799E-238	0	10,72639069	0,127443916
90	1,9228E-216	0	10,58879376	0,099435036
100	4,6284E-208	0	10,58634605	0,087855766

Çizelge 4.3. Sphere fonksiyonu için sürelerin karşılaştırılması ve hızlanma değerleri

N	Seri Süre(sn)	Paralel Süre(sn)	Hızlanma
10	38,23226575	16,0321054	2,38
20	74,8414415	10,9254034	6,85
30	110,4928888	10,69297234	10,33
40	147,2953196	11,1871108	13,17
50	181,8856821	11,42834872	15,92
60	217,8399961	11,35301047	19,19
70	257,0881699	10,80896656	23,78
80	291,9038706	10,72639069	27,21
90	326,8070661	10,58879376	30,86
100	364,1060309	10,58634605	34,39

4.2. Ackley Fonksiyonu İçin Elde Edilen Sonuçlar

Ackley fonksiyonu için elde edilen sonuçlar(Çizelge 4.6) incelendiğinde ağaç sayısı 10 alındığı durumda paralel çözümün 3,14 kat daha hızlı olduğu görülmektedir. Yine ağaç sayısının 100 alındığı durumda paralel çözüm **123,16 kat daha hızlı** çalışarak ağaç sayısı arttıkça paralel çözümün çok daha hızlı olduğunu ortaya koymuştur. Paralel sonuçlar (Çizelge 4.5) incelendiğinde ağaç sayısı 10 iken çözüm süresi yaklaşık 13 saniye iken, ağaç sayısı 100 olduğu durumda çözüm süresi yaklaşık 3 saniye olarak bulunmuştur. Bu doğrultuda paralel çözümün belirli bir eşiğe kadar ağaç sayısının artmasından olumsuz yönde etkilenmediğini söyleyebiliriz. Seri sonuçlar(Çizelge 4.4) incelendiğinde ağaç sayısına bağlı olarak programın süresinin katlamalı olarak arttığı görülmektedir. Şekil 4.2'de Ackley fonksiyonu için seri-paralel süre karşılaştırması görülmektedir.

Çizelge 4.4. Seri uygulamada Ackley fonksiyonu için elde edilen sonuçlar

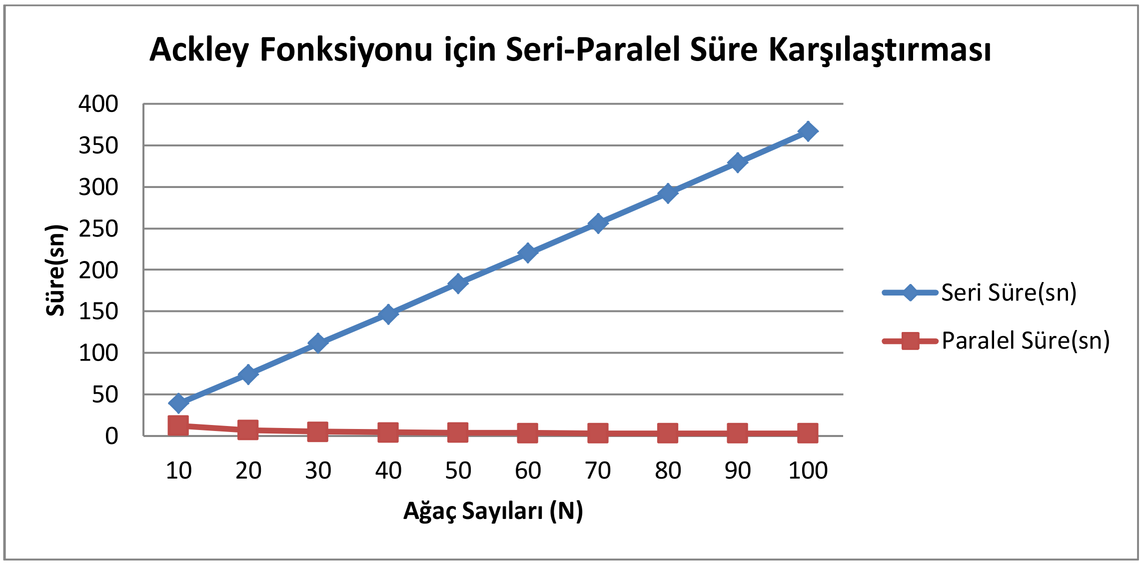
N	Seri Ortalama Sonuç	Seri Sonuç Std. Spm.	Seri Süre(sn)	Seri Süre Std. Spm.
10	4.20404E-15	9.01352E-16	39.22066396	2.939027688
20	4,32247E-15	6,48634E-16	74,38028578	0,1987056
30	3,61193E-15	1,52832E-15	111,6151469	4,362045207
40	3,9968E-15	1,34665E-15	146,924426	0,394820009
50	3,52311E-15	1,22834E-15	183,7623024	1,598663698
60	3,01981E-15	1,77022E-15	220,4304348	2,466880206
70	3,01981E-15	1,77022E-15	256,3795001	0,606815162
80	2,66454E-15	1,80672E-15	292,6929168	0,579886668
90	2,90138E-15	1,79059E-15	329,5146183	0,640534205
100	2,66454E-15	1,80672E-15	366,925874	0,543615483

Çizelge 4.5. Paralel uygulamada Ackley fonksiyonu için elde edilen sonuçlar

N	Paralel Ort. Sonuç	Paralel Sonuç Std. Spm.	Paralel Süre(sn)	Paralel Süre Std. Spm.
10	0,366800457	2,009048846	12,50005658	0,378418475
20	0,311761545	1,707588306	6,753480699	0,082412229
30	3,61193E-15	9,32988E-16	4,926522216	0,043971549
40	3,84877E-15	1,22834E-15	4,154538815	0,055105792
50	3,61193E-15	1,52832E-15	3,668753794	0,165744025
60	3,75996E-15	9,01352E-16	3,507599014	0,071296799
70	3,01981E-15	1,7413E-15	3,190874911	0,040886361
80	3,16784E-15	1,52832E-15	3,082820042	0,097523004
90	3,16784E-15	1,52832E-15	3,016109054	0,059287
100	2,45729E-15	1,79059E-15	2,979260521	0,060271156

Çizelge 4.6. Ackley fonksiyonu için sürelerin karşılaştırılması ve hızlanma değerleri

N	Seri Süre(sn)	Paralel Süre(sn)	Hızlanma
10	39,22066396	12,50005658	3,14
20	74,38028578	6,753480699	11,01
30	111,6151469	4,926522216	22,66
40	146,924426	4,154538815	35,36
50	183,7623024	3,668753794	50,09
60	220,4304348	3,507599014	62,84
70	256,3795001	3,190874911	80,35
80	292,6929168	3,082820042	94,94
90	329,5146183	3,016109054	109,25
100	366,925874	2,979260521	123,16



Şekil 4.2. Ackley fonksiyonu için seri-paralel süre karşılaştırması

4.3. Griewank Fonksiyonu İçin Elde Edilen Sonuçlar

Griewank fonksiyonu için elde edilen sonuçlar(Çizelge 4.9) incelendiğinde ağaç sayısı 10 alındığı durumda paralel çözümün 2,81 kat daha hızlı olduğu görülmektedir. Yine ağaç sayısının 100 alındığı durumda paralel çözüm **149,47 kat daha hızlı** çalışarak ağaç sayısı arttıkça paralel çözümün çok daha hızlı olduğunu ortaya koymuştur. Paralel sonuçlar (Çizelge 4.8) incelendiğinde ağaç sayısı 10 iken çözüm süresi yaklaşık 14 saniye iken, ağaç sayısı 100 olduğu durumda çözüm süresi yaklaşık 3 saniye olarak bulunmuştur. Bu doğrultuda paralel çözümün belirli bir eşiğe kadar ağaç sayısının artmasından olumsuz yönde etkilenmediğini söyleyebiliriz. Seri sonuçlar(Çizelge 4.7) incelendiğinde ağaç sayısına bağlı olarak programın süresinin katlamalı olarak arttığı görülmektedir. Şekil 4.3'te Griewank fonksiyonu için seri-paralel süre karşılaştırması görülmektedir.

Çizelge 4.7. Seri uygulamada Griewank fonksiyonu için elde edilen sonuçlar

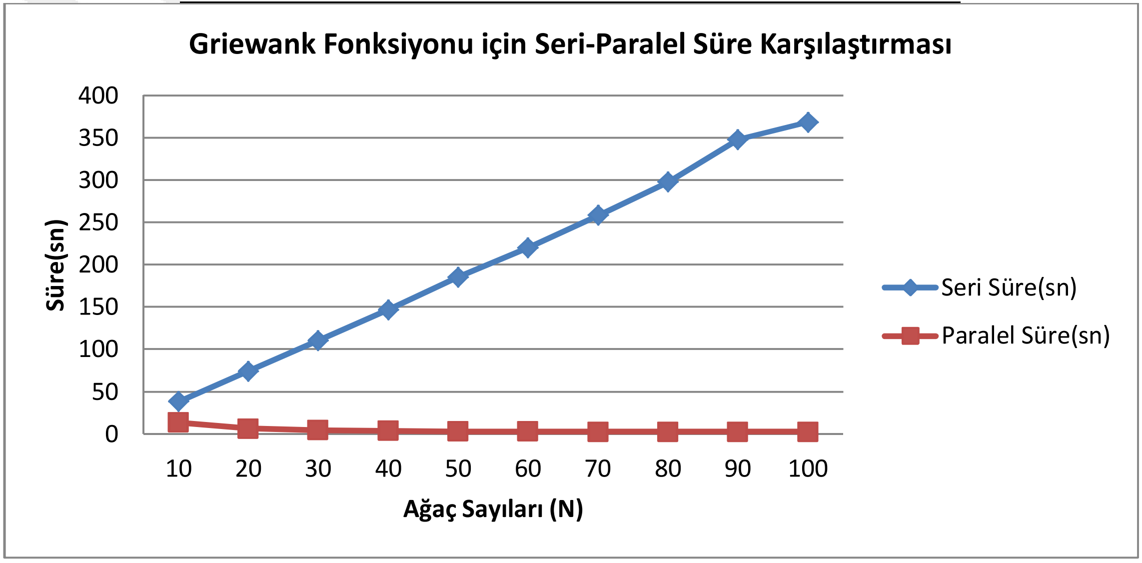
N	Seri Ortalama Sonuç	Seri Sonuç Std. Spm.	Seri Süre(sn)	Seri Süre Std. Spm.
10	0,022531545	0,012770933	38,63533937	2,096308401
20	0,007229607	0,008434079	74,59891746	0,231688688
30	0,006407225	0,010054097	110,7737428	0,286942987
40	0,003391533	0,00591853	146,9019468	0,36682313
50	0,002878846	0,008140742	185,6205897	4,19254284
60	0,004772121	0,013269732	220,3974623	1,985477213
70	0,004041558	0,007943586	258,4985922	7,041232067
80	0,002917476	0,008442986	297,9101156	20,29742458
90	0,003012924	0,00529594	348,3224636	36,84283471
100	0,002437377	0,007720097	368,6155271	1,669190713

Çizelge 4.8. Paralel uygulamada Griewank fonksiyonu için elde edilen sonuçlar

N	Paralel Ortalama Sonuç	Paralel Sonuç Std. Spm.	Paralel Süre(sn)	Paralel Süre Std. Spm.
10	0,02502022	0,03777243	13,76146884	1,284113391
20	0,032895817	0,056685842	6,513510395	0,078547444
30	0,029456813	0,034075603	4,573183592	0,083500962
40	0,025846819	0,052307762	3,746323545	0,117697647
50	0,020532609	0,035318519	3,180342085	0,114393648
60	0,04817493	0,04776617	2,84099703	0,155895584
70	0	0	2,765166703	0,042511155
80	0	0	2,643292819	0,073124666
90	0	0	2,512782338	0,088445306
100	2,07242E-16	1,11432E-15	2,466155922	0,103371943

Çizelge 4.9. Griewank fonksiyonu için sürelerin karşılaştırılması ve hızlanma değerleri

N	Seri Süre(sn)	Paralel Süre(sn)	Hızlanma
10	38,63533937	13,76146884	2,81
20	74,59891746	6,513510395	11,45
30	110,7737428	4,573183592	24,22
40	146,9019468	3,746323545	39,21
50	185,6205897	3,180342085	58,36
60	220,3974623	2,84099703	77,58
70	258,4985922	2,765166703	93,48
80	297,9101156	2,643292819	112,70
90	348,3224636	2,512782338	138,62
100	368,6155271	2,466155922	149,47



Şekil 4.3. Griewank fonksiyonu için seri-paralel süre karşılaştırması

4.4. Rastrigin Fonksiyonu İçin Elde Edilen Sonuçlar

Rastrigin fonksiyonu için elde edilen sonuçlar(Çizelge 4.12) incelendiğinde ağaç sayısı 10 alındığı durumda paralel çözümün 3,14 kat daha hızlı olduğu görülmektedir. Yine ağaç sayısının 100 alındığı durumda paralel çözüm **148,02 kat daha hızlı** çalışarak ağaç sayısı arttıkça paralel çözümün çok daha hızlı olduğunu ortaya koymuştur. Paralel sonuçlar (Çizelge 4.11) incelendiğinde ağaç sayısı 10 iken çözüm süresi yaklaşık 12 saniye iken, ağaç sayısı 100 olduğu durumda çözüm süresi yaklaşık 3 saniye olarak bulunmuştur. Bu doğrultuda paralel çözümün belirli bir eşiğe kadar ağaç sayısının artmasından olumsuz yönde etkilenmediğini söyleyebiliriz. Seri sonuçlar(Çizelge 4.10) incelendiğinde ağaç sayısına bağlı olarak programın süresinin katlamalı olarak arttığı görülmektedir. Şekil 4.4'de Rastrigin fonksiyonu için seri-paralel süre karşılaştırması görülmektedir.

Çizelge 4.10. Seri uygulamada Rastrigin fonksiyonu için elde edilen sonuçlar

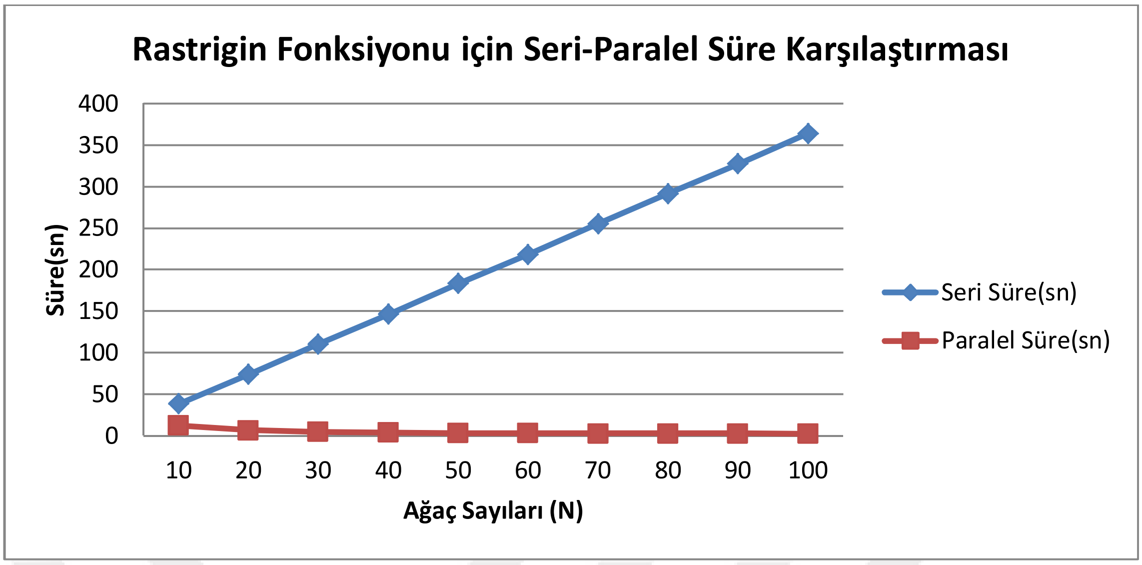
N	Seri Ortalama Sonuç	Seri Sonuç Std. Spm.	Seri Süre(sn)	Seri Süre Std. Spm.
10	1,724595699	0,975323576	38,45207357	0,152570965
20	0,198991811	0,404787255	74,18440942	0,115936696
30	0,16582651	0,377138257	110,6099671	0,141388963
40	0,033165302	0,18165384	146,4812711	0,322366647
50	0	0	183,3511055	0,30594467
60	0	0	218,6772724	0,864105411
70	0	0	255,5540963	1,114255657
80	0	0	291,7249214	1,577413336
90	0	0	327,6867569	0,493587678
100	0	0	364,4600577	0,666931909

Çizelge 4.11. Paralel uygulamada Rastrigin fonksiyonu için elde edilen sonuçlar

N	Paralel Ort. Sonuç	Paralel Sonuç Std. Spm.	Paralel Süre(sn)	Paralel Süre Std. Spm.
10	1,052424046	1,958259903	12,25771842	0,058068368
20	1,154727639	2,58988704	6,494478963	0,06809164
30	0,710277493	1,680431915	4,640404838	0,112256036
40	0,75192839	1,584202032	3,759595713	0,139834142
50	0,788920281	1,569461458	3,224064712	0,174946941
60	0,754217952	1,733508233	2,95221228	0,181533469
70	1,18135E-07	6,47054E-07	2,867018322	0,100143102
80	0,035451127	0,194167025	2,64661653	0,125610088
90	0,021566115	0,118119218	2,590742682	0,151841583
100	0,053819148	0,275757294	2,462298612	0,186850087

Çizelge 4.12. Rastrigin fonksiyonu için sürelerin karşılaştırılması ve hızlanma değerleri

N	Seri Süre(sn)	Paralel Süre(sn)	Hızlanma
10	38,45207357	12,25771842	3,14
20	74,18440942	6,494478963	11,42
30	110,6099671	4,640404838	23,84
40	146,4812711	3,759595713	38,96
50	183,3511055	3,224064712	56,87
60	218,6772724	2,95221228	74,07
70	255,5540963	2,867018322	89,14
80	291,7249214	2,64661653	110,23
90	327,6867569	2,590742682	126,48
100	364,4600577	2,462298612	148,02



Şekil 4.4. Rastrigin fonksiyonu için seri-paralel süre karşılaştırması

4.5. Rosenbrock Fonksiyonu İçin Elde Edilen Sonuçlar

Rosenbrock fonksiyonu için elde edilen sonuçlar(Çizelge 4.15) incelendiğinde ağaç sayısı 10 alındığı durumda paralel çözümün 3,15 kat daha hızlı olduğu görülmektedir. Yine ağaç sayısının 100 alındığı durumda paralel çözüm **149,59 kat daha hızlı** çalışarak ağaç sayısı arttıkça paralel çözümün çok daha hızlı olduğunu ortaya koymuştur. Paralel sonuçlar (Çizelge 4.14) incelendiğinde ağaç sayısı 10 iken çözüm süresi yaklaşık 12 saniye iken, ağaç sayısı 100 olduğu durumda çözüm süresi yaklaşık 3 saniye olarak bulunmuştur. Bu doğrultuda paralel çözümün belirli bir eşiğe kadar ağaç sayısının artmasından olumsuz yönde etkilenmediğini söyleyebiliriz. Seri sonuçlar(Çizelge 4.13) incelendiğinde ağaç sayısına bağlı olarak programın süresinin katlamalı olarak arttığı görülmektedir. Şekil 4.5'de Rosenbrock fonksiyonu için seri-paralel süre karşılaştırması görülmektedir.

Çizelge 4.13. Seri uygulamada Rosenbrock fonksiyonu için elde edilen sonuçlar

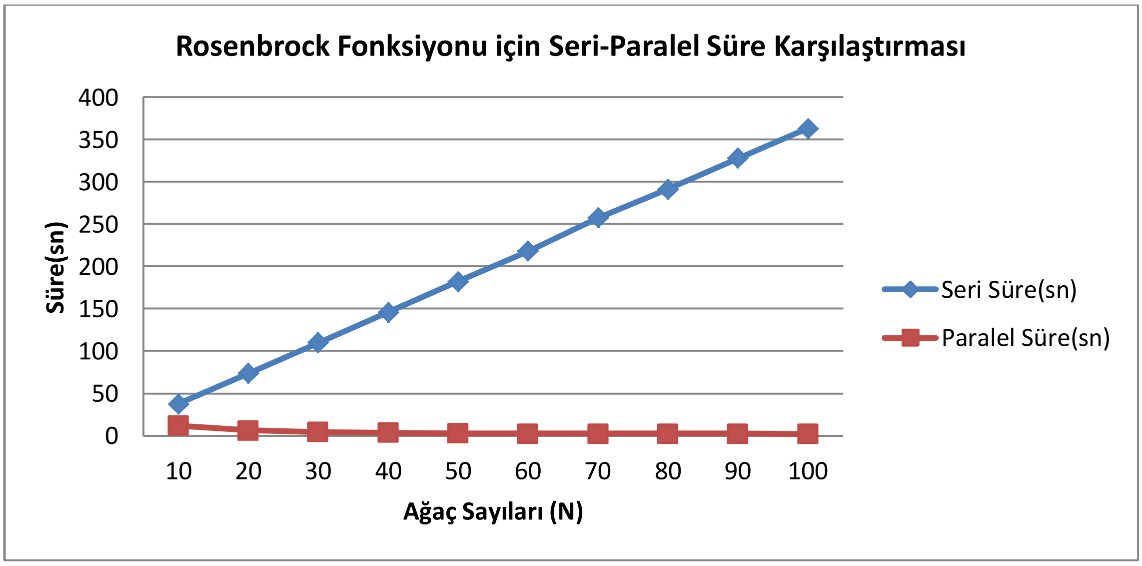
N	Seri Ortalama Sonuç	Seri Sonuç Std. Spm.	Seri Süre(sn)	Seri Süre Std. Spm.
10	0,559461155	1,379139386	38,12319902	0,077964998
20	0,028471731	0,051103391	73,95463371	0,191666503
30	0,040019467	0,069942274	110,2707814	0,169240017
40	0,025123774	0,051583167	146,144689	0,340623781
50	0,025888247	0,031216783	182,5466038	0,36564376
60	0,018779547	0,012345976	218,6060767	0,61505494
70	0,023370556	0,015299196	257,9680336	8,513452095
80	0,030280743	0,05548489	291,4668318	0,417022153
90	0,027958378	0,019509785	328,3470957	2,105886531
100	0,029744681	0,0171622	363,382942	0,545651037

Çizelge 4.14. Paralel uygulamada Rosenbrock fonksiyonu için elde edilen sonuçlar

N	Paralel Ort. Sonuç	Paralel Sonuç Std. Spm.	Paralel Süre(sn)	Paralel Süre Std. Spm.
10	3,704494307	2,785633027	12,09487829	0,039561483
20	4,31491512	3,811114733	6,401437052	0,047575526
30	3,555546287	1,405334342	4,496686346	0,048732548
40	3,908235356	1,431587529	3,584983353	0,040895288
50	3,632651097	2,185986709	3,047194583	0,060549583
60	3,366596777	2,153140023	2,794027244	0,062539955
70	2,69026667	1,410947572	2,747032724	0,077585379
80	2,452534414	1,549220712	2,591055648	0,151153252
90	2,641474617	1,459660024	2,49159068	0,122215872
100	2,657859749	1,264245889	2,429193252	0,142778451

Çizelge 4.15. Rosenbrock fonksiyonu için sürelerin karşılaştırılması ve hızlanma değerleri

N	Seri Süre(sn)	Paralel Süre(sn)	Hızlanma
10	38,12319902	12,09487829	3,15
20	73,95463371	6,401437052	11,55
30	110,2707814	4,496686346	24,52
40	146,144689	3,584983353	40,77
50	182,5466038	3,047194583	59,91
60	218,6060767	2,794027244	78,24
70	257,9680336	2,747032724	93,91
80	291,4668318	2,591055648	112,49
90	328,3470957	2,49159068	131,78
100	363,382942	2,429193252	149,59



Şekil 4.5. Rosenbrock fonksiyonu için seri-paralel süre karşılaştırması

4.6. Schwefel Fonksiyonu İçin Elde Edilen Sonuçlar

Schwefel fonksiyonu için elde edilen sonuçlar(Çizelge 4.18) incelendiğinde ağaç sayısı 10 alındığı durumda paralel çözümün 2,96 kat daha hızlı olduğu görülmektedir. Yine ağaç sayısının 100 alındığı durumda paralel çözüm **184,65 kat daha hızlı** çalışarak ağaç sayısı arttıkça paralel çözümün çok daha hızlı olduğunu ortaya koymuştur. Paralel sonuçlar (Çizelge 4.17) incelendiğinde ağaç sayısı 10 iken çözüm süresi yaklaşık 13 saniye iken, ağaç sayısı 100 olduğu durumda çözüm süresi yaklaşık 2 saniye olarak bulunmuştur. Bu doğrultuda paralel çözümün belirli bir eşiğe kadar ağaç sayısının artmasından olumsuz yönde etkilenmediğini söyleyebiliriz. Seri sonuçlar(Çizelge 4.16) incelendiğinde ağaç sayısına bağlı olarak programın süresinin katlamalı olarak arttığı görülmektedir.

Çizelge 4.16. Seri uygulamada Schwefel fonksiyonu için elde edilen sonuçlar

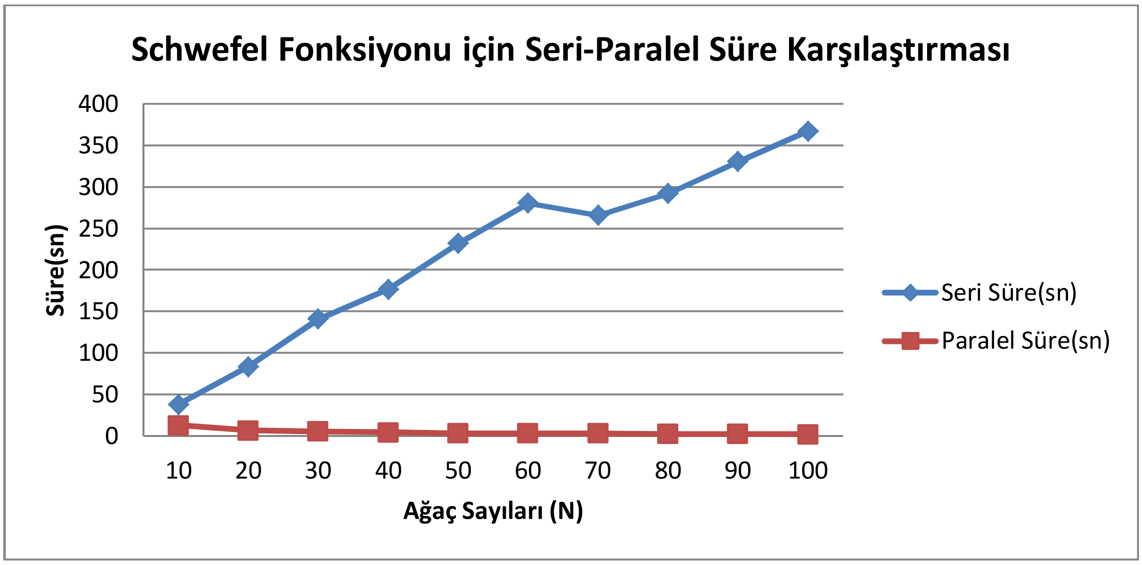
N	Seri Ortalama Sonuç	Seri Sonuç Std. Spm.	Seri Süre(sn)	Seri Süre Std. Spm.
10	165,8137957	134,1419956	38,15638327	0,141299373
20	27,63573869	50,95016605	83,61479086	10,18887047
30	3,948071763	21,62378251	141,1841942	7,038242505
40	0,000127276	0	176,771514	19,46595301
50	0,000127276	0	231,9142198	16,78729392
60	0,000127276	0	280,7581178	12,22054662
70	0,000127276	0	265,993142	21,20824994
80	0,000127276	0	292,4099343	0,476973693
90	0,000127276	0	330,7436664	1,441881015
100	0,000127276	0	367,5837712	0,961679455

Çizelge 4.17. Paralel uygulamada Schwefel fonksiyonu için elde edilen sonuçlar

N	Paralel Ort. Sonuç	Paralel Sonuç Std. Spm.	Paralel Süre(sn)	Paralel Süre Std. Spm.
10	794,2392807	422,5473024	12,89130097	1,311896797
20	739,1421741	379,3010978	6,418941038	0,031327324
30	786,1206278	377,3590242	5,403610881	0,074476689
40	603,3294846	410,1335925	4,293218825	0,078617018
50	628,078318	381,4766412	2,937048532	0,078271134
60	586,846404	353,4316761	3,222772776	0,097509202
70	452,1222117	233,160913	2,939231864	0,080848178
80	387,6242388	219,6644613	2,213762582	0,10732058
90	220,1357436	190,7875643	2,121904469	0,098295825
100	266,8443514	197,7964557	1,990755676	0,093069008

Çizelge 4.18. Schwefel fonksiyonu için sürelerin karşılaştırılması ve hızlanma değerleri

N	Seri Süre(sn)	Paralel Süre(sn)	Hızlanma
10	38,15638327	12,89130097	2,96
20	83,61479086	6,418941038	13,03
30	141,1841942	5,403610881	26,13
40	176,771514	4,293218825	41,17
50	231,9142198	2,937048532	78,96
60	280,7581178	3,222772776	87,12
70	265,993142	2,939231864	90,50
80	292,4099343	2,213762582	132,09
90	330,7436664	2,121904469	155,87
100	367,5837712	1,990755676	184,65



Şekil 4.6. Schwefel fonksiyonu için seri-paralel süre karşılaştırması

4.7. Styblinski Tang Fonksiyonu İçin Elde Edilen Sonuçlar

Styblinski Tang fonksiyonu için elde edilen sonuçlar(Çizelge 4.21) incelendiğinde ağaç sayısı 10 alındığı durumda paralel çözümün 3,19 kat daha hızlı olduğu görülmektedir. Yine ağaç sayısının 100 alındığı durumda paralel çözüm **159,45 kat daha hızlı** çalışarak ağaç sayısı arttıkça paralel çözümün çok daha hızlı olduğunu ortaya koymuştur. Paralel sonuçlar (Çizelge 4.20) incelendiğinde ağaç sayısı 10 iken çözüm süresi yaklaşık 12 saniye iken, ağaç sayısı 100 olduğu durumda çözüm süresi yaklaşık 2 saniye olarak bulunmuştur. Bu doğrultuda paralel çözümün belirli bir eşiğe kadar ağaç sayısının artmasından olumsuz yönde etkilenmediğini söyleyebiliriz. Seri sonuçlar(Çizelge 4.19) incelendiğinde ağaç sayısına bağlı olarak programın süresinin katlamalı olarak arttığı görülmektedir. Şekil 4.7'de Styblinski Tang fonksiyonu için seri-paralel süre karşılaştırması görülmektedir.

Çizelge 4.19. Seri uygulamada Styblinski Tang fonksiyonu için elde edilen sonuçlar

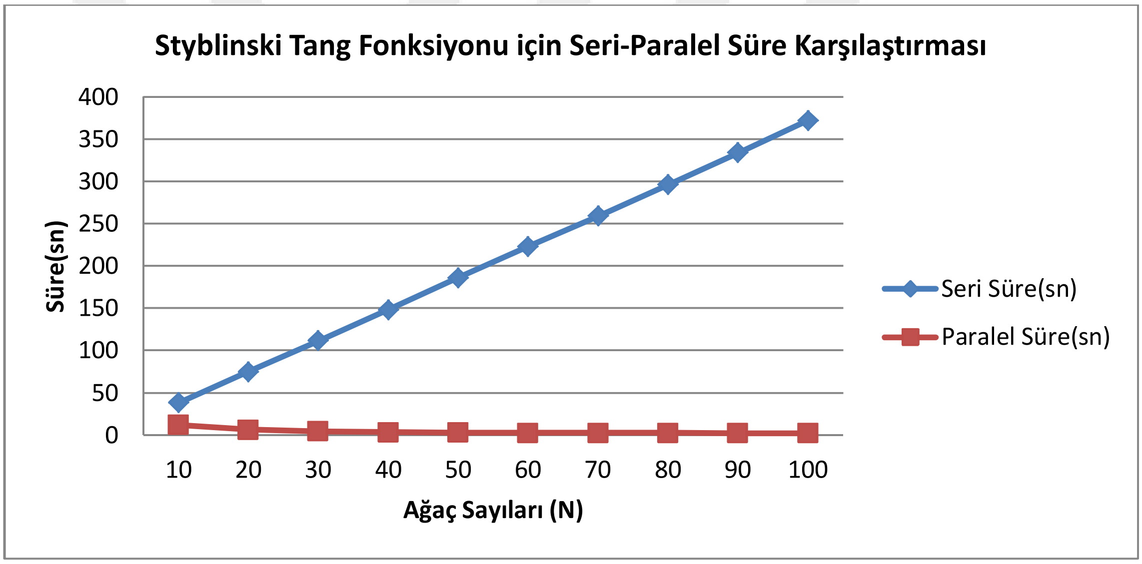
N	Seri Ortalama Sonuç	Seri Sonuç Std. Spm.	Seri Süre(sn)	Seri Süre Std. Spm.
10	-385,5357455	9,59756776	38,38153561	0,065796588
20	-390,2479851	4,313516961	75,37404909	0,158076917
30	-391,661657	1,67564E-13	111,9683031	0,268955645
40	-391,661657	1,12703E-13	148,4962733	0,217327268
50	-391,661657	1,1563E-13	186,2985318	0,522258392
60	-391,661657	1,1563E-13	223,5535536	1,336785591
70	-391,661657	1,18015E-13	259,2252265	0,640997296
80	-391,661657	1,1563E-13	296,5924365	0,401999192
90	-391,661657	1,18015E-13	334,2857333	1,582099004
100	-391,661657	1,18015E-13	372,3962661	5,902328695

Çizelge 4.20. Paralel uygulamada Styblinski Tang fonksiyonu için elde edilen sonuçlar

N	Paralel Ort. Sonuç	Paralel Sonuç Std. Spm.	Paralel Süre(sn)	Paralel Süre Std. Spm.
10	-389,1330679	6,754124311	12,04465015	0,030166374
20	-391,4387036	0,395569048	6,306356867	0,034811432
30	-391,4499962	0,300577151	4,455121918	0,042276092
40	-391,57092	0,201406398	3,562300408	0,045323026
50	-387,8756018	12,97345925	3,021138457	0,074306663
60	-390,7791985	3,82007741	2,752399838	0,082638983
70	-391,6614097	0,001243561	2,667520494	0,079898659
80	-391,6616119	0,000186737	2,545875257	0,072517969
90	-391,6615931	0,000305403	2,394039845	0,058847839
100	-391,6616548	6,80393E-06	2,335556156	0,056456394

Çizelge 4.21. Styblinski Tang fonksiyonu için sürelerin karşılaştırılması ve hızlanma değerleri

N	Seri Süre(sn)	Paralel Süre(sn)	Hızlanma
10	38,38153561	12,04465015	3,19
20	75,37404909	6,306356867	11,95
30	111,9683031	4,455121918	25,13
40	148,4962733	3,562300408	41,69
50	186,2985318	3,021138457	61,67
60	223,5535536	2,752399838	81,22
70	259,2252265	2,667520494	97,18
80	296,5924365	2,545875257	116,50
90	334,2857333	2,394039845	139,63
100	372,3962661	2,335556156	159,45



Şekil 4.7. Styblinski Tang fonksiyonu için seri-paralel süre karşılaştırması

4.8. Sum of Different Powers Fonksiyonu İçin Elde Edilen Sonuçlar

Sum of Different Powers fonksiyonu için elde edilen sonuçlar(Çizelge 4.24) incelendiğinde ağaç sayısı 10 alındığı durumda paralel çözümün 3,01 kat daha hızlı olduğu görülmektedir. Yine ağaç sayısının 100 alındığı durumda paralel çözüm **69,28 kat daha hızlı** çalışarak ağaç sayısı arttıkça paralel çözümün çok daha hızlı olduğunu ortaya koymuştur. Paralel sonuçlar (Çizelge 4.23) incelendiğinde ağaç sayısı 10 iken çözüm süresi yaklaşık 13 saniye iken, ağaç sayısı 100 olduğu durumda çözüm süresi yaklaşık 5 saniye olarak bulunmuştur. Bu doğrultuda paralel çözümün belirli bir eşiğe kadar ağaç sayısının artmasından olumsuz yönde etkilenmediğini söyleyebiliriz. Seri sonuçlar(Çizelge 4.22) incelendiğinde ağaç sayısına bağlı olarak programın süresinin katlamalı olarak arttığı görülmektedir. Şekil 4.8'de Sum of Different Powers fonksiyonu için seri-paralel süre karşılaştırması görülmektedir.

Çizelge 4.22. Seri uygulamada Sum of Different Powers fonksiyonu için elde edilen sonuçlar

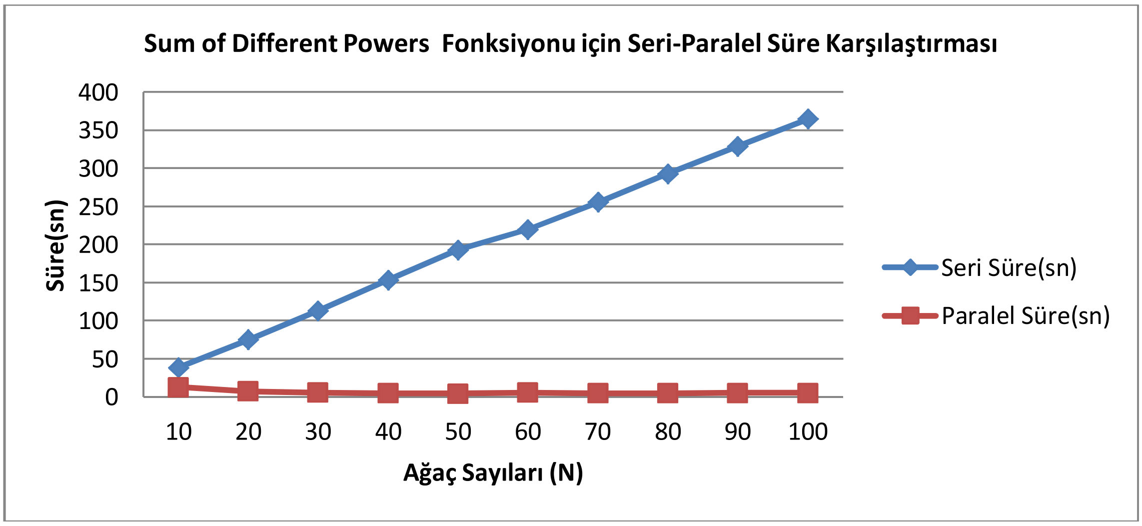
N	Seri Ortalama Sonuç	Seri Sonuç Std. Spm.	Seri Süre(sn)	Seri Süre Std. Spm.
10	0	0	38,69826895	0,139724193
20	0	0	75,49070365	0,257110567
30	0	0	113,7408357	1,570982153
40	0	0	153,7772693	12,49767838
50	0	0	193,3595023	16,76449669
60	0	0	220,2428	0,910100561
70	0	0	255,7998454	0,704562313
80	0	0	293,3903907	0,682725785
90	0	0	329,2438796	2,326991153
100	0	0	365,1776598	1,129428115

Çizelge 4.23. Paralel uygulamada Sum of Different Powers fonksiyonu için elde edilen sonuçlar

N	Paralel Ort. Sonuç	Paralel Sonuç Std. Spm.	Paralel Süre(sn)	Paralel Süre Std. Spm.
10	0	0	12,86954739	0,630777825
20	0	0	7,2639365	0,325785511
30	0	0	5,552112472	0,03035776
40	0	0	4,905457931	0,049211304
50	0	0	4,564265876	0,044988098
60	0	0	5,754298939	0,240037642
70	0	0	4,894203625	0,111994416
80	0	0	5,04833208	0,191591122
90	0	0	5,062206177	0,15967779
100	0	0	5,27091306	0,274919745

Çizelge 4.24. Sum of Different Powers fonksiyonu için sürelerin karşılaştırılması ve hızlanma değerleri

N	Seri Süre(sn)	Paralel Süre(sn)	Hızlanma
10	38,69826895	12,86954739	3,01
20	75,49070365	7,2639365	10,39
30	113,7408357	5,552112472	20,49
40	153,7772693	4,905457931	31,35
50	193,3595023	4,564265876	42,36
60	220,2428	5,754298939	38,27
70	255,7998454	4,894203625	52,27
80	293,3903907	5,04833208	58,12
90	329,2438796	5,062206177	65,04
100	365,1776598	5,27091306	69,28



Şekil 4.8. Sum of Different Powers fonksiyonu için seri-paralel süre karşılaştırması

4.9. Sum Squares Fonksiyonu İçin Elde Edilen Sonuçlar

Sum Squares fonksiyonu için elde edilen sonuçlar(Çizelge 4.27) incelendiğinde ağaç sayısı 10 alındığı durumda paralel çözümün 2,06 kat daha hızlı olduğu görülmektedir. Yine ağaç sayısının 100 alındığı durumda paralel çözüm **34,91 kat daha hızlı** çalışarak ağaç sayısı arttıkça paralel çözümün çok daha hızlı olduğunu ortaya koymuştur. Paralel sonuçlar (Çizelge 4.26) incelendiğinde ağaç sayısı 10 iken çözüm süresi yaklaşık 18 saniye iken, ağaç sayısı 100 olduğu durumda çözüm süresi yaklaşık 10 saniye olarak bulunmuştur. Bu doğrultuda paralel çözümün belirli bir eşiğe kadar ağaç sayısının artmasından olumsuz yönde etkilenmediğini söyleyebiliriz. Seri sonuçlar(Çizelge 4.25) incelendiğinde ağaç sayısına bağlı olarak programın süresinin katlamalı olarak arttığı görülmektedir. Şekil 4.9'da Sum Squares fonksiyonu için seri-paralel süre karşılaştırması görülmektedir.

Çizelge 4.25. Seri uygulamada Sum Squares fonksiyonu için elde edilen sonuçlar

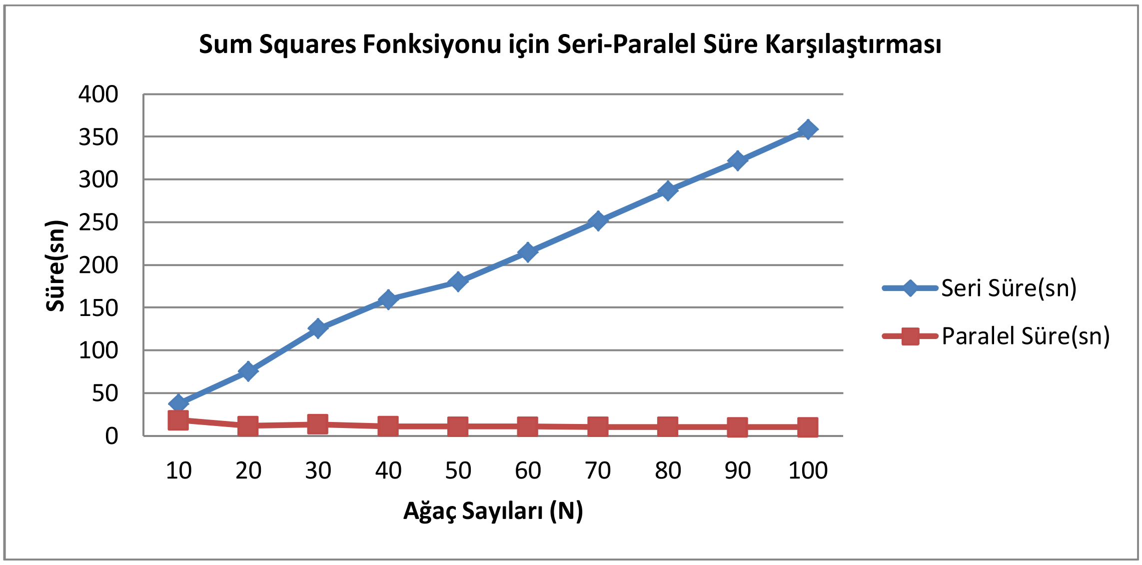
N	Seri Ortalama Sonuç	Seri Sonuç Std. Spm.	Seri Süre(sn)	Seri Süre Std. Spm.
10	0	0	37,69141144	0,212628492
20	0	0	75,53422423	4,38074998
30	0	0	125,750481	16,04237391
40	0	0	159,7815126	20,4621059
50	0	0	180,3762126	1,272806366
60	0	0	215,2964983	0,847455734
70	1,0281E-304	0	251,7571659	0,631457092
80	8,2621E-286	0	287,2449719	1,619540777
90	3,2627E-269	0	322,0684506	0,879325873
100	1,7387E-254	0	358,7588714	3,100941919

Çizelge 4.26. Paralel uygulamada Sum Squares fonksiyonu için elde edilen sonuçlar

N	Paralel Ort. Sonuç	Paralel Sonuç Std. Spm.	Paralel Süre(sn)	Paralel Süre Std. Spm.
10	0	0	18,29649175	1,583024477
20	0	0	11,52860136	0,141532613
30	0	0	13,33013339	0,785878116
40	0	0	11,10381842	0,109516113
50	1,5249E-273	0	10,80028608	0,099310966
60	1,7704E-252	0	10,78250082	0,138401144
70	3,1324E-239	0	10,3522873	0,086733066
80	1,2903E-215	0	10,34243232	0,155581098
90	2,484E-192	0	10,26854823	0,097598522
100	1,1654E-195	0	10,27655678	0,096155919

Çizelge 4.27. Sum Squares fonksiyonu için sürelerin karşılaştırılması ve hızlanma değerleri

N	Seri Süre(sn)	Paralel Süre(sn)	Hızlanma
10	37,69141144	18,29649175	2,06
20	75,53422423	11,52860136	6,55
30	125,750481	13,33013339	9,43
40	159,7815126	11,10381842	14,39
50	180,3762126	10,80028608	16,70
60	215,2964983	10,78250082	19,97
70	251,7571659	10,3522873	24,32
80	287,2449719	10,34243232	27,77
90	322,0684506	10,26854823	31,36
100	358,7588714	10,27655678	34,91



Şekil 4.9. Sum Squares fonksiyonu için seri-paralel süre karşılaştırması

4.10. Zakharov Fonksiyonu İçin Elde Edilen Sonuçlar

Sum Squares fonksiyonu için elde edilen sonuçlar(Çizelge 4.30) incelendiğinde ağaç sayısı 10 alındığı durumda paralel çözümün 2,45 kat daha hızlı olduğu görülmektedir. Yine ağaç sayısının 100 alındığı durumda paralel çözüm **46,56 kat daha hızlı** çalışarak ağaç sayısı arttıkça paralel çözümün çok daha hızlı olduğunu ortaya koymuştur. Paralel sonuçlar (Çizelge 4.29) incelendiğinde ağaç sayısı 10 iken çözüm süresi yaklaşık 15 saniye iken, ağaç sayısı 100 olduğu durumda çözüm süresi yaklaşık 8 saniye olarak bulunmuştur. Bu doğrultuda paralel çözümün belirli bir eşiğe kadar ağaç sayısının artmasından olumsuz yönde etkilenmediğini söyleyebiliriz. Seri sonuçlar(Çizelge 4.28) incelendiğinde ağaç sayısına bağlı olarak programın süresinin katlamalı olarak arttığı görülmektedir. Şekil 4.10'da Zakharov fonksiyonu için seri-paralel süre karşılaştırması görülmektedir.

Çizelge 4.28. Seri uygulamada Zakharov fonksiyonu için elde edilen sonuçlar

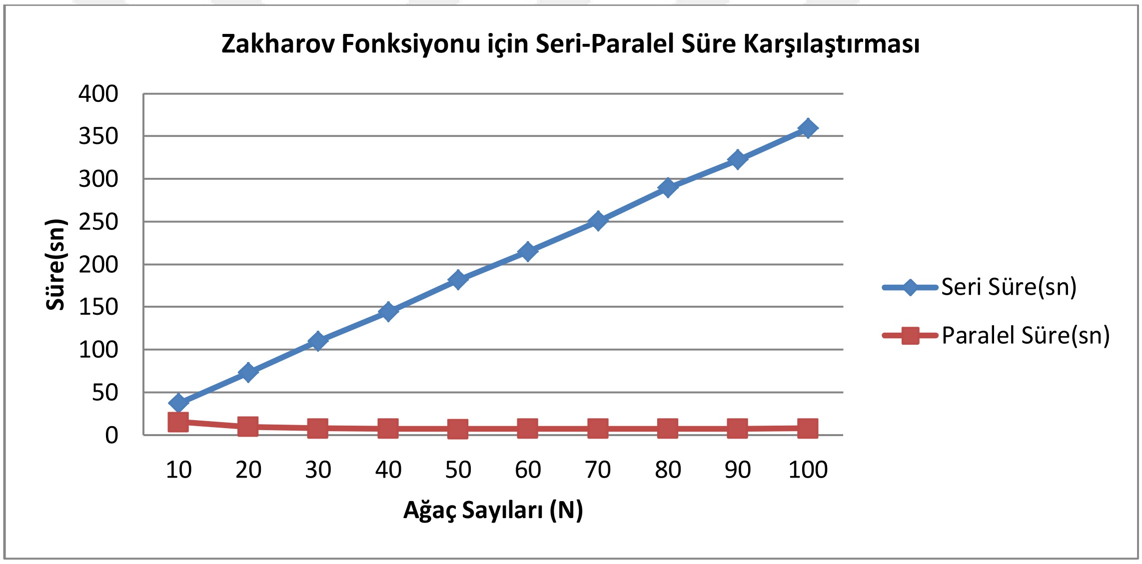
N	Seri Ortalama Sonuç	Seri Sonuç Std. Spm.	Seri Süre(sn)	Seri Süre Std. Spm.
10	1,302E-139	6,0489E-139	37,47606971	0,140320588
20	1,3369E-137	4,9575E-137	73,35233171	0,464713119
30	1,0548E-133	5,4697E-133	110,3078827	1,376915383
40	4,7019E-131	2,145E-130	144,7163356	1,451165055
50	5,2238E-128	2,1785E-127	182,1823897	1,801596764
60	5,8625E-126	8,8826E-126	215,3710555	0,565101632
70	2,0205E-122	5,5311E-122	250,9972584	0,758369574
80	4,1396E-120	5,7264E-120	289,8942069	1,987011498
90	1,5317E-117	2,8089E-117	322,9637168	0,986780269
100	5,9936E-115	6,823E-115	359,3547668	2,422894523

Çizelge 4.29. Paralel uygulamada Zakharov fonksiyonu için elde edilen sonuçlar

N	Paralel Ort. Sonuç	Paralel Sonuç Std. Spm.	Paralel Süre(sn)	Paralel Süre Std. Spm.
10	9,3383E-142	4,9915E-141	15,30602973	0,088696203
20	1,8528E-125	1,0148E-124	9,569470166	0,105693816
30	7,1718E-125	3,0314E-124	8,004739099	0,496892227
40	9,4783E-119	5,1913E-118	7,314301765	0,116848491
50	5,2238E-128	4,9316E-113	7,203881164	0,134516664
60	1,1147E-109	5,4659E-109	7,312128923	0,138223421
70	2,0703E-99	1,08429E-98	7,336652408	0,169933552
80	1,32762E-96	5,51372E-96	7,428516541	0,164383302
90	9,35886E-93	5,12606E-92	7,611244009	0,153332247
100	4,18294E-88	1,6845E-87	7,718188363	0,180356233

Çizelge 4.30. Zakharov fonksiyonu için sürelerin karşılaştırılması ve hızlanma değerleri

N	Seri Süre(sn)	Paralel Süre(sn)	Hızlanma
10	37,47606971	15,30602973	2,45
20	73,35233171	9,569470166	7,67
30	110,3078827	8,004739099	13,78
40	144,7163356	7,314301765	19,79
50	182,1823897	7,203881164	25,29
60	215,3710555	7,312128923	29,45
70	250,9972584	7,336652408	34,21
80	289,8942069	7,428516541	39,02
90	322,9637168	7,611244009	42,43
100	359,3547668	7,718188363	46,56



Şekil 4.10. Zakharov fonksiyonu için seri-paralel süre karşılaştırması

4.11.Tüm Sonuçların Genel Değerlendirmesi

Çizelge 4.31’de de görüleceği üzere ağaç sayısı 10 alındığı durumda en az 2,06 en çok 3,19 kat hızlanma sağlanmıştır. Yine ağaç sayısı 100 alındığı zaman en az 34,39 en çok 184,65 kat hızlanma sağlanmıştır. Bu sonuçlara göre ağaç sayısı arttıkça paralel uygulamanın performansı daha net bir şekilde ortaya çıkmaktadır.

Çizelge 4.31. Fonksiyonların ağaç sayılarına göre hızlanma katsayıları

Fonksiyon / Ağaç Sayısı	10	20	30	40	50	60	70	80	90	100
Sphere	2,38	6,85	10,33	13,17	15,92	19,19	23,78	27,21	30,86	34,39
Sum Squares	2,06	6,55	9,43	14,39	16,70	19,97	24,32	27,77	31,36	34,91
Zakharov	2,45	7,67	13,78	19,79	25,29	29,45	34,21	39,02	42,43	46,56
Sum of Different Powers	3,01	10,39	20,49	31,35	42,36	38,27	52,27	58,12	65,04	69,28
Ackley	3,14	11,01	22,66	35,36	50,09	62,84	80,35	94,94	109,25	123,16
Rastrigin	3,14	11,42	23,84	38,96	56,87	74,07	89,14	110,23	126,48	148,02
Griewank	2,81	11,45	24,22	39,21	58,36	77,58	93,48	112,70	138,62	149,47
Rosenbrock	3,15	11,55	24,52	40,77	59,91	78,24	93,91	112,49	131,78	149,59
Styblinski Tang	3,19	11,95	25,13	41,69	61,67	81,22	97,18	116,50	139,63	159,45
Schwefel	2,96	13,03	26,13	41,17	78,96	87,12	90,50	132,09	155,87	184,65

Çizelge 4.32'de Sphere, Sum Squares, Zakharov, Sum of Different Powers ve Ackley fonksiyonlarının ağaç sayılarına göre seri uygulama ortalama sonuçları ile minimum ve maksimum değerleri verilmiştir.

Çizelge 4.32. Sphere, Sum Squares, Zakharov, Sum of Different Powers ve Ackley fonksiyonlarının ağaç sayılarına göre seri uygulama ortalama sonuçları ile minimum ve maksimum değerleri

Ağaç Sayısı	Sphere	Sum Squares	Zakharov	Sum of Different Powers	Ackley
10	0	0	1,30E-139	0	4,20E-15
20	0	0	1,34E-137	0	4,32E-15
30	0	0	1,05E-133	0	3,61E-15
40	0	0	4,70E-131	0	4,00E-15
50	0	0	5,22E-128	0	3,52E-15
60	0	0	5,86E-126	0	3,02E-15
70	1,96E-303	1,03E-304	2,02E-122	0	3,02E-15
80	2,26E-284	8,26E-286	4,14E-120	0	2,66E-15
90	8,59E-268	3,26E-269	1,53E-117	0	2,90E-15
100	4,81E-253	1,74E-254	5,99E-115	0	2,66E-15
Minimum	0	0	1,30E-139	0	2,66E-15
Maksimum	4,8E-253	1,7387E-254	5,9936E-115	0	4,32E-15

Çizelge 4.33'de Rastrigin, Griewank, Rosenbrock, Styblinski Tang ve Schwefel fonksiyonlarının ağaç sayılarına göre seri uygulama ortalama sonuçları ile minimum ve maksimum değerleri verilmiştir.

Çizelge 4.33. Rastrigin, Griewank, Rosenbrock, Styblinski Tang ve Schwefel fonksiyonlarının ağaç sayılarına göre seri uygulama ortalama sonuçları ile minimum ve maksimum değerleri

Ağaç Sayısı	Rastrigin	Griewank	Rosenbrock	Styblinski Tang	Schwefel
10	1,724596	0,022531545	0,559461155	-385,5357455	165,8138
20	0,198992	0,007229607	0,028471731	-390,2479851	27,63574
30	0,165827	0,006407225	0,040019467	-391,661657	3,948072
40	0,033165	0,003391533	0,025123774	-391,661657	0,000127
50	0	0,002878846	0,025888247	-391,661657	0,000127
60	0	0,004772121	0,018779547	-391,661657	0,000127
70	0	0,004041558	0,023370556	-391,661657	0,000127
80	0	0,002917476	0,030280743	-391,661657	0,000127
90	0	0,003012924	0,027958378	-391,661657	0,000127
100	0	0,002437377	0,029744681	-391,661657	0,000127
Minimum	0	0,002437377	0,018779547	-391,661657	0,000127
Maksimum	1,724596	0,022531545	0,559461155	-385,5357455	165,8138

Çizelge 4.34'de Sphere, Sum Squares, Zakharov, Sum of Different Powers ve Ackley fonksiyonlarının ağaç sayılarına göre paralel uygulama ortalama sonuçları ile minimum ve maksimum değerleri verilmiştir.

Çizelge 4.34. Sphere, Sum Squares, Zakharov, Sum of Different Powers ve Ackley fonksiyonlarının ağaç sayılarına göre paralel uygulama ortalama sonuçları ile minimum ve maksimum değerleri

Ağaç Sayısı	Sphere	Sum Squares	Zakharov	Sum of Different Powers	Ackley
10	0	0	9,34E-142	0	0,366800457
20	0	0	1,85E-125	0	0,311761545
30	0	0	7,17E-125	0	3,61E-15
40	0	0	9,48E-119	0	3,85E-15
50	2,59E-292	1,52E-273	5,22E-128	0	3,61E-15
60	3,92E-261	1,77E-252	1,11E-109	0	3,76E-15
70	2,31E-246	3,13E-239	2,07E-99	0	3,02E-15
80	1,88E-238	1,29E-215	1,33E-96	0	3,17E-15
90	1,92E-216	2,48E-192	9,36E-93	0	3,17E-15
100	4,63E-208	1,17E-195	4,18E-88	0	2,46E-15
Minimum	0	0	9,3383E-142	0	2,45729E-15
Maksimum	4,6284E-208	2,484E-192	4,18E-88	0	0,366800457

Çizelge 4.35'de Rastrigin, Griewank, Rosenbrock, Styblinski Tang ve Schwefel fonksiyonlarının ağaç sayılarına göre paralel uygulama ortalama sonuçları ile minimum ve maksimum değerleri verilmiştir.

Çizelge 4.35. Rastrigin, Griewank, Rosenbrock, Styblinski Tang ve Schwefel fonksiyonlarının ağaç sayılarına göre paralel uygulama ortalama sonuçları ile minimum ve maksimum değerleri

Ağaç Sayısı	Rastrigin	Griewank	Rosenbrock	Styblinski Tang	Schwefel
10	1,052424046	0,02502022	3,704494307	-389,1330679	794,2392807
20	1,154727639	0,032895817	4,31491512	-391,4387036	739,1421741
30	0,710277493	0,029456813	3,555546287	-391,4499962	786,1206278
40	0,75192839	0,025846819	3,908235356	-391,57092	603,3294846
50	0,788920281	0,020532609	3,632651097	-387,8756018	628,078318
60	0,754217952	0,04817493	3,366596777	-390,7791985	586,846404
70	1,18E-07	0	2,69026667	-391,6614097	452,1222117
80	0,035451127	0	2,452534414	-391,6616119	387,6242388
90	0,021566115	0	2,641474617	-391,6615931	220,1357436
100	0,053819148	2,07E-16	2,657859749	-391,6616548	266,8443514
Minimum	1,18135E-07	0	2,452534414	-391,6616548	220,1357436
Maksimum	1,154727639	0,04817493	4,31491512	-387,8756018	794,2392807

5. SONUÇLAR VE ÖNERİLER

5.1 Sonuçlar

Yapılan çalışmada doğa esinli sezgisel optimizasyon algoritmalarının en yenilerinden birisi olan Ağaç-Tohum algoritması, CUDA paralel programlama altyapısına uygun olarak yeniden kodlanarak paralel uygulamanın aynı kalitede sonuçları, çok daha hızlı bulduğu ortaya koyulmuştur.

Çalışmada kullanılan kıyas fonksiyonları olabildiğince farklı seçilerek performansın dağılımı gözlemlendi. Ağaç sayılarının 100 alındığı paralel uygulamanın sonuçlarına göre bir çok yerel minimumu olan fonksiyonlardan Ackley(123,16 kat), Griewank(149,47 kat), Rastrigin(148,02 kat) ve Schwefel(184,65 kat) hızlı sonuca ulaşmıştır. Konveks(dışbükey) kıyas fonksiyonlarından Sphere(34,39 kat), Sum of Different Powers(69,28 kat), Sum Squares(34,91 kat) hızlı bir şekilde sonuca ulaşmıştır. Levha(plaka) şekilli bir kıyas fonksiyonu olan Zakharov fonksiyonu 46,56 kat daha hızlı bir şekilde sonuca ulaşmıştır. Vadi şekilli bir test fonksiyonu olan Rosenbrock(149,59 kat) hızla sonuca ulaşmıştır. Yine kıyas fonksiyonlarından Styblinski-Tang(159,45 kat) hızla sonuca ulaşmıştır.

Seri uygulamanın sonuçları incelendiğinde ağaç sayısına bağlı olarak düzenli olarak katlamalı artan bir hız kaybı ortadayken, paralel uygulama için test ettiğimiz aralıkta (10-100 ağaç arası) ağaç sayısının artmasının performansı etkilemediği görülmüştür. Bu da çok büyük optimizasyon problemlerinin çok hızlı çözülmesi için önerdiğimiz CUDA tabanlı uygulamanın daha kullanılabilir olduğunu ortaya koymaktadır.

Yapılan tez çalışmasında maksimum çevrim sayısı ağaç sayısı ve problemin boyutuna bağlı olarak **Problemin Boyutu x 10000 / Ağaç Sayısı** şeklinde alınmış, böylece bir çok kıyas fonksiyonu global minimum değerlerine ulaşmıştır.

Sonuçların kalitesi ve dış ortamdan etkilenmemesi adına her deney düzeneği 30 kere çalıştırılmış, bu çalıştırılmaların sonuçlarının ortalaması alınmış, ayrıca alınan süreler de bu çalıştırmaların ortalama süresidir. Yine bu çalıştırılmaların salınımlarının gözlemlenebilmesi için standart sapma değerleri ilgili çizelgelere eklenmiştir.

Rosenbrock ve Schwefel kıyas fonksiyonları hariç diğer 8 kıyas fonksiyonunda bir çok defa global minimuma erişilmiş, erişilemediği durumda da en uygun değere yakın değerlerle kaliteli çözümler üretilmiştir.

Rosenbrock fonksiyonu için en uygun değer olan 0'ı seri uygulama bazı ağaç sayıları için yakalamış fakat bazı sonuçlarda paralel uygulama gibi ancak yaklaşabilmiştir.

Schwefel fonksiyonunun yapısı gereği hem seri, hem de paralel uygulama çeşitli yerel minimumlara takılarak ortak bir çözüme ulaşamamışlardır. Kıran (2015)'ın çalışmasında vermiş olduğu ortalama ve standart sapma değerlerinin çalışmamızda ağaç sayısının 30 alındığı durumla aynı olması iki çalışmanın aynı yerel minimuma takıldığını göstermektedir. Yine Kıran (2015)'ın çalışmasında Schwefel fonksiyonu için ilgili çalışmada kıyaslanan algoritmalarında farklı farklı değerler bulması bu durumun normal olarak değerlendirilebileceğini anlatmaktadır.

5.2 Öneriler

Her geçen gün yeni bir optimizasyon algoritması önerildiğinden, işlemci teknolojisinin giderek çok çekirdekli bir hal almasından, grafik kartlar ile genel amaçlı hesaplama işleminin yaygınlaşmasından dolayı paralel olarak programlanabilecek her tür uygulamanın CUDA mimarisine göre dizayn edilmesi zamansal anlamda ciddi kazançları beraberinden getirecektir.

CUDA, NVIDIA firmasının güçlü desteği ile her geçen gün farklı platformlara verdiği desteği artırmakta, birçok programlama dili ile bütünleşmiş hale gelmekte, sürekli genişleyen kütüphaneleriyle daha önceden çok zor olan işlerin çok kolay bir şekilde halledilebilmesine zemin hazırlamaktadır. Bu doğrultuda akademik camiada yaygın olarak kullanılan MATLAB programı ile oldukça kolaylaştıran kod yazımı ve Paralel Hesaplama Araç Kutusu(Parallel Computing Toolbox) süreçte değerlendirilmesi ve kullanılması gereken yazılımlardır.

Özellikle hesaplanması çok uzun süren gerçek dünya problemlerinden paralelleştirilebilecek olanlar için çok ideal bir çözüm olan CUDA ile paralel programlama yaklaşımı, probleme bağlı olarak binlerce kat hızlandırma yapabilmesiyle geleceğe ışık tutmaktadır.

KAYNAKLAR

Akay, B., 2009, Nümerik optimizasyon problemlerinde yapay arı kolonisi (artificial bee colony) algoritmasının performans analizi, *Doktora Tezi, Erciyes Üniversitesi Fen Bilimleri Enstitüsü, Kayseri*.

Akçay, M. ve Erdem, H. A., 2010, Paralel Hesaplama ve Matlab Uygulamaları, *Akademik Bilişim Muğla Üniversitesi*.

Akçay, M., Şen, B., Orak, İ. ve Çelik, A., 2011, Paralel Hesaplama ve CUDA, *International Advanced Technology, IATS*, 26-28.

Akgün, D. ve Erdoğmuş, P., 2015, GPU accelerated training of image convolution filter weights using genetic algorithms, *Applied Soft Computing*, 30, 585-594.

Altıntaş, V. ve Yegenoğlu, E., 2011, Görüntü İşlemede Seri ve Paralel Programlamanın Performansı, *6th International Advanced Technologies Symposium (IATS'11)*, 16-18.

Ananth, G., Gupts, A., Karypis, G. ve Kumar, V., 2003, Introduction to Parallel computing, Boston, MA: Addison-Wesley.

Barney, B., 2010, Introduction to parallel computing, *Lawrence Livermore National Laboratory*, 6 (13), 10.

Beni, G. ve Wang, J., 1993, Swarm Intelligence in Cellular Robotic Systems, In: Robots and Biological Systems: Towards a New Bionics?, Eds: Dario, P., Sandini, G. ve Aebischer, P., *Berlin, Heidelberg*: Springer Berlin Heidelberg, p. 703-712.

Blum, C. ve Roli, A., 2003, Metaheuristics in combinatorial optimization: Overview and conceptual comparison, *ACM Computing Surveys (CSUR)*, 35 (3), 268-308.

Bukata, L., Šůcha, P. ve Hanzálek, Z., 2015, Solving the Resource Constrained Project Scheduling Problem using the parallel Tabu Search designed for the CUDA platform, *Journal of Parallel and Distributed Computing*, 77, 58-68.

Bukharov, O. E. ve Bogolyubov, D. P., 2015, Development of a decision support system based on neural networks and a genetic algorithm, *Expert Systems with Applications*, 42 (15), 6177-6183.

Çelik, A. ve Özmen, A., 2009, DAĞITIK PARALEL SİSTEMLER HAKKINDA KIYASLAMALI BİR ÇALIŞMA: PVM VE MPI A COMPARATIVE STUDY ON DISTRIBUTED PARALLEL SYSTEMS: PVM AND MPI.

Dorigo, M., Maniezzo, V., Colorni, A. ve Maniezzo, V., 1991, Positive feedback as a search strategy.

Eberhart, R. ve Kennedy, J., 1995, A new optimizer using particle swarm theory In: Proceedings of the sixth international symposium on micro machine and human science, vol 43 IEEE, *New York*.

Eldem, H., 2014, Karınca Koloni Optimizasyonu (KKO) ve Parçacık Sürü Optimizasyonu (PSO) Algoritmaları Temelli Bir Hiyerarşik Yaklaşım Geliştirilmesi, *Selçuk Üniversitesi*.

Farmer, J. D., Packard, N. H. ve Perelson, A. S., 1986, The immune system, adaptation, and machine learning, *Physica D: Nonlinear Phenomena*, 22 (1), 187-204.

Feo, T. A. ve Resende, M. G., 1995, Greedy randomized adaptive search procedures, *Journal of global optimization*, 6 (2), 109-133.

Gazioğlu, E., 2015, Bilişsel Radyo Ağlarında Üst Sezgiseller ile Kanal Atama Probleminin Çözülmesi, *İstanbul Teknik Üniversitesi*.

Glover, F., 1986, Future paths for integer programming and links to artificial intelligence, *Computers & operations research*, 13 (5), 533-549.

Güneş, A., 2011, Paralel programlama ile el yazısı rakamlarının tanınması, *SDÜ Fen Bilimleri Enstitüsü*.

Haklı, H., 2013, Sürekli Fonksiyonların Optimizasyonu için Doğa Esinli Algoritmaların Geliştirilmesi, Yüksek Lisans Tezi, *Selçuk Üniversitesi Fen Bilimleri Enstitüsü*, 1-2.

Hansen, P., 1986, The steepest ascent mildest descent heuristic for combinatorial programming, *Congress on numerical methods in combinatorial optimization, Capri, Italy*, 70-145.

Janousešek, J., Gajdoš, P., Radecký, M. ve Snášel, V., 2014, Classification via Nearest Prototype Classifier Utilizing Artificial Bee Colony on CUDA, *International Joint Conference SOCO'14-CISIS'14-ICEUTE'14*, 21-30.

Kaçka, S., 2011, Bilimsel hesaplama problemlerinin çözümünde paralel hesaplama yöntemlerinin kullanılması, *Selçuk Üniversitesi Fen Bilimler Enstitüsü*.

Kai, Z., Ming, Q., Lin, L. ve Xiaoming, L., 2015, Solving Graph Coloring Problem by Parallel Genetic Algorithm Using Compute Unified Device Architecture, *Journal of Computational and Theoretical Nanoscience*, 12 (7), 1201-1205.

Kalivarapu, V. ve Winer, E., 2015, A study of graphics hardware accelerated particle swarm optimization with digital pheromones, *Structural and Multidisciplinary Optimization*, 51 (6), 1281-1304.

Karaboga, D., 2005, An idea based on honey bee swarm for numerical optimization, *Technical report-tr06, Erciyes university, engineering faculty, computer engineering department*.

Karaboğa, D., 2011, Yapay Zeka Optimizasyon Algoritmaları, Nobel Yayın Dağıtım.

Kıran, M. S., 2015, TSA: Tree-seed algorithm for continuous optimization, *Expert Systems with Applications*, 42 (19), 6686-6698.

Kıran, M. S., 2016, An Implementation of Tree-Seed Algorithm (TSA) for Constrained Optimization, In: Intelligent and Evolutionary Systems, Eds: Springer, p. 189-197.

Kirkpatrick, S., 1984, Optimization by simulated annealing: Quantitative studies, *Journal of statistical physics*, 34 (5-6), 975-986.

Kneusel, R., 2014, Curve-Fitting on Graphics Processors Using Particle Swarm Optimization, *International Journal of Computational Intelligence Systems*, 7 (2), 213-224.

Lastra, M., Molina, D. ve Benítez, J. M., 2015, A high performance memetic algorithm for extremely high-dimensional problems, *Information Sciences*, 293, 35-58.

Lucic, P. ve Teodorovic, D., 2001, Bee system: modeling combinatorial optimization transportation engineering problems by swarm intelligence, *Preprints of the TRISTAN IV triennial symposium on transportation analysis*, 441-445.

Mussi, L., Daolio, F. ve Cagnoni, S., 2011, Evaluation of parallel particle swarm optimization algorithms within the CUDA™ architecture, *Information Sciences*, 181 (20), 4642-4657.

Nvidia, C., 2015, CUDA C Programming Guide 7.5, *NVIDIA Corporation, Jul*.

Ortakcı, Y., 2011, Parçacık sürü optimizasyonu yöntemlerinin uygulamalarla karşılaştırılması, *Karabük Üniversitesi*.

Ouyang, A., Tang, Z., Zhou, X., Xu, Y., Pan, G. ve Li, K., 2015, Parallel hybrid pso with cuda for ld heat conduction equation, *Computers & Fluids*, 110, 198-210.

Pacheco, P. S., 1997, Parallel programming with MPI, Morgan Kaufmann, p.

Parhami, B., 2006, Introduction to parallel processing: algorithms and architectures, Springer Science & Business Media, p.

Passino, K. M., 2002, Biomimicry of bacterial foraging for distributed optimization and control, *Control Systems, IEEE*, 22 (3), 52-67.

Peker, M., Şen, B. ve Gürüler, H., 2015, Rapid Automated Classification of Anesthetic Depth Levels using GPU Based Parallelization of Neural Networks, *Journal of medical systems*, 39 (2), 1-11.

Pham, D., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S. ve Zaidi, M., 2005, The bees algorithm. Technical note, *Manufacturing Engineering Centre, Cardiff University, UK*, 1-57.

Platos, J., Snasel, V., Jezowicz, T., Kromer, P. ve Abraham, A., 2012, A PSO-based document classification algorithm accelerated by the CUDA platform, *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, 1936-1941.

Rymut, B. ve Kwolek, B., 2015, Real-time multiview human pose tracking using graphics processing unit-accelerated particle swarm optimization, *Concurrency and Computation: Practice and Experience*, 27 (6), 1551-1563.

Sarıman, G., 2012, Paralel programlama ile web madenciliğinde log analizi, *SDÜ Fen Bilimleri Enstitüsü.*

Sayar, A. ve Ergün, U., 2014, Fonksiyonel Programlama Dilleri ile Paralel Programlama, *Niğde Üniversitesi Mühendislik Bilimleri Dergisi*, 3 (2), 1-17.

Silva, E. H. ve Bastos Filho, C. J., 2015, PSO Efficient Implementation on GPUs Using Low Latency Memory, *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 13 (5), 1619-1624.

Solomon, S., Thulasiraman, P. ve Thulasiram, R., 2011, Collaborative multi-swarm PSO for task matching using graphics processing units, *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 1563-1570.

Şahin, U., 2014, Trendden Arındırılmış Finans Verileri Üzerinde Evrimsel Algoritmalar ile Salınım-Tabanlı Teknik Analiz Göstergelerinin Parametre Eniyilemesi, *TOBB Ekonomi ve Teknoloji Üniversitesi.*

Temirci, G., 2009. Cluster Computing. Erişim Adresi, http://web.firat.edu.tr/bilmuh/gaydin/dersler/0809/bmu401/ppt/clusterComputing.doc

Teodorović, D. ve Dell'Orco, M., 2005, Bee colony optimization–a cooperative learning approach to complex transportation problems, *Advanced OR and AI Methods in Transportation: Proceedings of 16th Mini–EURO Conference and 10th Meeting of EWGT (13-16 September 2005).–Poznan: Publishing House of the Polish Operational and System Research*, 51-60.

Tsuchida, Y. ve Yoshioka, M., 2015, A Parallelization Method for Neural Network Learning, *Electrical Engineering in Japan*, 191 (2), 17-23.

Vajda, A., 2011, Programming many-core chips, Springer Science & Business Media, p.

Wang, P., Li, H. ve Zhang, B., 2015, A GPU-based Parallel Ant Colony Algorithm for Scientific Workflow Scheduling, *International Journal of Grid and Distributed Computing*, 8 (4), 37-46.

William, G., Ewing, L. ve Thomas, S., 2003, Beowulf cluster computing with Linux, The MIT Press, Cambridge, Massachusetts London, England.

Yang, X.-S., 2005, Engineering optimizations via nature-inspired virtual bee algorithms, In: Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach, Eds: Springer, p. 317-323.

Yuan, H., Zhao, T., Yang, W. ve Pan, H., 2015, 1821. Annealing evolutionary parallel algorithm analysis of optimization arrangement on mistuned blades with non-linear friction, *Journal of Vibroengineering*, 17 (8).

Zarrabi, A., Samsudin, K. ve Karuppiah, E. K., 2015, Gravitational search algorithm using CUDA: a case study in high-performance metaheuristics, *The Journal of Supercomputing*, 71 (4), 1277-1296.

Zhang, Z. ve Seah, H. S., 2012, CUDA acceleration of 3D dynamic scene reconstruction and 3D motion estimation for motion capture, *Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on*, 284-291.

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Ahmet Cevahir ÇINAR
Uyruğu : TC
Doğum Yeri ve Tarihi : Çumra, 07.10.1986
Telefon : +90.506.229.73.39
Faks : +90.332.241.72.02
e-mail : accinar@selcuk.edu.tr

EĞİTİM

Derece		**Adı, İlçe, İl**	**Bitirme Yılı**
Lise	:	A. Meteoroloji Mes. Lis., Keçiören, Ankara	2004
Üniversite	:	Selçuk Üni., Bilgisayar Mühendisliği	2009
Yüksek Lisans	:	Selçuk Üni., Bilgisayar Mühendisliği	-
Doktora	:	-	

İŞ DENEYİMLERİ

Yıl	**Kurum**	**Görevi**
2005-	Meteoroloji 8. Bölge Müdürlüğü	Meteorolojist

YABANCI DİLLER

İngilizce

YAYINLAR

Ş. Taşdemir, A. C. Çınar, MENDEL'2011 konferansı dahilinde "17th International Conference on Soft Computing" bildiri kitapçığındaki "Application of Artificial Neural Network Forecasting of Daily Maximum Temperature in Konya", 236-243 pp., Brno, Czech Republic, Haziran 2011.