Journal of the Faculty of Engineering and Architecture of Gazi University 33:4 (2018) 1397-1409

Mühendislik Mimarlık Fakültesi Dergisi
Journal of The Faculty of Engineering and Architecture of Gazi University

Elektronik / Online ISSN : 1304 - 4915
Basılı / Printed ISSN : 1300 - 1884

A parallel implementation of Tree-Seed Algorithm on CUDA-supported graphical processing unit

Ahmet Cevahir Çınar[1], Mustafa Servet Kıran[2]

[1]Turkish State Meteorological Service, Konya Division, 42090 Konya, Turkey.
[2]Konya Technical University, Faculty of Engineering and Natural Sciences, Department of Computer Engineering, 42075, Konya, Turkey.

Highlights:

- A parallel version of Tree-Seed Algorithm which is one of the recently proposed algorithms has been presented.
- The parallel Tree-Seed algorithm has been implemented within CUDA platform.
- The performance analysis of parallel TSA has been investigated on benchmark functions and compared with the serial version of TSA.

Keywords:

- Tree-seed algorithm
- CUDA
- Parallel computation
- Benchmark function

Article Info:

Received: 16.03.2017
Accepted: 21.08.2017

DOI:

10.17341/gazimmfd.416436

Acknowledgement:

The authors wish to thank Scientific Research Projects Coordinatorship at Selcuk University (Research Project No: 15401121) and The Scientific and Technological Research Council of Turkey for their institutional supports.

Correspondence:

Author: Mustafa Servet Kiran
e-mail: mskiran@selcuk.edu.tr
phone: +90 332 223 8644

Graphical/Tabular Abstract

In recent years, while the collected data are increased, we need effective computation methods to process these data. Due to the fact that most of the real world problems are difficult to solve, swarm intelligence and evolutionary computation algorithms are interested because they guarantee the near optimal solution for the problem in a reasonable time but not guarantee the optimal solution. In another perspective, if the data or process can be parallelized, the parallel computation is a good choice instead of serial programming approaches in terms of time effectiveness. In this study, the tree-seed algorithm, which is a recently proposed population-based iterative search algorithm, is implemented within CUDA platform in parallel. The performance of the parallel version of the algorithm has been investigated on the benchmark functions and compared with the performance of the serial version of the algorithm. The dimensionality of the problems is taken as 10 and the performance analysis and comparisons have been conducted under the condition of different sizes of the population. Experimental studies show that the parallel version of the algorithm is accelerated to 184.65 times in accordance with the serial version of the algorithm on some problems.

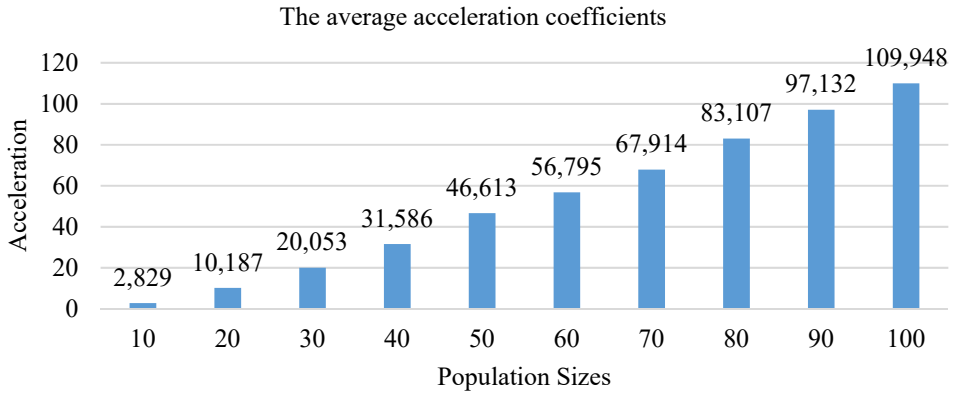


Figure A. The average acceleration coefficients

Purpose: The main purpose is the parallelization of tree-seed algorithm.

Theory and Methods:
In this work, one of the newest nature-inspired metaheuristic optimization algorithms, Tree-Seed algorithm (TSA) has been re-coded within CUDA parallel programming infrastructure and it has been shown that parallel application is more efficient than serial application in terms of time.

Results:
When the results of the serial and parallel applications of TSA are compared, it is seen that there is a steady increase in speed due to the number of trees. This suggests that the CUDA-based application that we recommend for faster resolution of very large optimization problems is more usable.

Conclusion:
It is concluded that CUDA is a good alternative to accelerate swarm intelligence and evolutionary computation algorithms in this study.

Journal of the Faculty of Engineering and Architecture of Gazi University 33:4 (2018) 1397-1409

Mühendislik Mimarlık Fakültesi Dergisi
Journal of The Faculty of Engineering and Architecture of Gazi University

Elektronik / Online ISSN : 1304 - 4915
Basılı / Printed ISSN : 1300 - 1884

Ağaç-tohum algoritmasının CUDA destekli grafik işlem birimi üzerinde paralel uygulaması

Ahmet Cevahir Çınar[1], Mustafa Servet Kıran*[2]

[1] Meteoroloji Genel Müdürlüğü, Meteoroloji 8. Bölge Müdürlüğü, 42090, Konya, Türkiye
[2] Konya Teknik Üniversitesi, Mühendislik ve Doğa Bilimleri Fakültesi, Bilgisayar Mühendisliği Bölümü, 42075, Konya, Türkiye

Ö N E Ç I K A N L A R

- Bu makale literatüre yeni kazandırılmış olan Ağaç-tohum algoritmasının paralel bir versiyonunu sunar
- Ağaç-tohum algoritmasının CUDA platformu içerisinde paralel versiyonu geliştirilmiştir
- Performans analizi kıyas fonksiyonları üzerinde yapılmış ve seri versiyonu ile karşılaştırılmıştır

Makale Bilgileri
Geliş: 16.03.2017
Kabul: 21.08.2017

DOI:
10.17341/gazimmfd.416436

Anahtar Kelimeler:
Ağaç-Tohum algoritması, Birleşik Aygıt Mimarisi Hesaplama, paralel hesaplama, kıyas fonksiyonu

ÖZET

Son yıllarda toplanan verinin artmasıyla birlikte verimli hesaplama yöntemlerinin de geliştirilmesi ihtiyacı artmaktadır. Çoğunlukla gerçek dünya problemlerinin zor olması sebebiyle optimal çözümü garanti etmese dahi makul zamanda yakın optimal çözümü garanti edebilen sürü zekâsı veya evrimsel hesaplama yöntemlerine olan ilgi de artmaktadır. Diğer bir açıdan seri hesaplama yöntemlerinde verinin veya işlemin paralelleştirilebileceği durumlarda paralel algoritmaların da geliştirilmesi ihtiyacı ortaya çıkmıştır. Bu çalışmada literatüre son yıllarda kazandırılmış olan popülasyon tabanlı ağaç-tohum algoritması ele alınmış ve CUDA platformu içerisinde paralel versiyonu geliştirilmiştir. Algoritmanın paralel versiyonunun performansı kıyas fonksiyonları üzerinde analiz edilmiş ve seri versiyonunun performansı ile karşılaştırılmıştır. Kıyas fonksiyonlarında problem boyutluluğu 10 olarak alınmış ve farklı popülasyon ve blok sayıları altında performans analizi yapılmıştır. Deneysel çalışmalar algoritmanın paralel versiyonunun algoritmanın seri sürümüne göre bazı problemler için 184,65 kata performans artışı sağladığı görülmüştür.

A parallel implementation of Tree-Seed Algorithm on CUDA-supported graphical processing unit

H I G H L I G H T S

- In this study, a parallel version of Tree-Seed Algorithm which is one of the recently proposed algorithms has been presented.
- The parallel version of Tree-Seed algorithm has been implemented within CUDA platform.
- The performance analysis of parallel TSA has been investigated on benchmark functions and compared with the serial version of TSA.

Article Info
Received: 16.03.2017
Accepted: 21.08.2017

DOI:
10.17341/gazimmfd.416436

Keywords:
Tree-seed algorithm, Compute Unified Device Architecture, parallel computation, benchmark function

ABSTRACT

In recent years, while the collected data are increased, we need effective computation methods to process these data. Due to the fact that most of the real world problems are difficult to solve, swarm intelligence and evolutionary computation algorithms are interested because they guarantee the near optimal solution for the problem in a reasonable time but not guarantee the optimal solution. In another perspective, if the data or process can be parallelized, the parallel computation is a good choice instead of serial programming approaches in terms of time effectiveness. In this study, the tree-seed algorithm, which is a recently proposed population-based iterative search algorithm, is implemented within CUDA platform in parallel. The performance of the parallel version of the algorithm has been investigated on the benchmark functions and compared with the performance of the serial version of the algorithm. The dimensionality of the problems is taken as 10 and the performance analysis and comparisons have been conducted under the condition of different sizes of the population. Experimental studies show that the parallel version of the algorithm is accelerated to 184.65 times in accordance with the serial version of the algorithm on some problems.

*Sorumlu Yazar/Corresponding Author: mskiran@selcuk.edu.tr / Tel: +90 332 223 8644

1.GİRİŞ (INTRODUCTION)

Son zamanlarda paralel hesaplama ve programlama yaklaşımının önemi gittikçe artmaktadır. NVIDIA® firması tarafından sunulan bir paralel programlama mimarisi olan CUDA® 'nın uygulamaları kolay kullanım ve arkasındaki güçlü destekten dolayı gittikçe yaygınlaşmaktadır. 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 günümüzde genel ve özel amaçlı hesaplamalar için de kullanılmaya başlanmıştır [1].

Optimizasyon (en iyileme); bir sistemin tasarımında olası tüm çözümlerin arasından en iyisinin bulunması olarak ifade edilebilir. 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 [2].

Doğa esinli metasezgisel optimizasyon algoritmaları, basit uyarlanabilirlik, verimli çalışma ve farklı yapıdaki çözüm uzayına sahip problemlerin çözümündeki etkinlik nedeniyle literatürde oldukça çok çalışılan konuların başında gelmektedir. Bu algoritmalar zor problemlerin çözümünde göstermiş oldukları başarılardan dolayı birçok bilim dalında, mühendislik problemlerinde ve askeri uygulamalarda sıklıkla kullanılmaktadır [3]. Bu algoritmaların esin kaynağı doğadaki canlı veya cansız varlıkların özellik ve davranışlarıdır. Bu çalışmada kullanılan Ağaç-Tohum algoritması sürekli optimizasyon problemlerinin çözümü için ağaçlar ve tohumlar arasındaki çoğalma (neslin veya türün devamı) ilişkisinden yola çıkılarak önerilmiştir [4]. 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[4] ve kısıtlı bir problem olan basınç tankı tasarımı [5], radyal tabanlı fonksiyon sinir ağı eğitimi [6] gibi problemlerin çözümünde kullanılmıştır.

CUDA platformu Nvidia firması tarafından 2007 yılında duyurulmuştur [7] ve doğa esinli metasezgisel algoritmalar kullanılarak yapılmış CUDA tabanlı çalışmalar gün geçtikçe artmaktadır. Mussi ve Cagnoni [8] çalışmalarında Parçacık Sürü Optimizasyonu (PSO) algoritması ile 22 kat hızlanma elde etmişlerdir. Molnár vd. [9] hava kirliliği modeli oluştururken kabul edilebilir bir zamanda, en doğruya yakın çözümler üreten bir sistem kurmayı amaçlamışlardır. Çalışmanın sonucunda yaklaşık 60 ila 120 kat hızlanma elde edildiği açıklanmıştır. Mussi vd. [10] PSO algoritmasını paralelleştirerek CUDA tabanlı uygulamasını geliştirmiştir. Farklı grafik kartları ve farklı boyutlarla yapılan testler sonucunda yaklaşık 8 ila 138 kat hızlanma elde edilmiştir. Ayrıca gelişmiş sürücü yönetim sistemi problemlerinden yol sinyal algılama için bir uygulama yapmışlardır. Solomon vd. [11] çalışmalarında görev eşleştirme (task matching) problemini PSO algoritması ile CUDA tabanlı uygulamalar ile 37 kat daha hızlı çözmüştür. Zhang ve Seah [12] hareket yakalama üzerine yaptıkları çalışmalarında PSO destekli stokastik arama algoritması ile CUDA tabanlı bir uygulama gerçekleştirmişlerdir. Platos vd. [13] CUDA ile PSO tabanlı Doküman Sınıflandırma Algoritmasını hızlandırmışlardır. Kumar vd. [14] CUDA üzerinde Cooperative Particle Swarm Optimization (CPSO) algoritmasını gerçeklemiştir. Farklı popülasyon boyutlarında 10-12 kat hızlanma elde edilmiştir. Wang vd. [15] Farksal Gelişim (Differential Evolution) algoritmasını GPU üzerinden paralelleştirmiştir ve (GOjDE) ismini önermiştir. Bu çalışma daha önce önerilen Generalized Opposition based Differential Evolution (GODE) [16] isimli çalışmanın geliştirilmesiyle ortaya çıkmıştır. Çalışmada algoritmanın performansını en üst düzeye çıkarmak için kendi kendine uyarlanan bir parametre ayarlama stratejisi önerilmiştir. Çalışmada 1,15 ve 7,84 hızlanma elde edilmiş, ayrıca boyut büyüdükçe hızlanma azalmıştır. Bu beklenmeyen durumun thread sayısının 128 ile sınırlandırılmasından dolayı olduğunu izah etmişlerdir. Popülasyon sayısı=Thread sayısı alınmış ve 128 ile sınırlandırıldığından istenen performans elde edilememiştir. Kneusel [17] çalışmasında PSO algoritması destekli CUDA tabanlı bir uygulama ile 2 boyutlu resimlerin işlenmesi üzerine bir çalışma yapmıştır. Luo vd. [18] Arı Algoritması (Bees Algorithm) için paralel bir yaklaşım geliştirilerek CUBA (CUDA based Bees Algorithm) ismini vermiştir. Paralel yaklaşım kıyas fonksiyonlarına bağlı olarak 13-56 kat arasında hızlanma elde etmiştir. Janousešek vd. [19] sınıflandırma problemlerinin çözümünde Yapay Arı Kolonisi (Artificial Bee Colony) algoritmasını paralelleştirerek CUDA altyapısı ile uygulama geliştirmişlerdir. Geliştirilen uygulamanın 86,25 kat daha hızlı çalıştığı gözlemlenmiştir. Rymut ve Kwolek [20] 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 vd. [21] 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 [22] sinir ağları ve genetik algoritma ile bir karar destek sisteminin geliştirilmesinde CUDA'dan yararlanmışlardır. 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 vd. [23] 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 vd. [24] CUDA altyapısını kullanarak Paralel Genetik Algoritma ile Grafik Boyama Sorunu'nu çözmeye çalışmışlardır. Grafik boyutu büyüdükçe seriye göre tartışmasız bir üstünlük çıktığı belirtilmiştir. Kalivarapu ve Winer [25], 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 [26], 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 [27], genetik algoritmalar kullanarak görüntü evrişim filtresi ağırlıklarının eğitimini CUDA ile hızlandırmışlardır. 256x256, 512x512, 1024x1024 boyutlu resimler kullanarak seri, direk paralel metot, popülasyon tabanlı paralel metot, blok tabanlı paralel metot ve alt resimler tabanlı paralel metot olarak ayrı ayrı çalıştırılmıştır. 55 ila 90 kat hızlanma elde edilmiştir. Zarrabi vd. [28] 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 [29], 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 metotla kıyaslandığında parametrelere göre 3 ila 6 kat hızlanma sağlanmıştır. Ouyang vd. [30] bir boyutlu ısı iletim denklemi için CUDA ile paralel hibrit PSO algoritmasını önermiştir. İki farklı GPU'da yapılan testler sonucunda 12,12 ve 13,06 kat hızlanma elde edilmiştir. Bukata vd. [31] 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 vd. [32] anestezik derinlik düzeylerinin hızlı ve otomatik sınıflandırılması için sinir ağları kullanarak GPU tabanlı paralelleştirme yapmışlardır. Anestezik derinlik düzeyinin belirlenmesi için kullanılması gereken çok büyük ve kompleks bir veri seti olduğundan dolayı CUDA ile yapılan işlemler hızlandırılmıştır. Lastra vd. [33] 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. Çınar ve Kıran [34] Ağaç-Tohum algoritmasının paralel bir versiyonunun 5 test fonksiyonu üzerindeki sonuçlarını bir bildiri olarak sunmuşlardır. Bu çalışmada ise iki katı sayıda fonksiyon üzerinde yöntemin performansı detaylı olarak analiz edilmiş ve grafikler üzerinde sonuçlar tartışılmıştır. Ayrıca fonksiyon karmaşıklığı ve karakteristiklerinin hızlanmaya olan etkileri de bu çalışmada detaylı olarak açıklanmaktadır. Kısacası mevcut çalışma sempozyum bildirisinin [34] genişletilmiş ve detaylandırılmış bir versiyonudur. Kıran [5], 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. Bu makalede Ağaç-Tohum algoritmasının paralel olarak gerçeklenmesi sağlanmış ve test problemlerini çok daha kısa zamanda çözdüğü gösterilmiştir. Paralelleştirme yapılırken adım adım yapılan bazı işlemlerin aynı anda yapılması için çeşitli teknikler kullanılmış ve bunlar ayrıntılı olarak anlatılmıştır. Makalenin geri kalan kısmı şu şekilde düzenlenmiştir. 2. bölümde Ağaç-Tohum Algoritması (TSA) anlatılmış, CUDA ile ilgili kısa bir bilgi verilmiş, TSA algoritmasının paralelleştirilmesi için yapılan değişikler izah edilmiş, makalede kullanılan kıyas fonksiyonları tanıtılmıştır. 3. bölümde kurulan test ortamı ile birlikte bulgular ve sonuçlar verilmiştir. 4. bölümde ise sonuçlar ve geleceğe yönelik araştırılabilir konular üzerinde durulmuştur.

2. AĞAÇ-TOHUM ALGORİTMASI
(TREE-SEED ALGORITHM)

Ağaç-Tohum algoritması sürekli optimizasyon problemlerinin çözümü için ağaçlar ile tohumlar arasındaki ilişkiden yola çıkılarak önerilmiş olan doğa esinli popülasyon tabanlı iteratif optimizasyon algoritmalarından biridir [4]. Optimizasyon problemi ağaçların ve tohumların üretildiği çevre, problem için olası bir çözüm ise ağaç ve tohum pozisyonlarına karşılık gelmektedir. Başlangıçta rastgele çevreye dağıtılan ağaçlar ayrık zamanda tohumların

üretilmesi ile uzayın araştırılması sağlanır. Ağaç-tohum algoritmasının her bir iterasyonunda her bir ağaç için belirli sayıda tohum üretilir. Bu üretilen tohumlar üretildiği ağaç ile karşılaştırılır ve eğer tohumun (çözümün) kalitesi daha iyiyse ağaç popülasyondan silinir ve bu tohum artık bir ağaç olarak popülasyona yerleştirilir. Bu süreç önceden belirlenen bir durdurma kriteri sağlanana kadar sürdürülür. Durdurma kriteri olarak maksimum çevrim sayısı, maksimum fonksiyon çağrı sayısı ve belirli bir optimizasyon hatası gibi kriterlerden biri algoritmada kullanılabilir.

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. Her ağacın tohum sayısı rastgele olarak belirlenmektedir. Tohum üretmek için iki farklı denklem kullanılabilir. Araştırma eğilimi (search tendency, ST) parametresi ise tohum oluşumu sırasında bu denklemlerden hangisinin seçileceğini belirlemek amacıyla kullanılır. 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 belirlemek için kullanılır.

Tohum oluşumu sırasında araştırma eğilimi parametresine göre Eş. 1 ve Eş. 2 eşitlikleri kullanılır. Algoritmanın parametresi olan ST 0-1 arasında bir sayı olarak başlangıçta belirlenir. 0-1 arasında üretilen rastgele bir sayı eğer araştırma eğilimi parametresinden küçükse Eş. 1, büyükse Eş. 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.

$$S_{k,j}=T_{i,j}+\alpha_{i,j}\times(B_j-T_{r,j}) \quad (1)$$

$$S_{k,j}=T_{i,j}+\alpha_{i,j}\times(T_{i,j}-T_{r,j}) \quad (2)$$

Burada $S_{k,j}$ k. tohumun j.boyutunu (i. ağaç için üretilmektedir), $T_{i,j}$ i.ağacın, j.boyutunu, $\alpha_{i,j}$ -1,1 arasında üretilen ölçekleme parametresi, B_j şimdiye kadar elde edilen en iyi ağacın j.boyutunu, $T_{r,j}$ r.ağacın j.boyutunu; r rastgele seçilen i'den farklı bir ağaç olduğunu göstermektedir.

Başlangıçta çözülecek optimizasyon probleminin sınırları içerisinde rastgele ağaçlar Eş. 3'e göre oluşturulur.

$$T_{i,j}=L_{j,min}+r_{i,j}\left(H_{j,max}-L_{j,min}\right) \quad (3)$$

Burada $L_{j,min}$ arama uzayının alt sınırını (j.parametre için), $H_{j,max}$ ise arama uzayının üst sınırını, $r_{i,j}$ 0-1 arasında üretilen rastgele sayıyı (i.ağacın j.parametresi için) göstermektedir.

Minimizasyon problemleri için en iyi çözüm Eş. 4'te gösterildiği üzere tüm ağaçların fonksiyon değerleri hesaplanarak bulunur.

$$B=\mathrm{ArgMin}\left\{f(\vec{T_i})\right\}\ i=1,2,\ldots,N \quad (4)$$

Temel çalışmada algoritmanın popülasyon sayısı, ST parametresi ve her ağaç için üretilecek tohum sayısı irdelenmiş ve popülasyon için 10 ilâ 40 arası, her ağaca ait tohum sayısının 1'den az olmamak kaydıyla popülasyon boyutunun yüzde 10'u ile yüzde 25'i arasında bir değer olması ve ST'nin de 0,1 olarak kullanılması ile makul sonuçlar elde edilebileceği önerilmiştir [4]. Şekil 1'de Ağaç Tohum Algoritması (TSA)'nın akış diyagramı görülmektedir.

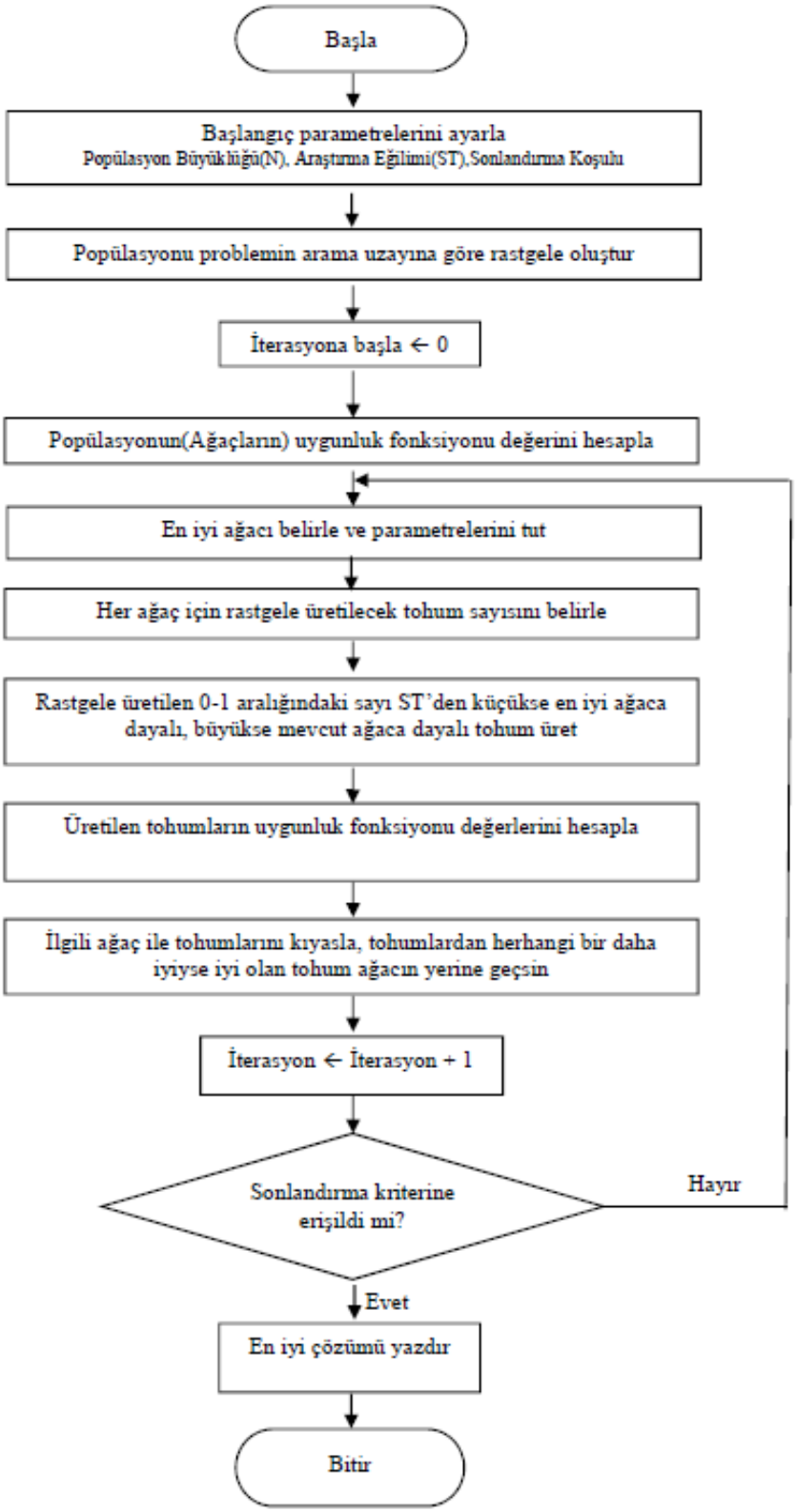
3. CUDA (COMPUTE UNİFİED DEVİCE ARCHİTECTURE)

CUDA grafik işlemcilerde çalışmak üzere oluşturulmuş bir mimari ve bir geliştirme ortamıdır. 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 donanımındaki işlem birimlerini 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 çekirdeğin istenilen kadar kopyasını çalıştırarak onu paralel hale getirir. 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 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 [35].

Global, paylaşımlı (shared) ve yerel (local) bellek olmak üzere üç farklı bellek türü tanımlanmıştır. Host (CPU), device (GPU), blok ve iş parçacıklarının belleklere erişimi farklı seviyelerde gerçekleşmektedir. Global bellek, ekran kartının RAM'idir. GPU'daki tüm iş parçacıkları tarafından



Şekil 1. Ağaç Tohum Algoritması(TSA)'nın Akış Diyagramı (Flow Chart of Tree-Seed Algorithm)

erişilebilmektedir. Erişim hızı nispeten yavaştır. Her bloğun okuma ve yazma yapılabildiği bu bellek merkezi işlem birimi ile grafik işlem birimi 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.

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 sabit ve doku 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. Paralel programlama için işlevler, ızgara (grid), blok (block) ve iş parçacığı (thread) olarak isimlendirilmiş üç yapılı bir sistem ile gerçekleştirilmektedir. İş parçacıkları 1, 2 veya 3 boyutlu bileşenler şeklinde blokları oluşturur. Bir boyutlu diziler veya iki boyutlu blok metrisleri ise ızgarayı meydana getirmektedir.

4. KIYAS FONKSİYONLARI (BENCHMARK FUNCTIONS)

Çalışmada test için literatürde sıklıkla kullanılan kıyas fonksiyonlarından olabildiğince farklı karakteristikte olanlar seçilmiştir. Ackley, Griewank, Rastrigin ve Schwefel bir çok yerel minimumu olan fonksiyonlardır. Sphere, Sum of Different Powers, Sum Squares fonksiyonları tek yerel minimumu olan fonksiyonlarındandır. Zakharov fonksiyonu levha (plaka) şekilli bir kıyas fonksiyonudur. Rosenbrock fonksiyonu vadi şekilli bir test fonksiyonudur. Styblinski-Tang fonksiyonu ise farklı karakteristiklere sahip bir kıyas fonksiyonudur. Kıyas fonksiyonları seri uygulama için Matlab ortamında yazılmıştır ve 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. Tablo 1'de

Tablo 1. Kıyas fonksiyonlarının arama aralığı ve formülleri (Search range and formulas of the benchmark functions)

Fonksiyonun İsmi (Function Name)	Arama Aralığı (Search Range)	Fonksiyonun Formülü (Function Formula)
Sphere	$-100<x_i<100$	$f(x)=\sum_{i=1}^{d} x_i^2$
Rosenbrock	$-2{,}048 \le x_i \le 2{,}048$	$f(x)=\sum_{i=1}^{d-1}[100(x_{i+1}-x_i^2)^2-(x_i-1)^2]$
Rastrigin	$-5{,}12 \le x_i \le 5{,}12$	$f(x)=10d+\sum_{i=1}^{d} x_i^2-10\cos(2\pi x_i)$
Griewank	$-600 \le x_i \le 600$	$f(x)=\sum_{i=1}^{d}\frac{x_i^2}{4000}-\prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$
Ackley	$-32{,}768 \le x_i \le 32{,}768$	$f(x)=-a\exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right)-\exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right)+a+\exp(1)$
Schwefel	$-500 \le x_i \le 500$	$f(x)=418{,}9829d-\sum_{i=1}^{d} x_i\sin(\sqrt{\lvert x_i\rvert})$
Sum Squares	$-10<x_i<10$	$f(x)=\sum_{i=1}^{d} ix_i^2$
Sum of Different Powers	$-1<x_i<1$	$f(x)=\sum_{i=1}^{d}\lvert x_i\rvert^{i+1}$
Zakharov	$-5<x_i<10$	$f(x)=\sum_{i=1}^{d}x_i^2+\left(\sum_{i=1}^{d}0{,}5ix_i\right)^2+\left(\sum_{i=1}^{d}0{,}5ix_i\right)^4$
Styblinski-Tang	$-5<x_i<5$	$f(x)=\frac{1}{2}\sum_{i=1}^{d}(x_i^4-16x_i^2+5x_i)$

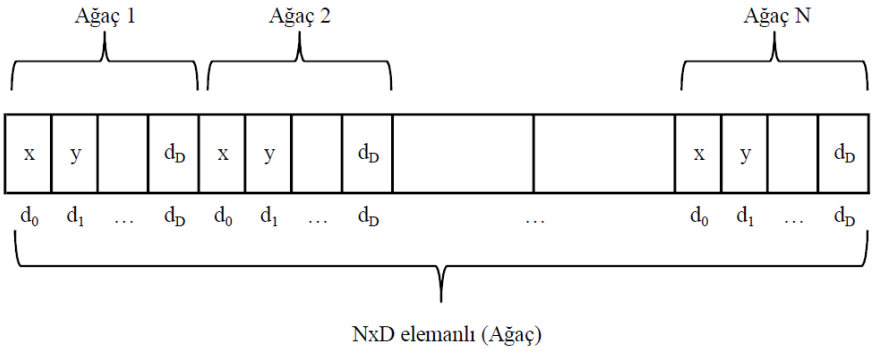
kullanılan kıyas fonksiyonlarının arama aralığı ve formülasyonları verilmiştir.

5. PARALELLEŞTİRME SÜRECİ (PARALLELIZATION PROCESS)

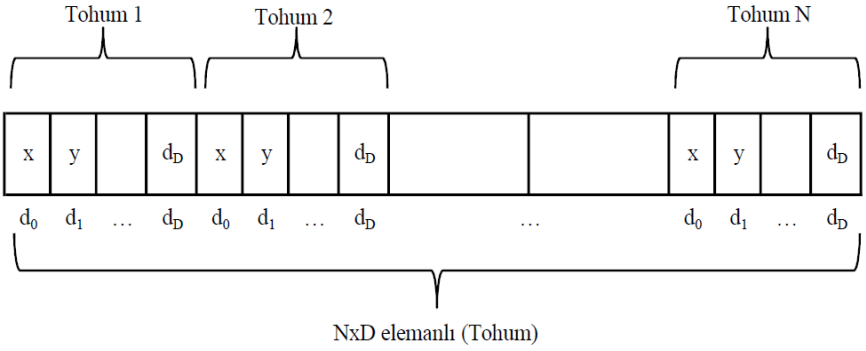
Ağaç-Tohum algoritmasının paralelleştirilmesi iki ana aşama olarak veri paralelliği (data parallelism) ve görev paralelliği (task parallelism) çerçevesinde uygulanmıştır. Veri paralelliğini sağlamak için algoritmada her adımda üretilerek kullanılan rastgele sayılar toplu halde GPU belleğine gönderilmiş, 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 değişkenlerin seri uygulamadaki bellek yerleşimleri Şekil 2 ve Şekil 3'te gösterilmiştir. Paralel uygulamada ise Şekil 4'te görüldüğü üzere bellekte önceden belirlenmiş alanlara yerleştirilmiş, böylelikle sabit noktalardaki verilere dayanarak sorunsuz bir şekilde eş zamanlı paralel işlemi yapılmıştır. Ağaçların amaç fonksiyon değerlerinin bellekte yerleşimi ile en iyi ağacın parametrelerinin bellekte yerleşimi hem seri hem de paralel uygulamada aynı olup satır bazlı sıralama (row major order) şeklindedir. Görev paralelliği çerçevesinde ise kıyas fonksiyonları, ağaçların oluşturulması, tohumların oluşturulması paralel olarak yeniden kodlanmış, adım adım işlenen süreçler eş zamanlı işlem görecek şekilde ayarlanmıştır. Tablo 2'de kullanılan kıyas fonksiyonlarından Sphere'ın seri ve paralel kodlanmış hali görülmektedir. Uygulama MATLAB R2014a ve Paralel Hesaplama Araç Kutusu (Parallel Computing Toolbox) kullanılarak Tablo 3'de CPU ve GPU'sunun teknik özellikleri verilen bilgisayarda çalıştırılmıştır.

6. SONUÇLAR VE TARTIŞMALAR (RESULTS AND DISCUSSIONS)

Ağaç-Tohum Algoritması seri ve paralel olarak kodlanmış, 10 farklı kıyas fonksiyonu (Sphere, Ackley, Griewank,



Şekil 2. Seri uygulamada ağaçların bellekte yerleşimi (Trees placement in memory at serial application)



Şekil 3. Seri uygulamada tohumların bellekte yerleşimi (Seeds placement in memory at serial application)

1.Ağaç					2.Ağaç					3.Ağaç					4.Ağaç					5.Ağaç				
d0	d1	d2	d3	d4	d0	d1	d2	d3	d4	d0	d1	d2	d3	d4	d0	d1	d2	d3	d4	d0	d1	d2	d3	d4

1.Ağacın 1.Tohumu					2.Ağacın 1.Tohumu					3.Ağacın 1.Tohumu					4.Ağacın 1.Tohumu					5.Ağacın 1.Tohumu				
d0	d1	d2	d3	d4	d0	d1	d2	d3	d4	d0	d1	d2	d3	d4	d0	d1	d2	d3	d4	d0	d1	d2	d3	d4

1.Ağacın 2.Tohumu					2.Ağacın 2.Tohumu					3.Ağacın 2.Tohumu					4.Ağacın 2.Tohumu					5.Ağacın 2.Tohumu				
d0	d1	d2	d3	d4	d0	d1	d2	d3	d4	d0	d1	d2	d3	d4	d0	d1	d2	d3	d4	d0	d1	d2	d3	d4

Şekil 4. Paralel uygulamada 5 ağaç ve 2'şer tohumunun bellekte yerleşimi (5 trees and 2 seeds placement in memory at parallel application)

Tablo 2. Sphere fonksiyonu için Seri ve Paralel kodlar (Serial and Parallel Codes for Sphere Function)

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

Seri Kod (Serial Code)	Ağaçlar için Paralel Kod (Parallel Code for Trees)	Tohumlar için Paralel Kod (Parallel Code for Seeds)

Tablo 3. Donanım ve yazılım yapılandırması (Hardware and software configurations)

Device	CPU	GPU
Processor	Intel® Xeon® Processor E5-2670 (2 adet)	GeForce GTX TITAN X
Number of cores	8 physical 16 thread	3072 CUDA Cores
Clocks	2.6 GHz (max 3.3 GHz)	1000 MHz (max 1075 MHz)
Memory	DDR3 800/1066/1333/1600	GDDR5
Memory size	32 GB	12 GB
Operating System	Windows 10	-
Compute Capability/ CUDA version	-	5.2 / 7.5

Rastrigin, Rosenbrock, Schwefel, Styblinski Tang, Sum of Different Powers, Sum Squares, Zakharov) için D (Boyut)=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 (paralelleştirme için sabitlenerek) çalıştırılmıştır. Durdurma kriteri olarak 20000 defa fonksiyon çağrı sayısı kullanılmıştır. Seri ve paralel uygulama birbirinden bağımsız şekilde 30'ar kere aynı şartlarda çalıştırılmıştır. Tablo 4'te 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. Bunun sebebi fonksiyon çağrı sayısı her popülasyon için aynı alındığında düşük boyutlu popülasyonlarda daha fazla çevrim yapılması, yüksek boyutlu popülasyonlarda daha az çevrim yapılmasıdır. Yani ardışıl çevrim sayısı azaltılmak kaydıyla toplam fonksiyon çağrı sayısı eşit tutulduğu takdirde paralelleştirmeden elde edilecek fayda maksimize edilebilmektedir. Elde edilen hızlanmanın verimliliği, hızlanma katsayısının ekran kartının sahip olduğu fiziksel çekirdek sayısına bölümüyle bulunur. Tablo 5'te kıyas fonksiyonlarının farklı ağaç sayıları ile ne kadar verimli bir şekilde çalıştığı görülmektedir. Ağaç sayısı arttıkça paralel uygulamanın daha verimli bir şekilde çalıştığı görülmektedir.

6.1. Karşılaştırmalar (Comparısons)

Paralelleştirme işleminin esas amacı, seri algoritmanın veri ve görev paralelliği kapsamında analiz edilmesi ve yeniden tasarlanması ve paralel programlama ortamında kodlanmasıdır. Algoritmanın GPGPU üzerinde uygulanmasının esas amaç ise GPU kartlarının üzerinde bulunan çok sayıdaki işlemci çekirdeklerini kullanarak hızlanma elde edilmesi olduğu göz önüne alındığında karşılaştırma işleminin yukarıda yaptığımız şekilde seri uygulamayla kıyaslanmış olması yeterli görünmektedir. Aynı hesaplama işleminin seri veya paralel donanım üzerinden gerçekleştirmenin sonucu değiştirmeyeceği kesindir. Veri ve görev paralel şekilde uygulanmış algoritma ile hızlanma ortaya konulabiliyorsa çalışma başarılı olarak değerlendirilmektedir. Bu yüzden uygulama sonuçlarının farklı optimizasyon algoritmaları ile karşılaştırılması bu çalışmanın konusu olmamakla birlikte çalışmada paralelleştirilen Ağaç-Tohum Algoritması, [4] numaralı çalışmada Yapay Arı Kolonisi (ABC), Parçacık Sürü Optimizasyonu (PSO), Harmoni Arama (HS), Ateşböceği (FA) ve Yarasa (BA) algoritmaları ile kıyaslanmıştır. Bundan dolayı yöntemin optimizasyon hatasının veya

Tablo 4. Algoritmanın kıyas fonksiyonları üzerinde ağaç sayısına göre hızlanma katsayıları (Acceleration coefficients of the algorithm on benchmark functions based on stand size)

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

Tablo 5. Algoritmanın kıyas fonksiyonları üzerindeki ağaç sayılarına göre verimlilik oranları (Efficiency rates of the algorithm on benchmark functions according to stand size)

Fonksiyon / Ağaç Sayısı	10	20	30	40	50	60	70	80	90	100
Sphere	0,10	0,29	0,43	0,55	0,66	0,80	0,99	1,13	1,29	1,43
Sum Squares	0,09	0,27	0,39	0,60	0,70	0,83	1,01	1,16	1,31	1,45
Zakharov	0,10	0,32	0,57	0,82	1,05	1,23	1,43	1,63	1,77	1,94
Sum of Different Powers	0,13	0,43	0,85	1,31	1,77	1,59	2,18	2,42	2,71	2,89
Ackley	0,13	0,46	0,94	1,47	2,09	2,62	3,35	3,96	4,55	5,13
Rastrigin	0,13	0,48	0,99	1,62	2,37	3,09	3,71	4,59	5,27	6,17
Griewank	0,12	0,48	1,01	1,63	2,43	3,23	3,90	4,70	5,78	6,23
Rosenbrock	0,13	0,48	1,02	1,70	2,50	3,26	3,91	4,69	5,49	6,23
Styblinski Tang	0,13	0,50	1,05	1,74	2,57	3,38	4,05	4,85	5,82	6,64
Schwefel	0,12	0,54	1,09	1,72	3,29	3,63	3,77	5,50	6,49	7,69

başarısının yeniden karşılaştırılmasına bu çalışmada ihtiyaç duyulmamıştır. Hızlanma karşılaştırması yapmak istersek iki ayrı durum ortaya çıkmaktadır. Birinci durumda, diğer çalışmalardaki donanım ortamlarının oluşturulmasıyla bu çalışmadaki yöntem, bu donanım ortamında çalıştırılmalıdır ki bu maliyet imkânı açısından bizim için mümkün değildir. Diğer seçenek ise diğer yöntemlerin bizim oluşturduğumuz donanım ortamında yeniden uygulanmasına dayanır. Buradaki problem ise yöntemlerin programcı yeteneklerine aynen riayet edilerek yeniden kodlanmasının zorluğudur. Literatürdeki diğer çalışmalar [8-11, 13, 14, 17, 18, 20, 23, 26, 28-30, 33, 36] incelendiğinde her biri farklı GPU kartları ile farklı fonksiyon değerlendirme sayıları ile farklı test fonksiyonları için yapılmış çalışmalar bulunmaktadır. Her biri tek bir sürü zekâsı veya evrimsel hesaplama algoritmasının seri uygulanmasını paralel hale getirmiş ve hızlanma analizi sadece yöntem üzerinde yapmıştır ve diğer yöntemlerle hem başarı hem hızlanma açısından karşılaştırma yoluna gidilmemiştir. Literatürün bu açıdan incelenmesi neticesinde bu çalışmada da Ağaç-Tohum Algoritmasının paralel versiyonu geliştirilmiş ve farklı test fonksiyonları üzerinde hızlanma açısından seri ve paralel versiyonların karşılaştırılması yapılmıştır.

6.2. Blok Sayısının İşlem Süresine Etkisinin Analizi
(Analysis of Block Size Effect on Process Time)

CUDA iş parçacıkları blok ismi verilen ve SM'ler üzerinde paralel olarak çalışan yapılar içerisinde bulunmaktadır. Bu yüzden blok sayısının seçimi, işlem sürelerini etkilemektedir. Kullanılan donanıma bağlı olarak kullanılan blok sayısının etkisi değişmektedir. Çalışmamızda kullandığımız ekran kartı 24 SM'ye sahiptir. Deneysel çalışmalar sırasında blok sayısı ağaç sayısına eşit olarak alınmıştır. Blok sayısının işlem sürelerine etkisini göstermek için Ağaç sayısı 50 ve blok sayıları 2, 4, 8, 16, 32, 64, 128, 256 alınarak 30'ar kere çalıştırılmış ve blok sayısının kıyas fonksiyonlara göre etkileri Tablo 6'da gösterilmiştir. Sphere, Sum Squares, Zakharov, Sum of Different Powers kıyas fonksiyonları için ağaç sayısından daha az sayıda blok ile işlem yapıldığı zaman hızlanma için herhangi katkı sağlamadığı, ağaç sayısından fazla blok sayısı

Tablo 6. Kıyas fonksiyonları için blok sayılarının işlem süresine etkisi
(Effect of block size for processing time on benchmark functions)

	Blok=2	Blok=4	Blok=8	Blok=16	Blok=32	Blok=64	Blok=128	Blok=256
Sphere	2,56	2,52	2,51	2,53	2,53	9,57	9,56	9,56
Sum Squares	2,66	2,59	2,58	2,57	2,59	9,75	9,78	9,76
Zakharov	2,58	2,58	2,58	2,56	2,58	6,38	6,41	6,42
Sum of Different Powers	2,54	2,55	2,55	2,56	2,56	4,22	4,19	4,19
Ackley	2,58	2,56	2,57	2,58	2,57	3,28	3,26	3,27
Rastrigin	2,58	2,55	2,56	2,57	2,57	2,87	2,91	2,91
Griewank	2,62	2,59	2,60	2,60	2,61	2,83	2,86	2,86
Rosenbrock	2,55	2,55	2,54	2,56	2,54	2,73	2,75	2,74
Styblinski Tang	2,56	2,58	2,70	2,74	2,75	2,77	2,78	2,79
Schwefel	2,68	2,60	2,62	2,65	2,61	2,69	2,70	2,71

kullanıldığında ise performansın düştüğü görülmektedir. Ackley, Rastrigin, Griewank, Rosenbrock, Styblinski Tang ve Schwefel fonksiyonları için farklı blok sayılarının kullanılması işlem süresini önemli ölçüde etkilememiştir. Bu sonuçlar blok sayısı olarak ağaç sayısının alınmasının uygun olduğunu göstermektedir.

7. SONUÇLAR (CONCLUSIONS)

Bu makalede doğa esinli metasezgisel optimizasyon algoritmalarının en yenilerinden birisi olan Ağaç-Tohum algoritması (TSA) CUDA paralel programlama altyapısına uygun olarak yeniden kodlanarak paralel uygulamanın zaman açısından daha verimli olduğu ortaya koyulmuştur.

Çalışmada kullanılan kıyas fonksiyonları olabildiğince farklı seçilerek performansın dağılımı gözlemlenmiştir. Ağaç sayılarının 100 alındığı paralel uygulamanın sonuçlarına göre birçok 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 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. Fonksiyonlar incelendiğinde görülür ki fonksiyonun bir defa hesaplaması için ayrılan süre ne kadar fazla olursa paralelleştirmeden elde edilecek kazanımda o kadar fazla olmaktadır. 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 daha hızlı çözülmesi için önerdiğimiz CUDA tabanlı uygulamanın daha kullanılabilir olduğunu ortaya koymaktadır. 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 kaydedilmiş, olağan dışı bir durum gözlenmediğinden makalemize eklenmemiştir. Rosenbrock ve Schwefel kıyas fonksiyonları hariç diğer 8 kıyas fonksiyonunda bir çok defa global minimuma erişilmiş, erişilemediği durumda da en uygun değere yakın 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 [4]'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 [4]'ın çalışmasında Schwefel fonksiyonu için ilgili çalışmada kıyaslanan algoritmalarında farklı farklı değerler bulması bu durumun normal olarak değerlendirilebileceğini anlatmaktadır.

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 üstesinden gelinmesine zemin hazırlamaktadır. Bu doğrultuda akademik camiada yaygın olarak kullanılan MATLAB ortamı ile oldukça kolaylaştıran kod yazımı ve Paralel Hesaplama Araç Kutusu (Parallel Computing Toolbox) araştırmacıların ve uygulayıcıların işlerini oldukça kolaylaştırmaktadır. CUDA tabanlı paralel hesaplamanın büyük veri analizinde ve çözümü uzun zaman alan gerçek dünya problemlerinin çözümünde oldukça yararlı araçlar olacağı öngörülmektedir.

TEŞEKKÜR (ACKNOWLEDGEMENT)

Test ortamının oluşturulması için gereken donanım Selçuk Üniversitesi Bilimsel Araştırma Projeleri Koordinatörlüğü

tarafından desteklenen 15401121 numaralı Araştırma Projesi kapsamında alınabilmiştir. Yazarlar bu katkısından dolayı Selçuk Üniversitesi Bilimsel Araştırma Projeleri Koordinatörlüğü'ne teşekkür eder. Ayrıca bu çalışmanın erken bir versiyonu "Selçuk International Conference on Applied Sciences" konferansında sunulmuştur. Yazarlar, bu sunum ve sonrasında eleştirileri ile yapıcı katkılar sunan ve makale haline gelmesinde teşvik eden meslektaşlarına teşekkürü bir borç bilir. Bildirinin aldığı eleştiriler kapsamında test fonksiyonu sayısı iki katına çıkarılmıştır. Analizler ve sunumlar daha detaylı ve kapsayıcı hale getirilmiştir. Hızlanma ile fonksiyon karakteristikleri ve popülasyon boyutu arasındaki ilişkiler detaylı olarak verilmiştir. Makalenin literatür özeti daha detaylı olarak ele alınmış ve sempozyum bildirisine göre oldukça genişletilmiştir.

KAYNAKLAR (REFERENCES)

1. Cinar A.C., A Cuda-based Parallel Programming Approach to Tree-Seed Algorithm, MSc Thesis, Selçuk University, Graduate School of Natural Sciences, Konya, 2016.
2. Akyol S., Alataş B., Automatic mining of accurate and comprehensible numerical classification rules with cat swarm optimization algorithm, Journal of the Faculty of Engineering and Architecture of Gazi University, 31 (4), 839-857, 2016.
3. Haklı H., Sürekli Fonksiyonların Optimizasyonu için Doğa Esinli Algoritmaların Geliştirilmesi, Yüksek Lisans Tezi, Selçuk Üniversitesi Fen Bilimleri Enstitüsü, Konya, 2013.
4. Kiran M.S., TSA: Tree-seed algorithm for continuous optimization, Expert Systems with Applications, 42 (19), 6686-6698, 2015.
5. Kıran M.S., An Implementation of Tree-Seed Algorithm (TSA) for Constrained Optimization, Intelligent and Evolutionary Systems, Springer, 189-197, 2016.
6. Muneeswaran V., Rajasekaran, M.P., Performance evaluation of radial basis function networks based on tree seed algorithm, International Conference on Circuit, Power and Computing Technologies (ICCPCT), 2016.
7. Nvidia Compute unified device architecture programming guide, 2007.
8. Mussi, L., Cagnoni S., Particle swarm optimization within the CUDA architecture, Viale G. Usberti 181a, I-43124 Parma, Italy, 2009.
9. Molnár F., Szakaly T., Meszaros R., Lagzi I., Air pollution modelling using a Graphics Processing Unit with CUDA, Computer Physics Communications, 181 (1), 105-112, 2010.
10. Mussi L., Daolio F., Cagnoni S., Evaluation of parallel particle swarm optimization algorithms within the CUDA™ architecture, Information Sciences, 181 (20), 4642-4657, 2011.
11. Solomon S., Thulasiraman P., Thulasiram R.Collaborative multi-swarm PSO for task matching using graphics processing units, Proceedings of the 13th annual conference on Genetic and evolutionary computation, 2011.
12. Zhang Z., Seah H.S., CUDA acceleration of 3D dynamic scene reconstruction and 3D motion estimation for motion capture, IEEE 18th International Conference on Parallel and Distributed Systems (ICPADS), 2012.
13. Platos J., Snasel V., Jezowicz T., Kromer P., Abraham, A.A PSO-based document classification algorithm accelerated by the CUDA platform, IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2012.
14. Kumar J., Singh L., Paul S.GPU based parallel cooperative particle swarm optimization using C-CUDA: a case study, IEEE International Conference on Fuzzy Systems (FUZZ), 2013.
15. Wang H., Rahnamayan S., Wu Z., Parallel differential evolution with self-adapting control parameters and generalized opposition-based learning for solving high-dimensional optimization problems, Journal of Parallel and Distributed Computing, 73 (1), 62-73, 2013.
16. Wang H., Wu Z., Rahnamayan S., Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems, Soft Computing, 15 (11), 2127-2140, 2011.
17. Kneusel R., Curve-Fitting on Graphics Processors Using Particle Swarm Optimization, International Journal of Computational Intelligence Systems, 7 (2), 213-224, 2014.
18. Luo G.-H., Huang S.-K., Chang Y.-S., Yuan S.-M., A parallel Bees Algorithm implementation on GPU, Journal of Systems Architecture, 60 (3), 271-279, 2014.
19. Janousešek J., Gajdoš P., Radecký M., Snášel, V., Classification via Nearest Prototype Classifier Utilizing Artificial Bee Colony on CUDA, International Joint Conference SOCO'14-CISIS'14-ICEUTE'14, 2014.
20. Rymut B., Kwolek B., Real-time multiview human pose tracking using graphics processing unit-accelerated particle swarm optimization, Concurrency and Computation: Practice and Experience, 27 (6), 1551-1563, 2015.
21. Yuan H., Zhao T., Yang W., Pan H., 1821. Annealing evolutionary parallel algorithm analysis of optimization arrangement on mistuned blades with non-linear friction, Journal of Vibroengineering, 17 (8),4078-4095, 2015.
22. Bukharov O.E., Bogolyubov D.P., Development of a decision support system based on neural networks and a genetic algorithm, Expert Systems with Applications, 42 (15), 6177-6183, 2015.
23. Wang P., Li H., Zhang B., A GPU-based Parallel Ant Colony Algorithm for Scientific Workflow Scheduling, International Journal of Grid and Distributed Computing, 8 (4), 37-46, 2015.
24. Kai Z., Ming Q., Lin L., Xiaoming, L., Solving Graph Coloring Problem by Parallel Genetic Algorithm Using Compute Unified Device Architecture, Journal of Computational and Theoretical Nanoscience, 12 (7), 1201-1205, 2015.

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

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

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

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

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

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

31. Bukata L., Šůcha P. Hanzálek Z., 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, 2015.

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

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

34. Cinar A.C. ve Kiran M.S., A Parallel Version of Tree-Seed Algorithm (TSA) within CUDA Platform, Selçuk International Scientific Conference On Applied Sciences, Antalya, 2016.

35. Nvidia CUDA C PROGRAMMING GUIDE, 2016.

36. Zarrabi A., Karuppiah E.K., Kok Y.K., Hai, N.C., See, S., Gravitational Search Algorithm Using CUDA, IEEE 15th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2014.