T.C.

İNÖNÜ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

ÇOK ÇEKİRDEKLİ MİMARİLERDE PARALEL PROGRAMLAMA İLE GENETİK ALGORİTMALARIN UYGULAMASI

Kenan İNCE

YÜKSEK LİSANS

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

Ocak 2013

MALATYA

Tezin Başlığı : Çok Çekirdekli Mimarilerde Paralel Programlama ile Genetik Algoritmaların Uygulaması

Tezi Hazırlayan : Kenan İnce

Sınav Tarihi : 07.01.2013

Yukarıda adı geçen tez jürimizce değerlendirilerek Bilisayar Mühendisliği Ana Bilim Dalında Yüksek Lisans Tezi olarak kabul edilmiştir.

Sınav Jürisi Üyeleri

Doç. Dr. Ali Karcı (Jüri Başkanı, Danışman)

İnönü Üniversitesi

Yrd. Doç. Dr. M. Fatih Talu (Üye)

İnönü Üniversitesi

Yrd. Doç. Dr. N. Murat Yağmurlu (Üye)

İnönü Üniversitesi

İnönü Üniversitesi Fen Bilimleri Enstitüsü Onayı

Prof. Dr. Mehmet Alpaslan

Enstitü Müdürü

ONUR SÖZÜ

Yüksek Lisans Tezi olarak sunduğum "Çok Çekirdekli Mimarilerde, Paralel Programlama ile Genetik Algoritmaların Uygulaması" başlıklı bu çalışmanın bilimsel ahlak ve geleneklere aykırı düşecek bir yardıma başvurmaksızın tarafımdan yazıldığını ve yararlandığım bütün kaynakların, hem metin içinde hem de kaynakça yöntemine uygun biçimde gösterilenlerden oluştuğunu belirtir, bunu onurumla doğrularım.

Kenan İNCE

ÖZET

Yüksek Lisans Tezi

ÇOK ÇEKİRDEKLİ MİMARİLERDE, PARALEL PROGRAMLAMA İLE GENETİK ALGORİTMALARIN UYGULAMASI

Kenan İnce

İnönü Üniversitesi

Fen Bilimleri Enstitüsü

Bilgisayar Mühendisliği Anabilim Dalı

66+x sayfa

2012

Danışman: Doç. Dr. Ali Karcı

Bu tezde, teknolojik gelişmelerin hızlı yükselişi sayesinde günlük kullandığımız bilgisayarlarda bile yaygınlaşan çok çekirdekli bilgisayarlarda, paralel programlama teknikleri kullanarak elde edilebilecek hız artışının gösterilmesi amaçlanmıştır.

Bugün, cep telefonlarında dahi çok çekirdekli mimariler kullanılmaktadır. Ancak, mevcut olan yazılım ve yazılım alt yapıları bu imkanı kullanmamaktadır. Yazılımlar halen tek kanal üzerinde çalışmakta, ancak işletim sistemlerinin, çalışan işlemlere çalışma süresince farklı kanallar tahsis etmesi sayesinde, devam eden bir işlem birden fazla kanalda işlem görebilir. Doğal olarak işletim sisteminin desteklemesi sayesinde, birden fazla işlem, her işlem tek kanalda çalışmak üzere birden fazla kanal kullanabilmektedir. Bu tezde, bir optimizasyon algoritması olan genetik algoritma kullanılarak, işletim sisteminin yönetiminden bağımsız, bilgisayardaki mevcut bütün çekirdeklerin kullanılması temeline dayanan bir uygulama geliştirilmiştir.

Çalışmamızın sonucunda, normal bir bilgisayarda bile paralel programlama sonucunda kayda değer performans artışı gözlenmiştir.

ANAHTAR KELİMELER: Genetik algoritma, paralel programlama, çok çekirdekli mimari

ABSTRACT
Master Thesis

IMPLEMENTATION OF GENETIC ALGORITHM ON MULTI-CORE ARCHITECTURES WITH PARALLEL PROGRAMMING

Kenan İnce

İnönü University

Graduate School of Natural and Applied Sciences

Department of Computer Engineering

66+x pages

2012

Supervisor: Assoc. Prof. Dr. Ali Karcı

In this thesis, it is aimed to show the increase in speed-up of multi-core PCs, which are widely used even in the daily used computers thanks to rapid technological developments by parallel programming techniques.

Today, multi-core architectures are used even in cell phones. However, avaliable software and hardware intrastracture does not use this opportunies. Software still run on single thread but a life cycle can run on multiple threads due to the fact that operating systems allocate different threads to the running process. Thus naturally multiple software threads can run on multiple threads but each are by a single thread at a time. In this thesis, an application independent of operating system and based on the use off all avaliable cores in the computer by using genetic algorithms, one of the optimization algorithms, has been developed.

In the end of our study, a remarkable speed-up by parallel programming even on a ordinary computer has been observed.

KEYWORDS : Genetic algorithm, parallel programming, multi-core

TEŞEKKÜR

Bu çalışmanın her aşamasında yardım, öneri ve desteğini esirgemeden beni yönlendiren danışman hocam Sayın Doç. Dr. Ali Karcı'ya;

Uygulama geliştirme aşamasında, her türlü fikri ve yazılım olarak yardım ve desteğini esirgememiş olan Bilgisayar Bilimleri Anabilim Dalı öğretim üyelerinden Yrd. Doç. Dr. M. Fatih Talu'ya;

Tezin hazırlanması ve düzenlenmesi aşamalarında yardımını esirgemeyen Matematik Bölümü öğretim üyelerinden Yrd. Doç. Dr. Murat Yağmurlu'ya;

Tez yazım aşamasında, test sistemi olarak bilgisayarını kullandığım İnşaat Mühendisliği araştırma görevlilerinden Arş. Grv. Talha Sarıcı'ya;

Tez yazım aşamasında, kriter fonksiyonlarının hazırlanmasında yardımcı olan Bilgisayar Mühendisliği Bölümü araştırma görevlilerinden Arş. Grv. A. Erhan Akkaya'ya

Teşekkür ederim.

TEŞEKKÜR

Tezin uygulama aşamasında, 2012-198 numaralı yüksek lisans projesi kapsamında, vermiş oldukları maddi ve manevi destekten dolayı, İnönü Üniversitesi Bilimsel Araştırma Projeleri Koordinasyon Birimine

Teşekkür ederim.

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

1. GİRİŞ

Günümüz bilgisayarları, genellikle birden fazla işlemciye sahiptir. Bazı mimarilerde birden fazla ayrık işlemci varken, bazı mimarilerde de tek işlemci içinde birden fazla çekirdek bulunmaktadır. Bazı sunucu sistemleri veya çalışma istasyonları gibi sistemlerde ise bu iki sınıfın birleşimi yani birden fazla çok çekirdek barındıran işlemci bulundurmaktadırlar.

Günlük bilgisayarlarımızda hatta cep telefonlarımızda yaygın olarak kullanılan mimari tek işlemcili ancak çok çekirdekli mimaridir. Bu mimari Simetrik Çok İşlem Ünitesi (Symmetric Multiprocessing - SMP) mimarisidir. Bugün yaygın olarak kullanılan bilgisayarların ve cep telefonlarının işlemcisi bu sınıftaki işlemciler olduğundan dolayı, bu tezde bu sınıftaki işlemcileri barındıran test sistemleri kullanılmıştır.

Donanım alanındaki hızlı gelişmeye rağmen, yazılım alanındaki gelişmeler bu denli hızlı olmamaktadır. Bugün halen bu donanım gücünü tam olarak kullanabilen yazılımlar geliştirilmemektedir. Bu donanım gelişmesine karşılık verebilecek yazılım tekniklerinden bir tanesi ve belki de en çok gelecek vaat edeni paralel programlamadır.

Paralel programlama, birçok işlemin eş zamanlı olarak yürütülmesi olarak özetlenebilir. Temel olarak, büyük bir problemin, eş zamanlı çözülecek küçük parçalara bölünerek çözülmesi prensibine dayanır. Bu ayrılan parçaların boyutuna göre, küçük parçacıklı paralellik (fine-grain parallelism) veya büyük parçalı (coarse-grain parallelism) paralellik teknikleri bulunmaktadır. Bu tezde kullanılan büyük parçalı paralelleştirme tekniğidir.

Paralel programlama teknikleri olarak şunları sıralayabiliriz;

- Bit seviyesinde paralelleştirme.
- Komut seviyesinde paralelleştirme.
- Veri seviyesinde paralelleştirme.
- Görev seviyesinde paralelleştirme.

Paralel programlamanın geliştirilme amacı, verimli kaynak kullanılmasıdır. Şöyle ki, büyük bir problemin daha kısa sürede çözülmesi iki farklı şekilde mümkündür. Birincisi, işlemcinin belli bir sürede yaptığı işlem sayısını arttırmaktır ki bu saat döngüsü (Clock Cycle) olarak ifade edilir. Saat döngüsünün arttırılması teorik olarak en hızlı ve en kısa yoldur. Örneğin bir adet 3 GHZ işlemci barındıran sistem, iki adet 1.5 GHZ işlemci barındıran sistemden, bir problemin çözümünü daha hızlı elde etmektedir. Çünkü paralel programlamada, problemin parçalanması ve tekrar çözüm parçacıklarının birleştirilmesi için geçen ek işlem süreleri vardır. Ancak tek çekirdeğin saat hızının, teknolojinin gelişmesi ile artması ile birlikte, fiziksel sınırları bulunmaktadır. Çünkü saat hızının arttırılması ile birlikte işlemcilerin güç gereksinimi artar ve fazla güç tüketen işlemcinin soğutulması sorunu ortaya çıkar.

İkinci teknik ise paralel programlamadır. Bu teknikte işlem parçalanarak çözüldüğünden fazla işlemciye gereksinim duyar ancak güç tüketimi ve bunun sonucu ortaya çıkan ısı problemi yoktur. Ek olarak, çözülmek istenen problemin büyüklüğüne göre maliyet artabilmekle birlikte, fiziksel sınırlar günümüz için, seri çözüm metodundan daha azdır. Örneğin 2011 yılının, işlemci saat döngüsünün Guinness rekoru 8.429 GHZ'dir (Guinness, 2011). Tek işlemci ile bunun anlamı saniyede 8 milyar döngü demektir ve bu işlemcinin işlem gücünü, bugün kullanılan çok çekirdekli mimari teknolojisi ile, 2.4 GHZ hızda çalışan dört çekirdekli bir işlemci kullanılarak geçilebilmektedir. 8 GHZ'in üzerine ancak özel soğutma sistemleri ile laboratuvar ortamlarında çıkılabilirken, 2.4 GHZ hızına bugün kullandığımız bir çok bilgisayar, dâhili olarak çıkabilmektedir. Bu da günümüz teknolojisi ele alındığında, paralel programlamanın önemini göstermektedir.

Yukarıda anlatılan avantajların yanında, paralel programlamanın bazı zorlukları ve gereksinimleri vardır. Öncelikle veri bağımlılığı olmaması gerekmektedir. Hiç bir çözüm, birbirine bağımlı en uzun çözüm süresinden kısa olamaz. Doğal olarak, paralelleştirilmek istenen problem ya da algoritmanın yapısı, elde edilebilecek performans artışı açısından önem arz etmektedir.

Bir algoritmanın ya da problemin veri bağımlılığı olması demek, bir eylemin sonuca erişebilmesi için başka bir eylemin sonucuna ihtiyaç duymasıdır. Doğal olarak diğer işlemin bitmesini beklemesi gerekmektedir ki bu da paralel programlamada istenmeyen bir durumdur. Daha da önemli olarak eğer bu iki işlemin her biri diğerinin

üreteceği sonuca ihtiyaç duymasıdır ki bu durum kilitlenme (deadlock) olarak tabir edilir. İki ya da daha fazla eylemin devam etmek için birbirlerinin bitmesini beklemesi ve sonuçta ikisinin de devam edememesi durumudur. Böyle durumlar, eğer problemin tamamında varsa paralelleştirme yapılamamaktadır. Bu tarz problemler ayrıştırılamaz problemler olarak tanımlanmaktadır. Ancak sadece bir kısmında varsa, problemin geri kalanı paralelleştirilebilir ancak bir kısmı seri çalışacağından, paralel programlamadan istenilen verim alınamaz hatta seri bölümler çok fazla ise, paralel programlamanın kaçınılmaz durumlarından biri olan veri parçalama ve sonuç toplama süreleri ile birlikte, seri işlem süresi paralel işlem süresinden daha kısa bile olabilir. Bu durum, bir algoritma ya da problemin paralelleştirilebilmesi için ya bağımsız çözüm kümeleri üzerinde çalışması gerektiğini ya da parçalanabilir olması gerektiğini göstermektedir. Bir problemin, tamamı ayrıştırılabilir olabileceği gibi, kısmen bazı bölümleri ayrıştırılabilir işlemlerden oluşmuşta olabilir. Genetik algoritma, tamamen birbirinden bağımsız çözümlere ayrıştırılabilir bir algoritmadır. Genetik algoritmanın bu özelliğinden dolayı, paralelleştirmeye uygun bir algoritmadır.

Genetik algoritma (Goldberg 1989, Holland 1975, Rechenberg 1973), doğal gelişim sürecini taklit eden gelişim algoritmalardan bir tanesidir. Bağımsız çözüm uzayına, genetik operatörlerin uygulanması ile bu çözümleri optimize ederek kabul edilebilir çözümü bulmaya çalışır. Genetik algoritma, birbirinden bağımsız ve çoğunlukla rastgele oluşturulmuş çözüm uzayında çalıştığından dolayı, paralelleştirmeye uygun bir algoritmadır. Algoritmada, çözümler kromozom olarak ifade edilir ve en iyi kromozom bulunmaya çalışılır. Genetik algoritmanın temeli, uygulanan genetik operatörlerdir. Genetik operatörler temel olarak üç tanedir;

- Seçim
- Çaprazlama
- Mutasyon

Seçim, çözüm popülâsyonundan, seçim kriterlerine göre, ebeveyn çözümlerin seçilmesidir. Ebeveynlerin mevcut çözümler içinde en iyilerden seçilebileceği gibi, rastgele de seçilebilir. Bu da uygulanan probleme göre değişiklik gösterebilmektedir.

Çaprazlama, ebeveyn olarak seçilen iki kromozomun, kendi aralarında farklı sayı ve noktalardan bölünerek elde edilen küçük gen parçalarının karşılıklı olarak değiştirilmesi sonucu yeni nesillerin elde edilmesi olarak özetlenebilir. Çaprazlama,

çözüm uzayının farklılaşmasını sağlayarak, optimum çözüme ulaşmada en önemli adımdır.

Mutasyon, alınan bir kromozomun, herhangi bir genini, belirlenen bir mutasyon fonksiyonuna tabi tutarak bu genin değiştirilmesi prensibine dayanır. Örneğin, ikili tabanda oluşturulmuş bir kromozomun, rastgele seçilen bir geninin NOT operatörüne tabi tutulması en temel mutasyon örneklerinden bir tanesidir.

Uygulamalı matematikte, optimizasyon algoritmalarının karakteristiklerinin belirlenmesinde kriter (benchmark) fonksiyonları kullanılmaktadır. Bu tezde, genetik algoritma için uygunluk fonksiyonu olarak farklı kriter fonksiyonları kullanılmış ve bunların seri ve paralel çalıştırılması arasındaki farklara yer verilmiştir.

Paralel programlama için farklı kütüphane çözümleri bulunmaktadır. Bunlardan en çok bilinen kütüphaneler;

- OpenMP (Open Multiprocessing)
- MPI (Message Passing Interface)
- NVIDIA CUDA
- Intel TBB (Intel Threading Building Blocks)

OpenMP, C, C++ ve Fortran dilleri üzerine yazılmış, var olan birçok işlemci mimarisi üzerinde çalışabilen paralel programlama API'sidir (Application Programming Interface). Birçok işletim sisteminde desteği bulunmaktadır. Açık kaynak kodlu bir kütüphanedir.

MPI, paralel bilgisayarlar üzerinde uygulama geliştirmek için geliştirilmiş bir taşınabilir mesaj iletim kütüphanesidir. Fortran 77 ve C programlama dilleri kullanılarak taşınabilir uygulamalar yazılabilir.

CUDA, NVIDIA tarafından geliştirilen paralel programlama platformudur. Ancak grafik işlemcisinin kullanılarak paralel uygulanması prensibine dayanmaktadır.

Intel TBB, çok çekirdek mimarisinde optimal seviyede performans elde etmek için gerekli olan alt seviye kanal (thread) detaylarını soyutlayan bir çalışma zamanı kütüphanesidir. C++ programlama dilini desteklemektedir. Yaygın kullanılan bütün işletim sistemlerinde desteği bulunmaktadır.

Bu tezde geliştirilen uygulamada Intel TBB kullanılmıştır. Bu kütüphanenin seçilmesindeki en önemli etken, programlamada kontrolü zor olan bazı karmaşıklıkların, kütüphanenin sunduğu bazı yapı ve algoritmalarla aşılmış olmasıdır. Kütüphane, her işlem parçasının, görev olarak algılanmasını sağlamaktadır ve bu sayede her çekirdeğe dinamik olarak ayrılmasını sağlamaktadır. Ayrıca kütüphane, algoritmaya bağlı olarak bağımlılık ağacını oluşturur, senkronize eder ve gerektiğinde ortadan kaldırır. Böylece kilitlenme gibi paralel programlamanın en büyük zorluklarından birinin aşılmasını sağlamış olmaktadır.

Bu tezdeki amacımız, düşük maliyetli bilgisayarlarda dahi, paralel programlama teknikleri kullanıldığı takdirde, elde edilebilecek performans artışının gösterilmesidir. Bu sebeple test sistemleri piyasada mevcut olan, dört farklı maliyet aralığında seçilmiş bilgisayarlardır.

Genetik algoritmanın doğasından ötürü paralleştirmeye uygun bir algoritma olduğundan bahsetmiştik. Bu alanda bir çok çalışma yapılmıştır. Ancak yapılan araştırmalar, genellikle küme sistemlerde veya grafik işlem ünitesi kullanılarak geliştirilmiş paralel çalışmalardır. Bu çalışmalar farklı algoritmaların harmanlanması ile veya farklı mimarilerde yapılmıştır.

Mohamed Wahib Wahib, Masaharu Munetomo ve Kiyoshi Munawar (Wahib vd. 2011) 2011 yılında yaptıkları çalışmada, nVidia GPU üzerinde çalışılan paralel genetik algoritmaların optimizasyonu üzerine bir çalışma yağmışlardır. Çalışmada, yukarıda da belirtildiği gibi, yapılmış çalışmalar açısından yeni bir bakış açısı olabilecek bir yaklaşım sergilenmiştir ve literatürdeki benzer çalışmalarla kıyaslamalar yapılmıştır.

Yine bu alandaki bir diğer çalışma, Jianming Li, Ximeng Lv ve Linlin Liu (Li v.d. 2011) tarafından yine 2011 yılı içerisinde yapılmış bir çalışmadır. Bu çalışmada yapılan, acil lojistikte büyük bir önem arz eden, araç yönlendirme işlemini hızlı ve etkin bir şekilde yapabilmek için, grafik işlemci kullanılarak genetik algoritma ile paralel yapılmış bir çözüm önerilmektedir. Bu çalışmada, her kromozomun grafik işlem ünitesinin bir işlemcisine gönderilerek paralelleştirme önerilmektedir ve normal işlemci ile yapılmış çalışmalarla performans olarak kıyaslaması verilmektedir (Li vd. 2011).

Bir diğer çalışma, filogenetik ağaçlarda maksimum benzerlik araması yapan asenkron bir paralel algoritma önerisidir. Çalışma 2012 yılında Miwako Tsuji,

Mitsuhisa Sato, Akifumi S. Tanabe, Yuji Inagaki ve Tetsuo Hashimoto tarafından yapılmıştır (Tsuji vd. 2012). Çalışmada, paralel genetik algoritmanın, filogenetik ağaçlar üzerinde yapılan aramalarda kaçınılmaz olduğu ancak çeşitlilik problemi ile karşılaşıldığından ve önerilen asenkron yöntem ile, iletişim ve senkronizasyon ihtiyacı olmadan bu çeşitliliğin sağlanabileceği gösterilmiştir.

Yine 2012 yılında yapılmış bir diğer çalışma, paralel genetik algoritmalar için önerilen, yeni bir göç stratejisidir. Max-Min stratejisi diye adlandırılan bu yöntemi, tepe tırmanma algoritması ile birleştirerek, performans artışı elde etmişlerdir (Falahiazar vd. 2012).

Genetik algoritmaların çeşitli alanlarda kullanıldığını gösteren diğer bir çalışmada 2011 yılında F. Valdez, P. Melin ve H. Parra tarafından yapılmıştır. Yapılan çalışmada, motif tanıma algoritmalarında modüler yapay sinir ağlarının paralel genetik algoritmalar ile optimize edilmesi çalışılmıştır (Valdez vd. 2011).

Genetik algoritmalarin, gerçek zamanlı uygulamalarki yetersizliğini ele alan bir çalışmada Wang Zhu-rong, Ju Tao, Cui Du-wu ve Hei Xin-jong tarafından 2011 yılında yapılmıştır. Bu çalışmada hibrid bir paralel genetik algoritma çalışması yapılmıştır. Çok çekirdekli bilgisayarlardan oluşan küme sistemler üzerinde, büyük parçacıklı paralel algoritma kullanılmıştır (Zhu-rong vd. 2011).

Tez dört ana başlık altında ele alınacaktır;

1. İlk başlık genetik algoritmalar olacaktır. Bu bölümde genetik algoritmaların tarihi, kimler tarafından geliştirildiği, gelişim aşamaları ve geliştirilme amaçlarına yer verilecektir.
2. Tezin ikinci bölümünde, geçmişten günümüze paralel programlama ve şu an mevcut olan ve en yaygın kullanılan paralel programlama kütüphaneleri anlatılacak ve tezde neden Intel TBB kütüphanesinin tercih edildiği anlatılacaktır.
3. Tezin üçüncü bölümünde, matematik ve bilgisayar biliminde kriter fonksiyonu olarak kullanılan ve uygulamamızda uygunluk fonksiyonu olarak yer alacak olan matematiksel fonksiyonlardan bahsedilecektir.
4. Tezin son bölümünde ise, test sistemlerimizde, farklı kombinasyonlar sonucu elde edilen test verilerine yer verilecektir. Her sistemde seri, iki

kanal paralel programlama, dört kanal paralel programlama ve test işlemcisinin desteklemesi durumunda sekiz kanal paralel programlama süreleri ve kaynak kullanımları ele alınarak değerlendirilecektir.

Bu çalışmadaki uygulama, tek bilgisayar üzerinde, tek işlemci barındıran ancak birden fazla çekirdek bulunduran MIMD mimarisinde, SMP türü işlemciler kullanılarak, veri paralelleştirme tekniği ile ve SPMD (Tek program çoklu veri - Single Program Multiple Data) metodolojisi kullanılarak geliştirilmiştir. Bu sayede sonuç alınan bilgisayardaki işlemci kaynağının tamamını en verimli şekilde kullanılması hedeflenmiştir.

2. GENETİK ALGORİTMA

Genetik Algoritma, günümüzde neredeyse evrensel olarak GA (Genetic Algorithms) şeklinde kısaltılmıştır. İlk olarak John Hollan (Holland, 1973) tarafından, *Adaptation in Natural And Artifiticial Systems* kitabında kullanılmıştır. Bu başlangıçtan itibaren araştırma ve uygulamaları, orjinal GA' dan çok daha derinlere gitmiştir. Ancak metasezgisel içerik olarak, GA'nın orjinal hali, bir kişinin konu hakkında bilmesi gerekenleri kapsar demek adil olur (Glover, Kochenberger 2003).

2.1. Genetik Algoritmaların Tarihi

Holland'ın konunun geliştirilmesindeki etkisi önemli olmakla birlikte, farklı geçmişlere sahip bir çok bilim adamı benzer konuların geliştirilmesinde dahil olmuşlardır.

1950-1960 yılları arasında bir çok bilim adamı gelişim algoritmaları üzerinde, gelişimin mühendislik problemlerinin optimize edilmesinde kullanılabilecek bir teknik olabileceği noktasında birbirinden bağımsız olarak çalışma yapmışlardır. Bunların en önemlisi David E. Golgberg'tir (Goldberg, 1989). Goldberg, genetik algoritmanın kurucusu sayılan John Holland'ın öğrencisidir. Olağan hali ile GA Goldberg tarafından tanımlanmıştır (Gen ve Cheng 1997).

Daha sonraları, I. Rechenberg (Rechenberg, 1973) ilk defa gerçek değerli parametreleri optimize edilmesine yönelik tasarlanmış Gelişim Stratejileri'ni (Evolution Strategies) tanıttı. Bu fikir daha sonraları Scwefel (Schewefel, 1984) taradından daha da geliştirildi. Gelişim stratejileri aktif bir araştırma alanı olarak kalmaya devam etti. 1990'ların sonlarında kesişmeye başlamasına rağmen, genetik algoritmalardan bağımsız olarak geliştirilmesi devam etmiştir.

Fogel, Owens ve Walsh 1966'da Gelişimsel Programlamayı geliştirmişlerdir. Bu teknik, verilen görevlere uygun aday çözümlerin sonlu-durum makinaları olarak ifade edilmesi ve bunların durumlarının rasgele olarak mutasyona uğratılması ve geçiş diagramları kullanarak en uygunun bulunmasıdır.

Yukarıda bahsedilen üç kanal, gelişim stratejileri, gelişimsel programlama ve genetik algoritma, gelişimsel hesaplamanın bel kemikleridir.

1950 ve 1960'lı yıllarında daha bir çok araştırmacı optimizasyon ve makina öğrenmesi için gelişimden esinlenen algoritmalar üzerine çalışmalar yapmışlardır. Box (1957), Friedman (1959), Bledsoe (1961), Bremermann (1962) ve Reed, Toombs, Baricelli (1967) hepsi bu alanda çalışmalar yapmış ancak gelişim stratejileri, gelişimsel programlama ya da genetik algoritmalar yönünde ya çok az ya da hiç ilgi çekmemiştir. Ayrıca bir kısım biyolog da, kontrollu deneylerin simulasyonu için bilgisayar kullanmışlardır. Bunlara örnek olarak da Baricelli (1957), Fraser (1957), Martin ve Cockerham (1960) gösterilebilir.

Genetik algoritma John Holland tarafından 1960'larda keşfedilmiş ve O ve O'nun öğrenci ve meslektaşları tarafından geliştirilmiştir. Gelişim stratejileri ve gelişimsel programlamanın tersine, Holland'ın ana amacı belli bir problemi çözmeye yönelik bir algoritma tasarlamak değildi. Onun çalışma alanı doğada olduğu şekli ile, adaptasyon sürecini bilgisayar ortamına aktarmaktı. 1975 yılında yayınladığı "Adaptation in Natural and Artificial Systems" adlı kitabında, genetik algoritmaları biyolojik sistemlerin bir soyutlaması olarak sundu. Holland, genetik algoritmaları, kromozomlardan oluşan bir popülasyonu, doğal seçim ile genetikten esinlenen çaprazlama, mutasyon ve ters çevirme işlemleri ile bir sonraki popülasyona gelişmeye uğratma şeklinde ifade etmiştir. Bu yaklaşımda her kromozom genlerden oluşmaktadır. Her gen ise belirli bir allelinin örneği şeklinde ifade ediliyor (örneğin 0 veya 1). Seçim operatörü, bu kromozomlardan üremeye izin verileceklerin seçilmesidir ki daha uygun kromozomların çiftleşmesi sonucu oluşacak olan yeni nesillerin, daha kötü uygunluğa sahip kromozomların çiftleşmesi sonucu oluşacak nesillerden daha uygun olacağı yaklaşımına dayanır. Çaprazlama, iki kromozomun alt kümelerinin karşılıklı değişimi ile yeni kromozomların türetilmesidir. Mutasyon ise rasgele olarak kromozomdaki bazı değerlerin değiştirilmesi prensibine dayanır. Ters çevirme ise, kromozomun bir kısmının ters çevrilerek genlerin tekrar düzenlenmesidir.

Buradaki açıklamada bahsedilen, Holland'ın bulduğu popülasyon temelli çaprazlama, ters çevirme ve mutasyon bu alandaki büyük bir yenilikti. Aksine Rechenberg'in gelişim stratejileri çalışması, bir ebeveyn ve onun mutasyona uğramış hali olan bir yavrudan oluşan popülasyondan ibaretti. Daha sonradan çok bireyli

popülasyon ve çaprazlama eklenmiştir. Fogel, Owens ve Walsh'da, çeşitliliği sağlamak için sadece mutasyonu kullanmışlardır(Melanie, 1999).

2.2.Genetik Algoritmaların Biyolojik Temelleri

Genetik algoritmalar doğal seçimi taklit eden arama algoritmalarıdır. Bütün yaşayan canlılar hücrelerden oluşur. Her hücre, bir veya daha fazla kromozom (DNA dizileri) ihtiva eder. Bu yapı canlıların parmak izidir. Kromozomlar, her biri belli bir proteini ifade eden genlere parçalanabilir ve her gen kromozomun belli bir yerinde yerleşik durumdadır.

Eşeyli üreyen bir çok canlı çift kromozom taşır (örneğin insan). Çiftleşme sırasında, çaprazlama meydana gelir. Her ebeveyn, genleri bir gamet oluşturacak şekilde kromozom çifti arasında değiş tokuş eder. Sonra her ebeveyndeki gametler birleşerek, çift kromozom haline gelir. Tek kromozomlu canlılarda ise, ebeveynlerin tek kromozomu arasında gen değişimi gerçekleşir. Tek nükleotidin ebeveynden yavruya değişerek geçmesi mutasyon olarak ifade edilir. Bir organizmanın uygunluğu, yeni nesiller üretecek kadar yaşaması veya ürettiği yeni nesil sayısı olarak ifade edilmektedir.

Genetik algoritmada, kromozom probleme üretilen aday çözümdür. Genelde bit dizileri olarak kodlanır. Gen ise kromozom içindeki bir bit veya çözümün belirli bir kısmını kodlayan ardışık bir bit dizisidir. Çaprazlama basit olarak iki kromozom arasındaki gen değişimidir. Mutasyon, rasgele seçilmiş bir bitin ters çevrilmesidir. Uygunluk, probleme ve problemin soyutlaştırılmasına göre değişiklik gösterebilmekle birlikte, genelde seçilen bir matematiksel fonksiyondan elde edilen sonuç olarak ele alınmaktadır.

Bir çok genetik algoritma uygulamasında, popülasyon tek kromozomlu bireylerden oluşmaktadır (Langton, 1998).

2.3.Genetik Algoritmanın Genel Yapısı

Genetik algoritma, doğal seçim temeline dayanan bir optimizasyon ve arama tekniğidir. Bir popülasyondaki bireylerin belli şartlar altında değişerek, uygunluğu maksimize eden yeni nesillerin türetilmesi prensibine dayanır.

Genetik algoritmalar gerek optimizasyon gerek de bilimsel araştırmalarda sıkça kullanılmış algoritmalardan bir tanesidir. Bu genetik algoritmaların esnek yapısından kaynaklanmaktadır.

Genetik algoritmaların avantajları aşağıdaki şekilde özetlenebilir;

- Sürekli veya kesikli değişkenleri optimize eder.
- Ön bilgi gerektirmez.
- Paralel bilgisayarlar için uygundur.
- Optimum çözümlerden oluşan bir küme üretir, tek çözüm değil.
- Kromozom kodlamasının probleme göre değiştirilebilir olması.
- Genetik operatörlerin, uygulanan problemin karakteristiğine göre farklılık gösterebilmesi.

Genel olarak genetik algoritmalar gücünü, geniş çözüm uzayından almaktadır. Bunun sonucu olarak küçük popülasyon gerektiren veya kolay denilebilecek problemlerin çözümünde, alternatif yöntemler genetik algoritmalardan daha hızlı çözüm üretebilmektedirler. Genetik algoritmanın bu gücü aynı zamanda zayıflığı da denilebilir çünkü çok sayıda bireyden oluşan bu popülasyondaki bütün bireylerin uygunluklarının başta ve her değişiklikte tekrar hesaplanması gerekmektedir. Bu da işlem yükünü arttırmakta, doğal olarak genetik algoritmanın uygulanacağı problemin geniş bir çözüm uzayına ihtiyaç duyan bir problem olması gerekmektedir.

Genetik algoritmaların uygulama adımları temel olarak şu şekildedir;

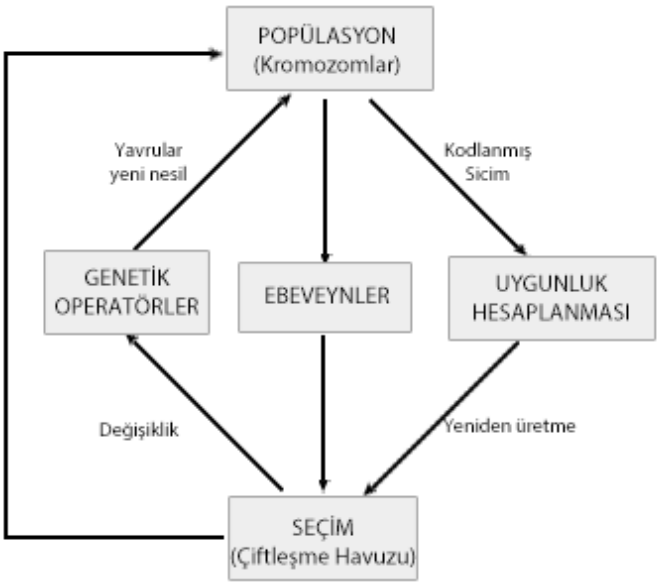
1. İlk popülasyonun seçilmesi (bu işlem genel olarak rasgele üretilen bireyler şeklinde uygulanmaktadır).
2. Her bireyin uygunluk değerinin hesaplanması.
3. Bitiş şartı sağlanana kadar (belli bir iterasyon sayısı ve bir eşik değeri).
   3.1.Çiftleşme için en uygun bireylerin seçilmesi.

3.2. Bu ebeveynlerden, genetik operatörler kullanılarak yeni bireylerin elde edilmesi.

3.3. Yeni bireylerin uygunluklarının hesaplanması.

3.4. Popülasyondaki daha kötü uygunluk değerine sahip bireyler ile yeni üretilen bireylerin yer değiştirilmesi.

Genetik algoritmaların temel adımlarının grafiksel olarak gösterimi şekil 2.1'de verilmiştir. Şekilde gösterilen bölümlerden, genetik algoritmalarda çeşitliliği genetik operatörler sağlar, kötü nesillerin elenmesi işlemi seçim bölümünde yapılır ve son olarak bireylerin kıyaslama işlemi uygunluklarına göre yapılır.



Şekil 2.1: Genetik Algoritmanın Çalışma Döngüsü

Genetik algoritmaların avantajları yanında dezavantajları da sayılabilir. Genetik algoritmaların en büyük dezavantajı, birden fazla lokal minimum noktası barındıran çözümlerde, çözümün lokal noktalara takılı kalmasıdır. Örneğin, tezin kriter fonksiyonlarının anlatıldığı bölümde anlatılan ve tezin test fonksiyonlarından bir tanesi olan rastrigin fonksiyonunun, birden fazla lokal minimum noktası bulunmaktadır. Bu durumda algoritmanın, bu lokal noktaların bir tanesine takılıp, çözümü bulamamasına yol açabilir. Ancak bu durum ek algoritmalarla giderilebilir. İlk popülasyonun rasgele oluşturulması, ortogonal diziler (Leung, Wang 2001) veya fidan algoritması (Karcı, 2002) gibi farklı algoritmalar ile biraz daha kontrollü olarak oluşturulabilir. Tabi bu

uygulanacak ek algoritmalar, probleme has olmaktadır ve problemin karakteristiğinin bilinmesi gerekmektedir.

2.4.Temel Genetik Operatörler

Bu bölümde, en temel genetik operatörler ve bu operatörler için en sık kullanılan teknikler anlatılacaktır.

En basit hali ile genetik algoritmalar üç çeşit operatörden meydana gelir. Bunlar seçim, çaprazlama ve mutasyondur.

2.4.1. Seçim operatörü

Seçim operatörü yeniden üretim için popülasyon içinden kromozomların seçilmesidir. Uygunluk değeri yüksek olan kromozomlar, yeni nesil üretilmesinde daha fazla kullanılacak kromozom olur doğal olarak.

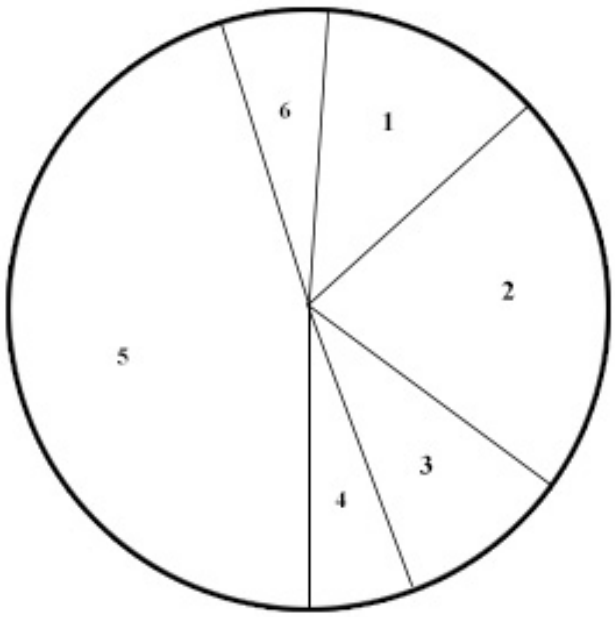
Seçim operatöründe genel olarak iki sınıfta incelenebilir.

2.4.1.1.Roulette tekerleği seçimi

Roulette tekerleği en basit seçim yaklaşımıdır. Bu seçimde, bütün bireyler uygunluklarıyla orantılı olarak bir tekerleğe yerleştirilir. Her birey tekerlekte bir segmente sahiptir. Roulette tekerleği seçiminde, bireylere performansına göre seçim önceliği vermeye dayanır. Roulette tekerleği [0,N] aralığında bireylerin normalleştirilmesi ile elde edilen bir ihtimal tekerleği olarak tanımlanabilir. Buradaki N sayısı popülasyondaki bütün bireylerin uygunluklarının toplamıdır. Şöyle ki, bireylerin uygunluklarının, popülasyondaki bütün uygunlukların toplamına bölünerek, her bireyin değerine göre tekerlekten uygunluğu ile orantılı dilim almasıdır. Örneğin şekil 2.2'de uygunluğu en yüksek olan 5 numaralı birey, tekerlekten en büyük dilimi kaplamıştır. Buna karşılık 4 ve 6 numaralı bireylerden uygunlukları kötü olduğundan en küçük

alanları kaplamışlardır. Seçim yapılacağı zaman, 0-N aralığında rasgele bir sayı üretilir ve hangi bireyin segment aralığına denk gelirse, o birey üretim için seçilmiş olur. Doğal olarak, daha geniş alan kaplayan bireyin seçim ihtimalide arttırılmış, böylece iyi çözümün bir sonraki nesile aktarılma ihtimali de arttırılmış olur.

Roulette tekerleğinde, ihtimal artmasına rağmen rasgele üretilecek seçim sayısının ard arda kötü bireyleri seçmesi ve bunun sonucu çözümün gelişememe ihtimali vardır. Şekil 2.2'deki örnek ele alındığında, çiftleşme için seçilecek iki bireyi seçmek için tekerleği iki kere çevirmek gerekmektedir. Her çevrimde aynı eleman gelebileceği gibi, örneğin ardarda 4 ve 6 numaralı kötü uygunluğa sahip bireylerde seçilebilir. Bu istenmeyen bir durumdur. Ancak genetik algoritmaların doğası düşünüldüğünde, bu durum doğal bir sonuçtur denilebilir.



Şekil 2.2: Roulette Tekerleği Seçim Yöntemi

2.4.1.2.Mertebe seçimi

Mertebe seçiminde (Rank Selection), popülasyondaki bireyleri uygunluklarına göre sıralanması ve derecelendirilmesidir. Sonra her bireyin, kendi uygunluğuna göre ihtimali hesaplanır ve seçim bu ihtimale göre yapılır (Baker, 1985). Bireyler kendi seçim olasılıklarına göre seçilir.

Mertebe seçimi çok hızlı yakınsamayı engeller ve seçim baskısı açısından roulette tekerleği seçiminden farklıdır. Roulette tekerleğine göre daha basit ve etkili bir yöntemdir (Glover, Kochenberger 2003).

2.4.1.3.Turnuva seçimi

Turnuva seçimi katı en iyiyi seçmeye yönelik bir seçim mekanizmasıdır. N sayıdaki kromozom arasında, uygunluğu en iyi olanların çiftleşme için seçilmesi mantığına dayanmaktadır. Bu yaklaşım, iki kromozomlu bir popülasyonda, derece seçiminin uygulanması gibi düşünülebilir (Glover, Kochenberger 2003). Doğal olarak her seçim aşamasında, en iyi iki kromozom seçilmiş olacaktır.

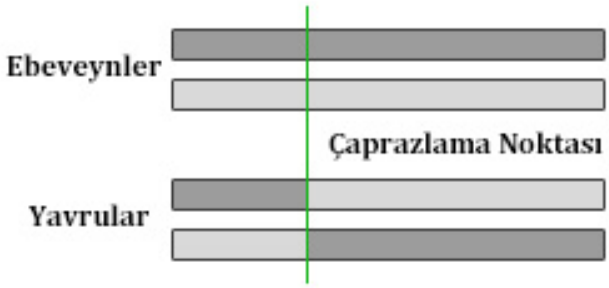
Turnuva seçiminin avantajı, sadece sıralı bir kromozom dizisine ihtiyaç duyması denilebilir. Ancak bu tekniğin en büyük dezavantajı ise, popülasyonun başlangıçta rasgele bireylerden üretildiği kabul edilirse, hep en iyilerin çaprazlama işlemine tabi tutulacağı için, çeşitlilik sağlayamaz.

Bu seçim tekniklerinden başka, önerilen tekniklerde bulunmaktadır. Ancak temel olarak yukarıdaki üç teknik kullanılmaktadır. Örneğin Rakesh Kumar (Kumar, 2012), Tavlı seçim (Annealed Selection), yukarıda bahsedilen roulette tekerleği ve derece seçiminin harmanlanmış halidir.

2.4.2. Çaprazlama

Çaprazlama temel olarak, ebeveynlerin aralarında karşılıklı gelen genlerinin değiştirilmesidir. Daha detaylı bir ifade ile, rasgele seçilen bir veya daha fazla noktadan iki kromozomun parçalanarak, iki kromozomdaki parçaların karşılıklı olarak yer değiştirilmesidir. Örneğin 1110010001 ve 1001000110 kromozomlarının üçüncü indisten itibaren çaprazlamaya tabi tutulması durumunda, ilk kromozomun ilk üç geni ve ikinci kromozomun ikinci yedi geni ilk yavruyu, yani 1111000110, ikinci kromozomun ilk üç geni ve ilk kromozomun ikinci yedi geni birleştirilerek ikinci yavru,

yani 1000010001 kromozomu meydana getirilir. Tek noktalı çaprazlama grafiksel örneği şekil 2.3'de verilmiştir.



Şekil 2.3: Tek Noktalı Çaprazlamanın grafiksel gösterimi

Çaprazlama noktasında farklı öneriler bulunmaktadır. En genel çaprazlama teknikleri tek noktalı çaprazlama ve çok noktalı çaprazlamadır. Tek noktalı çaprazlama yukarıda anlatıldığı ve şekil 2.3'de gösterildiği şekildedir. Çok noktalı çaprazlama, kesim noktasının birden fazla olması ve karşılıklı denk gelen genlerin değişimidir.

Çaprazlama operatörü genetik algoritmaların en önemli ve temel operatörüdür (Gen, Cheng 1997). Çünkü kromozomların çözüm sürecinde gelişerek sonuca doğru gelişmesindeki en büyük etken çaprazlamadır.

2.4.3. Mutasyon

Bu operatör, rasgele seçilen bir genin değerinin değiştirilmesidir. Eğer kromozomlar için ikili tabanda kodlanmış bir ifade kullanıldığı kabul edilirse, bu durumda mantıksal değil (NOT) operatörünün rasgele seçilen gene uygulanması örnek olarak verilebilir. Örneğin ikili tabanda kodlanmış olan 1110010001 kromozomunu ele aldığımızda, beşinci genine mutasyon uygularsak, kromozomun 1110110001 haline gelmesidir.

Elbette ki, probleme göre tercih edilen kodlama yöntemine göre, farklı mutasyon algoritmaları ya da fonksiyonları tercih edilebilir.

Genel olarak, genetik algoritma operatörlerinin tamamı için ortak olan bir nokta vardır oda hepsinin kişiselleştirilebilir olmasıdır. Bu da herkesin uyguladığı genetik

algoritmanın eşsiz olması demektir (Glover, Kochenberger 2003). Daha detaylı anlatılacak olursa, popülasyonun ilk oluşturulması, seçim aşamasında, çaprazlama sırasında ve mutasyon uygularken farklı yaklaşımlar sergilenebilir. Doğal olarak genetik algoritma, kişiselleştirilebilir bir algoritma olması aynı zamanda değiştirilerek farklı karakteristikteki problemlerin çözümünde kullanılabileceğinin bir delilidir. Zaten genetik algoritmaların bu özelliği yüzünden havacılıktan (Chipperfield ve Fleming 1996) çizge renklendirmeye (Erben, 2001), motif tanımadan (Abed v.d. 2010) robotiğe (Ayala, Coelho 2012) kadar bir çok alanda kendine yer bulmuş bir algoritmadır.

2.5.Uygunluk Fonksiyonu

Uygunluk fonksiyonu, genetik algoritmanın uygulandığı probleme göre, bireylerin verimliliklerinin kıyaslanabilmesi amacı ile, probleme has olarak kullanılan fonksiyon yada fonksiyonlardır. Örneğin, iki tabanda kodlanmış bir popülasyonu ele alacak olursak, çözümün en büyük değere sahip kromozom olduğunu kabul edelim. Bu durumda uygunluk fonksiyonumuz sadece sayı olarak büyük sayı daha uygun şeklinde bir yaklaşım olabilir.

Yukarıda bahsedildiği gibi, genetik algoritmanın asıl operatörü çaprazlamadır. Çünkü çeşitliliği gerçekleştiren ana operatördür. Ancak ilk oluşturulan ve sonradan türetilen bireylerin verimliliklerinin kıyaslanabilmesi için bir uygunluk fonksiyonuna ihtiyaç vardır. Çünkü genetik algoritmalar tek çözüm üreten bir algoritma değildir. Birden fazla çözümden oluşan bir çözüm kümesi verir. Doğal olarak da bu çözümlerin bir kritere göre sıralanması ve en uygun çözümün bulunması gerekmektedir.

Bu tezde uygunluk fonksiyonu olarak, kriter fonksiyonları kullanılmıştır. Bu fonksiyonlar Bölüm 4 de anlatılacaktır. Tezde yapılan çalışmada, bu kriter fonksiyonlarının minimum noktaları genetik algoritmalar kullanılarak bulunmaya çalışılmıştır. Örneğin Rastrigin fonksiyonu ile yapılan testte, bireylerin uygunlukları, fonksiyonda değer hesaplatılacak sayılardan oluşan kromozomlarının bu fonksiyondaki sıfıra yakınlıkları gözetilmiştir. Rastrigin fonksiyonunun global minimum noktasının sıfır noktası olduğu bilinmekte ve bireylerin fonksiyona sokulduklarındaki değerlerinin sıfıra yakınlıkları uygunluk değerleri olarak ele alınmaktadır.

Tezde genetik algoritmalar bölümünde anlatılan bilgiler, bölüm içerisinde belirtilen referanslar dışında, referanslar bölümünde detayları verilen aşağıdaki kaynaklar kullanılarak derlenmiştir: (Goldberg ve Deb, 1989), (Michalewicz, 1992), (Vose, 1999), (Haupt, 2004), (Yang, 2010), (Salomon, 1996), (Vose ve Whitley 1995), (Karaboğa, 2011) ve (Gend ve Cheng, 1997).

3. PARALEL PROGRAMLAMA

Paralel programlama anlatılırken, öncelikle paralel programlamaya olan ihtiyaç ve sebepleri anlatılacak, daha sonra paralel programlamanın teorik temelleri, teknikleri ve mevcut durumda en çok kullanılan paralel programlama kütüphaneleri anlatılacaktır.

3.1.Paralel Programlamaya Olan İhtiyaç

Günümüzde bilgisayarlar çok hızlı gelişim göstermektedir. On yıl önceki bilgisayarlarla kıyaslandığında bile, gerek mimari gerekse saat hızı olarak, işlemcilerde büyük bir gelişme gözlenmiştir. Ancak buna rağmen, yapılan bilimsel araştırmalarda yada bazı uygulamalarda sonuç alabilmek için saatler, günler hatta haftalar beklemek gerekebilmektedir.

Paralel programlamaya olan ihtiyacı ifade edebilmek için farklı bir bakış açısı sunmak gerekirse şöyle bir örnek verilebilir. Bilgisayar ve ilk işletim sistemlerinin tarihini göz önüne alırsak, teknolojik gelişmelerin çözmüş olduğu problemler kesinlikle mevcut olabileceği gibi göz ardı edilemeyecek bir gerçek vardır. Bu en kısa ifadesi ile, ister 1990 yılında olunsun ister 2012 yılında, dünyadaki mevcut teknolojik gelişmeler, bazı çalışmaları istenilen seviyede hızlandırmıyorsa ne yapılacağı gerçeğidir. Ya teknolojinin istenilen seviyeye gelmesi beklenecek ki bu bazı durumlarda çok uzun zaman alabileceği gibi imkansız seviyede bir artışa ihtiyaç duyan problemler bile olabilir, ya da mevcut teknoloji kullanılarak, nasıl bir çözüm üretileceğini düşünmek gerekir. Mevcut kaynaklarla bir problemin çözümünü hızlandırmak için, problemi parçalara ayırarak, her parçayı farklı bilgisayarlarda çözmek en optimum çözüm olacaktır. Yani tercih edilmesi gereken yöntem kesinlikle paralel programlamadır.

Tezin bu bölümünde paralel programlamayı anlatırken, sadece paralel programlama işlerinizi hızlandırır ifadesinden ziyade, paralel programlama bilimsel çalışmalar için kaçınılmaz olarak kullanılması gereken bir tekniktir yaklaşımını da ifade etmek amacı güdülmüştür.

Tezin giriş bölümünde de ifade edildiği gibi, 2011 yılındaki, işlemci saat hızı dünya rekoru 8,429 MHZ'dir (Guinness, 2011). Bu hız aşımı laboratuvar ortamında ve ortaya çıkan ısıyı önleyebilmek için sıvı nitrojen kullanılarak yapılmıştır. Doğal olarak bu hıza normal şartlar altında çıkılamayacağı göz önündedir. Ancak çıkılabildiğini kabul edilse bile, 8 GHZ hızındaki tek işlemci saniye 8 milyar işlem gücüne sahiptir. Bunun yerine yine, bugün netbook bilgisayarlarda bile kullanılan Atom 1.6 GHZ (Intel Atom 230) hızındaki işlemcilerden 6 tanesini barındıran bir sistem düşünülsün. Bu sistem teorik olarak saniyede 9.5 milyar işlem gücüne sahiptir ve bu konfigürasyondaki bir sistemi toparlamak, bahsedildiği gibi bir işlemciyi 8 GHZ in üzerine çıkarmaktan çok daha kolaydır. Kaldı ki, daha fazla işlem hızına ihtiyaç duyulduğunda daha fazla Atom işlemcili bilgisayar eklenebilecekken, 8 GHZ in üzerine daha fazla çıkmak kolay olmayacaktır.

3.2.Paralel Programlama Temelleri

Paralel programlama, işlem zamanını kısaltmak için paralel olarak işlem yapabilen bilgisayarların bir araya getirilerek kullanılmasıdır. Bugün paralel programlama, iklim modelleme, uçak tasarımı ve moleküler dinamik bilim alanlarındaki problemlerin çözümünde bir standart olarak kabul edilmektedir (Quinn,2003).

Paralel sistemlerde farklı sınıflandırmalar yapmak mümkündür. Topolojilerine göre sınıflandırmak, bağlantı yapılarına göre sınıflandırmak, veriyolu tasarımına göre sınıflandırmak gibi. Ancak bu tezde kullanılacak olan sınıflandırma, ilk olarak sistemin içerdiği bilgisayar sayısına göre, bu aşamadan sonra tezde kullanılacak olan tek bilgisayarlı sistemlerin bulundurdukları işlemcilerinin iç mimarisine göre ve en son olarakta paralelleştirme tekniklerine göre yapılacaktır. Bu bakış açısı ile üç sınıfa ayırmak mümkündür:

- Birden fazla bilgisayar içeren küme (cluster) sistemleri.
- Tek bilgisayar üzerinde çok çekirdek barındıran sistemler.
- Ve yukarıdaki iki sistemin karışımı olan sistemler.

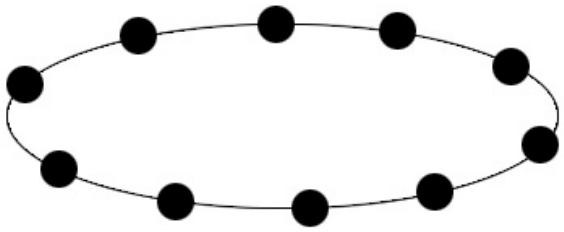
Küme sistemleri bir yüksek hızlı ağ vasıtası ile birbirine bağlanmış bilgisayar grubudur. Temel olarak farklı bilgisayarlardaki işlemcilerin, farklı topolojiler yardımı ile, birbiri ile mesajlaşabilecekleri bir yapıdır denilebilir.

Bu şekildeki paralel sistemler bu tezin konusunun dışında olduğundan, detaylarına girilmeyecek, sadece bir bakış açısı olarak incelenecektir. Bu alanda bir çok farklı ağ topolojisi geliştirilmiştir. Ancak bunların belli bir sınıflandırma yapısında tanımlamak zordur. Bu topolojilerden en genel kullanılanları şunlardır:

- Halka topolojisi
- İkili ağaç topolojisi
- Yıldız topolojisi
- Izgara topolojisi
- Çepeçevre ızgara topolojisi
- Hiperküp topolojisi

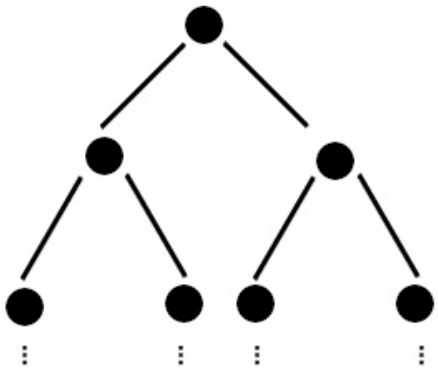
Bu topolojiler, genel olarak bilgisayarların ağdaki yerleştirilme veya daha genel bir ifade ile bağlanma şekillerine göre isimlendirilmektedir. Yukarıdaki örnek topolojilerin ilk iki tanesi kısaca açıklanacak olursa;

Halka topolojisinde bilgisayarlar birbirlerine bir halka üzerindeymiş gibi bağlanır ve her bilgisayar kendisinden bir önceki ve sonraki olmak üzere iki bilgisayar ile iletişim halindedir. Şekil 3.1' de halka topolojisinin grafiksel gösterimi verilmiştir.



Şekil 3.1: Halka topolojisi grafiksel gösterimi

Yine aynı şekilde, ikili ağaç topolojisi, eşit seviyedeki iki bilgisayarın, bir üst bilgisayara bağlı olması şeklinde köke kadar devam eden ikili ağaç şeklinde bir yapıdır. İkili ağaç topolojisi şekil 3.2'deki gibidir.



Şekil 3.2: İkili ağaç topolojisi grafiksel gösterimi

Üçüncü sistem türünü, çok çekirdekli bilgisayarlar içeren kümeler diye tanımlayabiliriz. Bu sınıfta bu tezin konusu dışındadır.

İkinci sistem ise, bir bilgisayarda entegre olarak birden fazla işlem ünitesini barındıran sistemlerdir. Tek bilgisayar sistemleri mimari açısından dört grupta incelenebilir:

- SISD : Tek komut, tek veri – Single Instruction, Single Data. Geleneksel seri programlama. Tek bir işlemci ve hafızadan oluşan sistemlerdir. Her döngüde tek komut çalıştırır.
- SIMD : Tek komut, çoklu veri – Single Instruction, Multiple Data. Vektör iletişim hattı. SISD gibidir ancak her komut birden fazla veriyi kontrol edebilir.
- MISD : Tek program, tekli veri – Multiple Instruction, Single Data. Çok nadir sistemlerdendir. Bütün işlemciler aynı programı çalıştırır.
- MIMD : Çoklu komut, çoklu veri – Multiple Instruction, Multipla Data. Standart paralel bilgisayarlardır. Bütün çok işlemci barındıran bilgisayarlar bu sınıfa girmektedir. Her işlemci tamamen bağımsız komutlarla, bağımsız veriler üzerinde çalışır.

Bu tezde kullanılan bilgisayarların işlemcilerinin tamamı MIMD sınıfındaki sistemlerdir. Bu sistemlerin en büyük avantajı belki de maliyettir. Zaten teze konu alınmasının temel sebebi de budur. Ancak avantajlarının yanında iki tane temel dezavantajı bulunmaktadır. Bunlar yük dengesinin gerekmesi ve programlamanın zor olmasıdır.

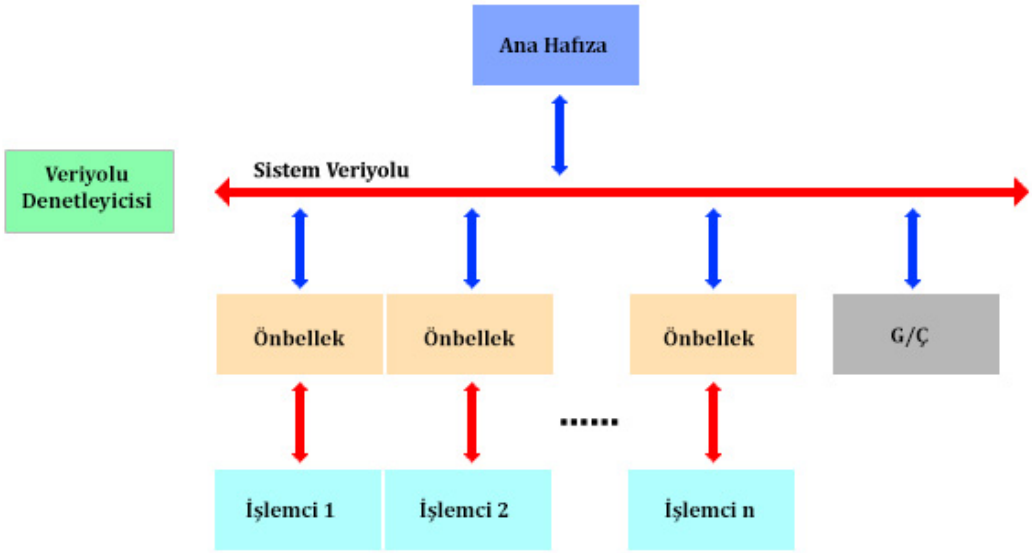
MIMD mimarisi de kendi içerisinde iki bölümde incelenebilir:

- Paylaşımlı bellek kullanan sistemler (Symmetrical Multiprocessor – SMP). Bu sistemlerde, her işlemci fiziksel olarak aynı bellek üzerinde farklı veriler ile çalışır. Veri paylaşımı hızlıdır ancak bant genişliği (bandwidth) problemi vardır.
- Ayrık bellek kullanan sistemler (Massively Parallel Systems – MPP). Bu sistem, birden fazla SISD sisteminin bir ağ ile birbirine bağlandığı sistemlerdir. Yine bu sistemde de bant genişliği problemi vardır. Bu sistemi birden fazla SMP sisteminin yine bir ağ yardımı ile bağlanması şeklinde de tasarlamak mümkündür. Bu durumda hibrid ayrık bellek mimarisi elde edilmiş olur.

Bu tezde kullanılan mimari daha detaylı olarak ifade edilecek olursa, SMP mimarisidir. Günümüzdeki çok çekirdekli mimarilerin çoğunluğu bu mimarideki sistemlerdir. SMP mimarisinde işlemciler ortak bir global paylaşılan belleği kullanırlar. Bu bellek işlemciler arasındaki senkronizasyon ve haberleşmeyi sağlar (Quinn, 2003).

Bu sistemde işlemciler tek bir işletim sistemi ile kontrol edilmektedir. Mimarideki işlemcilerin hepsi homojendir. Bu işlemcilerin her biri farklı programları farklı veriler üzerinde çalıştırabilir ve ortak kaynaklar (ana bellek, giriş-çıkış üniteleri, kesme (interrupt) sistemleri vb.) sayesinde bu işlemleri paylaşabilirler. Her işlemcinin kendine has yüksek hızdaki önbelleği bulunmaktadır. Bunu sebebi ana hafızaya erişimi hızlandırmak ve veriyolu üzerindeki trafiği minimize etmektir. İşletim sistemi desteği ile SMP mimarisinde, yük dengesinin sağlanması için görevler kolaylıkla işlemciler arasında değiştirilebilmektedir.

Bu mimarinin tercih edilme sebebi, giriş bölümünde de ifade edildiği gibi, kolay elde edilebilir olmasıdır. Şekil 3.3’ de SMP mimarisinin iç tasarımı verilmiştir.



Şekil 3.3: Simetrik çok işlemci mimarisi

Bu grupta yer alan işlemcilerden, tezdeki test sistemlerinin üçüncü nesil Intel işlemcileri olanlarda, ayrıca Intel HT (Hyper-Threading) teknolojisi bulunmaktadır. Bu teknoloji, Intel'in 2002 yılında geliştirdiği bir teknolojidir ve ilk olarak Xeon sunucu işlemcilerinde kullanılmıştır.

Genel mantığı, her fiziksel işlemci için, işletim sisteminin iki adet mantıksal işlemci adreslemesi ve yük dağılımını bunlar arasında yapmasıdır. HT'den performans alınabilmesi için, aynı zamanda işletim sistemi desteği de olması gerekmektedir. Çünkü HT teknolojisini barındıran işlemcilerde, işletim sisteminin algıladığının yarısı kadar fiziksel işlemci bulunmaktadır.

3.3.Paralelleştirme Teknikleri

Paralelleştirme teknikleri, paralel uygulama geliştirilirken, hangi seviyede paralelleştirme yapılacağını ifade etmektedir. Bu teknikleri dört grupta toplanmaktadır:

3.3.1. Veri paralelleştirmesi

Her işlem, eşsiz ve bağımsız veriler üzerinde aynı işlemi yapar. Fonksiyonel paralelleştirmeden daha ölçeklendirilebilir bir yöntemdir. Basit bir örnekle açıklamak gerekirse, n tane işçinin bir evi boyaması gibi düşünülebilir. Hepsinin yaptığı iş aynıdır, ancak farklı noktaları boyarlar.

Üst seviye olarak OpenMP veya TBB gibi bir kütüphane ile veya alt seviye olarak MPI ile uygulanabilir.

3.3.2. Fonksiyonel paralelleştirme

Her işlemci farklı bir fonksiyonu veya birbirinden bağımsız olan bir kod bloğunu işler. Buna örnek olarak da n tane işçinin bir evi inşa etmesi verilebilir. Hepsinin farklı bir görevi vardır. Ev inşa etmekte olduğu gibi bu teknikte de bağımsız görevler ele alınmalıdır. Örneğin evin temeli atılmadan kat yapılamayacağı gibi.

Mesaj iletim kütüphaneleri ile uygulanabilir.

3.3.3. Görev paralelleştirme

Bir ana işlem tarafından kontrol edilen, birbiri ile haberleşmeden her işlemcinin aynı fonksiyonu gerçekleştirmesidir. Buna örnek olarak bankamatik işlemleri verilebilir. Her bankamatik aynı işlemi yapar, birbiri ile haberleşmeleri yoktur ve hepsini bir ana işlem (banka sunucusu) kontrol eder.

3.3.4. İletişim hattı paralelleştirme

Her seviyede problemin bir kısmı çalışır. Bir seviyenin çıktısı, bir alt seviyenin girdisidir. Bu tür paralelleştirmenin en optimum hali, her seviyenin işlemini aynı zamanda bitirmesidir. Buna örnek olarak da otomasyon bantları verilebilir. N tane

bantta işlem devam eder. Bantlar NxM şeklinde bir matris olarak düşünülürse, N tane ürün M seviyede farklı işlemlere tabi tutulur ve sonuçta N tane ürün tamamlanmış olarak çıkar.

Bu tezde genetik algoritma paralelleştirilirken veri paralelleştirme tekniği kullanılmıştır.

3.4.Paralel Programlama Kütüphaneleri

Günümüzde bir çok paralel programlama kütüphanesi mevcuttur. Bu kütüphanelerden bazıları, belli bir paralel bilgisayar sisteminde çalışmaya optimize edilmiş olmakla birlikte, bir kısmı GPU gücünü kullanmaya özel bir kısmı ise işletim sisteminin zamanlayıcısı üzerinde bir katman olarak SMP sistemlerinin paralelleştirilmesinde kullanılan kütüphanelerdir.

En çok bilinen paralel programlama kütüphaneleri aşağıdaki şekildedir:

- NVIDIA CUDA Platform (CUDA, 2012).
- MPI (Message Passing Interface)(MPI,2012).
- OpenMP (OpenMP, 2012).
- PThreads (Pthreads, 2012).
- Boost (Boost, 2012).
- Intel TBB (Threading Building Blocks) (TBB, 2012)

3.4.1. Nvidia Cuda

Cuda, Nvidia tarafından geliştirilen paralel programlama modelidir. Grafik işlemcisinin gücünü kullanarak yüksek seviyede performans artışı sağlamaktadır. Cuda, Nvidia tarafından üretilen, Cuda aktif haldeki grafik kartlarında kullanılabilmektedir.

Cuda sayesinde, hiç bir çevirici diline (assembly) ihtiyaç duymadan, direk olarak C, C++ ve fortran kodu ile grafik işlemcisine erişim sağlanmaktadır.

3.4.2. MPI

MPI (Mesaj İletim Ara yüzü – Message Passing Interface), paralel sistemlerin programlanması için geliştirilmiş, dilden bağımsız bir iletişim protokolüdür. Noktadan noktaya (Point-to-point) ve toplu iletişim desteği bulunmaktadır.

MPI, bir çok paralel bilgisayar sisteminde çalıştırılmak üzere, akademik ve endüstriden bir grup araştırmacı tarafından geliştirilmiş, taşınabilir bir mesaj iletim standardıdır. Bu standart, kullanıcıların mesaj iletim programları yazabilmesi için bir çok sintaks ve mantıksal rutin tanımlamaktadır.

MPI, küme sistemlerinde ve yüksek derecede paralel sistemlerde kullanılmak üzere geliştirilmiş bir standarttır.

3.4.3. OpenMP

OpenMP, Unix ve Windows NT platformları da dahil olmak üzere bütün mimarilerde, C/C++ ve Fortran ile paylaşımlı bellek paralel programlama desteği sunan bir uygulama programı ara yüzüdür (Application Program Interface – API). OpenMP, masaüstü bilgisayarlarından süper bilgisayarlara kadar bir çok platformda basit ve esnek olarak, taşınabilir ve ölçeklenebilir paralel uygulamalar geliştirmeye imkan sağlamaktadır.

3.4.4. Pthreads – POSIX Threads

Pthreads, IEEE Posix 1003.1c standardı tarafından belirlenmiş, UNIX sistemler için bir ara yüzdür. Pthreads, kanal oluşturma ve üzerinde oynama yapabilmek için bir API sunar. Unix sistemlerde kullanıma açılmakla birlikte, üçüncü parti paketlerle windows ortamında da kullanılabilir olmuştur.

Pthreads temelde C kütüphanelerinden oluşmaktadır.

3.4.5. Boost

Paralel boost çizge kütüphanesi (Boost,2012), aslında boost çizge kütüphanesinin (BGL,2012) bir uzantısı niteliğindedir. Adından da anlaşılacağı üzere bir çizge kütüphanesidir ve boost çizge kütüphanesinin aynı arabirimlerini koruyarak grafik algoritmaları sunar. BGL (Boost Graph Library) ile yazılmış algoritmalar kolaylıkla paralel versiyonuna çevrilebilmektedir.

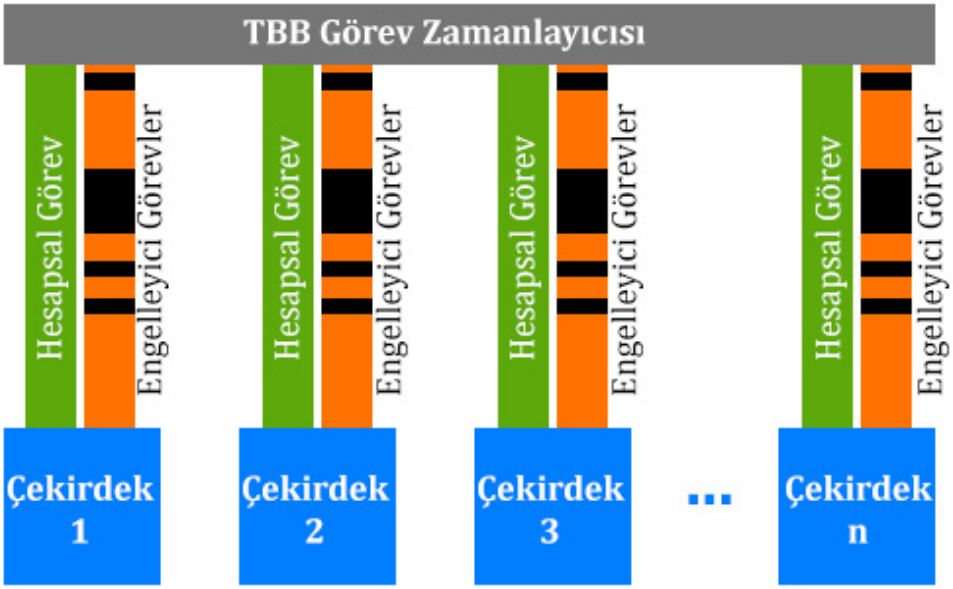
3.4.6. Intel TBB

TBB (Threading Building Block), çok çekirdek mimarisinde optimal seviyede performans elde etmek için gerekli olan alt seviye sicim (thread) detaylarını soyutlayan bir çalışma zamanı kütüphanesidir. Uygulaması zor olan alt seviyedeki sicim programlamasını, C++ şablonlarını kullanarak ortadan kaldırır.

TBB diğer sicim programlama modellerine göre daha az kod gereksinimi duyar. Yazılan uygulama platformdan bağımsız ve taşınabilir olur. Ayrıca kütüphanenin ölçeklendirilebilir olması sayesinde, mevcut çekirdek sayısının artması durumunda uygulamada bir değişikliğe gitmeye gerek kalmamaktadır. Böylece bir ortamda hazırlanmış uygulama, donanımsal olarak daha güçlü veya daha zayıf ortamlarda çalıştırılabilmektedir.

Intel TBB'nin bir diğer özelliği ise, kanal (thread) programlama bilgisine gereksinimi ortadan kaldırmasıdır. Şöyle ki TBB kodu, görevlerden (tasks) oluşur, kütüphane bu görevlerin kanallara geçişini kendi içinde halleder (TBB,2012).

Bu çalışmada Intel TBB'nin tercih edilmesinin sebebi gerek taşınabilir olması, gerek ölçeklendirilebilir olması gerekse de işletim sisteminden bağımsız olmasıdır. Ayrıca az kod gereksinimi de TBB'nin seçilmesinde etkili olmuştur. Şekil 3.4' da TBB'nin çalışma prensibi grafiksel olarak gösterilmiştir.



Şekil 3.4: Intel TBB Görev Zamanlayıcısı

Özet olarak bu tezde genetik algoritma, SMP mimarisi kullanılarak, veri paralelleştirme tekniği ile Intel TBB kütüphanesi ile paralelleştirilmiştir.

Paralel programlama bölümü anlatılırken, bölüm içerisinde verilen referanslar dışında, referanslar bölümünde detaylı verilen aşağıdaki kaynaklardan da yararlanılmıştır: (Chandra v.d. 2001), (Gebali, 2011), (Hockney, 1988), (Duato v.d. 2003), (Zomaya, 1996) ve (Kumar v.d. 1994).

4. KRİTER FONKSİYONLARI

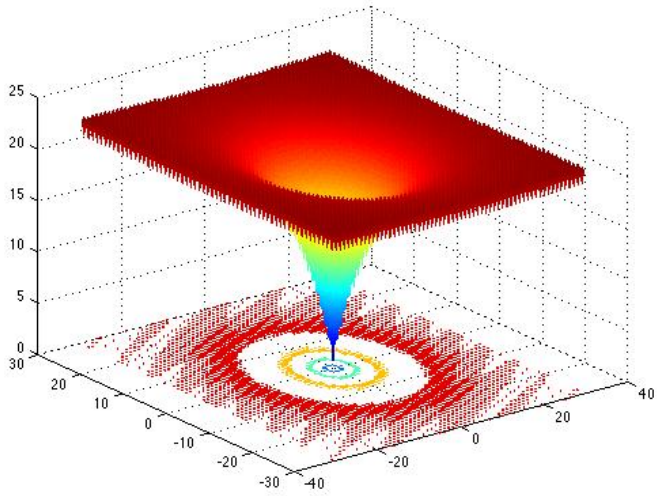
Kriter (Benchmark) fonksiyonları, algoritmaların kıyaslanmasında kullanılan matematiksel fonksiyonlardır. Bir algoritmanın bütün fonksiyonlarda en iyi sonuç vermesi diye bir durum yoktur [Wolpers, Macready 1997]. Ancak bu tezde yapılan çalışmada gerek genetik algoritmalar, gerekse de bu fonksiyonlar literatürde kabul görmüş ve paralel programlama da bize kıyaslama imkanı tanıyacak araçlardır. Doğal olarak genetik algoritmaların kriter fonksiyonlarında verdikleri sonuçlardan ziyade üzerinde durulacak asıl nokta, aynı kriter fonksiyonuna, aynı algoritma üzerinde, paralel programlama ile elde edilebilecek performans artışı asıl odak noktası olacaktır.

Bu tezde kullanılan kriter fonksiyonlarının matematiksel ifadeleri ve grafiksel gösterimleri (Şekil 4.1-4.9) aşağıdaki şekildedir:

4.1.Ackeley Fonksiyonu

Fonksiyonun matematiksel ifadesi :

$$\boldsymbol{F_{ackley}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right) + 20 + e} \quad (1)$$

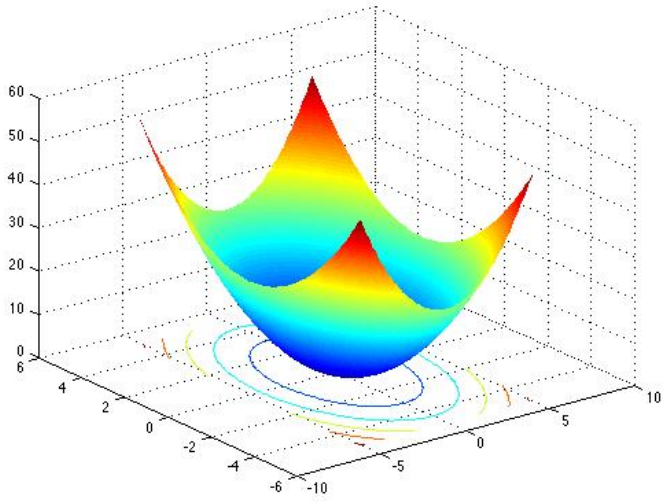


Şekil 4.1: Ackeley fonksiyonu grafiksel gösterimi

4.2.Sphere Fonksiyonu

Fonksiyonun matematiksel ifadesi aşağıdaki şekildedir:

$$F_{sphere}(x) = \sum_{i=1}^{D} x_i^2 \quad (2)$$

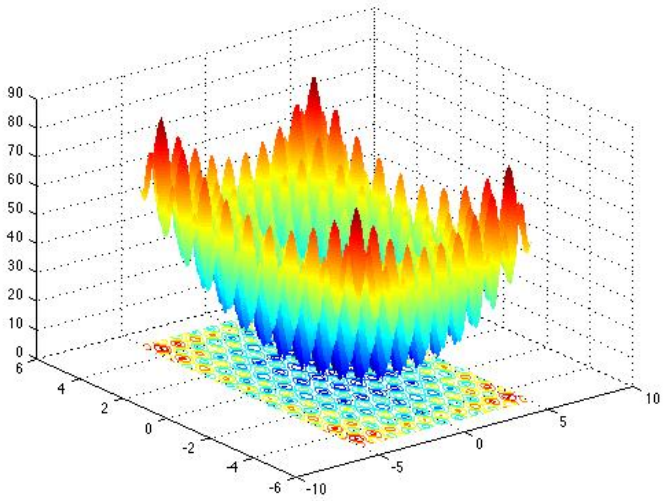


Şekil 4.2: Sphere fonksiyonunun grafiksel gösterimi

4.3.Rastrigin Fonksiyonu

Fonksiyonun matematiksel ifadesi:

$$F_{rastrigin}(x) = \sum_{i=1}^{D} \left[x_i^2 - 10cos(2\pi x_i) + 10\right] \quad (3)$$

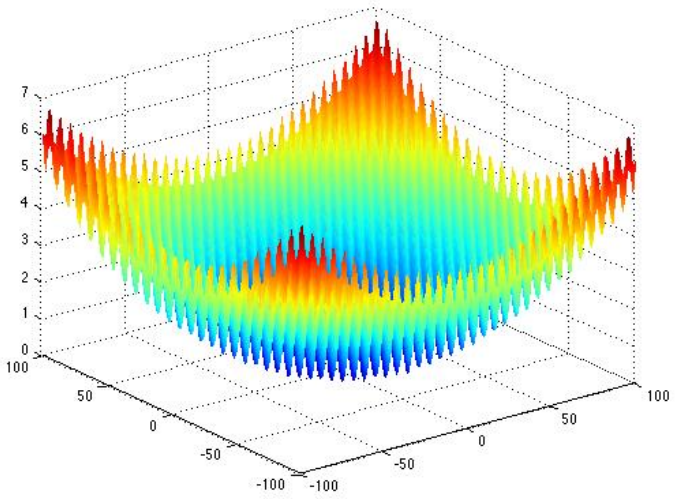


Şekil 4.3: Rastrigin fonksiyonunun grafiksel gösterimi

4.4.Griewank Fonksiyonu

Fonksiyonun matematiksel ifadesi:

$$F_{griewank} = \sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos\frac{x_i}{\sqrt{i}} + 1 \quad (4)$$

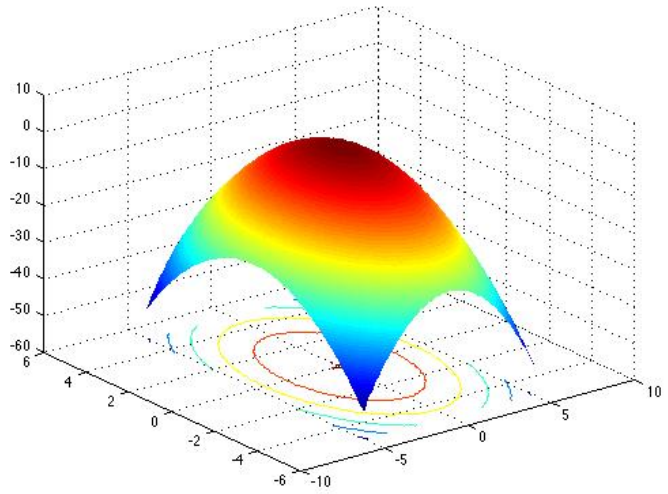


Şekil 4.4: Griewank fonksiyonu grafiksel gösterimi

4.5.Kosinüs Karma Fonksiyonu

Fonksiyonun matematiksel ifadesi:

$$F_{cossineMixture} = \frac{1}{10}\sum_{i=1}^{D} \cos(5\pi x_i) - \sum_{i=1}^{D} x_i^2 \quad (5)$$

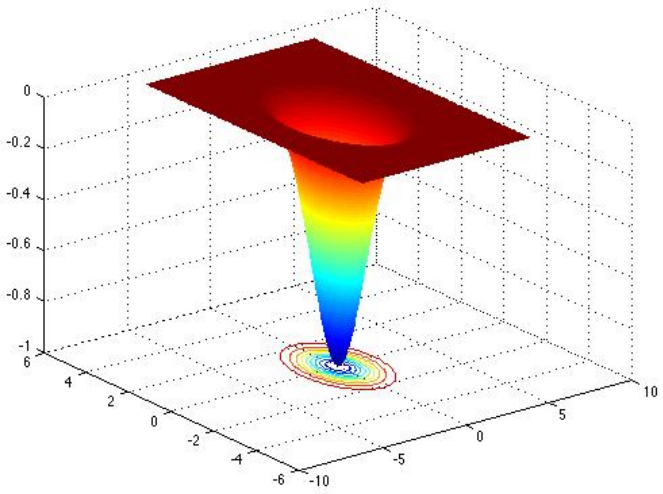


Şekil 4.5: Kosinüs karma fonksiyonu grafiksel gösterimi

4.6.Üssel Fonksiyon

Fonksyionun matematiksel ifadesi:

$$F_{exponential} = -e^{-\frac{1}{2}\sum_{i=1}^{D} x_i^2} \quad (6)$$

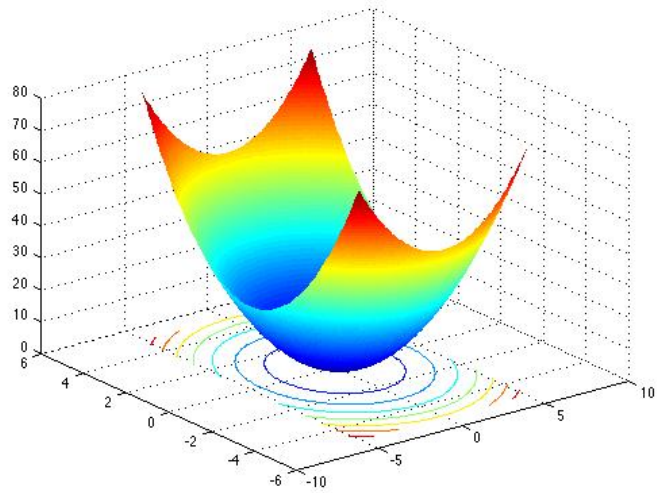


Şekil 4.6: Üssel fonksiyonun grafiksel gösterimi

4.7.Paralel Hyper Elipsoid Fonksiyonu

Fonksiyonun matematiksel ifadesi:

$$F_{hyperEllipsoid} = \sum_{i=1}^{D} i x_i^2 \quad (7)$$

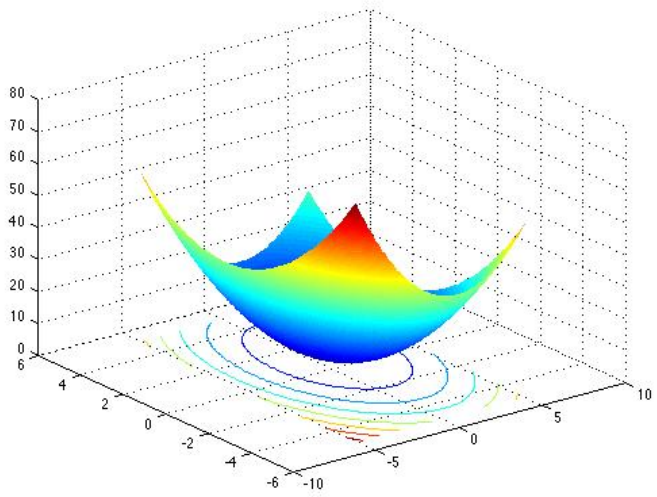


Şekil 4.7: Paralel hyper elipsoid fonksiyonunun grafiksel gösterimi

4.8.Neumaier Fonksyionu

Fonksiyonun matematiksel ifadesi:

$$F_{neumaier} = \sum_{i=1}^{D}(x_i - 1)^2 - \sum_{i=2}^{D} x_i x_{i-1} \quad (8)$$

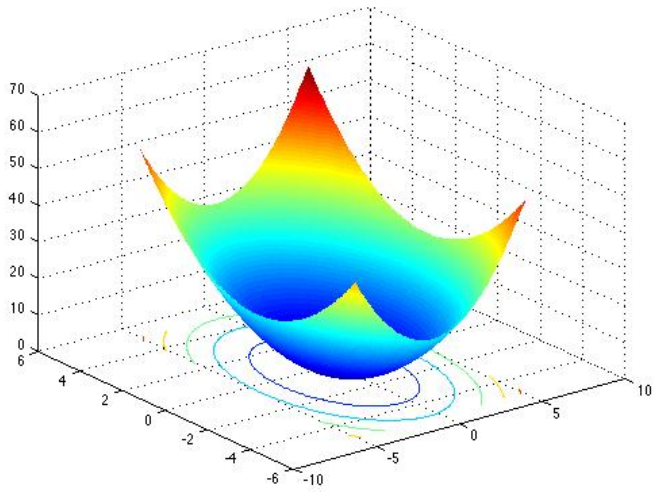


Şekil 4.8: Neumaier fonksiyonunun grafiksel gösterimi

4.9.Step Fonksiyonu

Fonksiyonun matematiksel ifadesi:

$$F_{step} = \sum_{i=1}^{D} \left| x_i + \frac{1}{2} \right|^2 \quad (9)$$



Şekil 4.9: Step fonksiyonu grafiksel gösterimi

5. UYGULAMA

Uygulama sonuçları dört farklı işlemci barındıran bilgisayardan alınmıştır. Testler yapılırken ölçeklenebilirlik açısından, işlem kanal sayısı değiştirilerek ve popülasyon boyutu değiştirilerek sonuçlar alınmıştır.

5.1.Test Sistemleri İşlemci Modelleri

Test sistemleri, imkan çerçevesinde, farklı sınıflardaki işlemciler arasından seçilmeye çalışılmıştır. İlk test işlemcisi, 2007 yılında çıkarılmış, diğerlerine göre eski bir işlemcidir. Diğer işlemciler ise, Intel firmasının üçüncü nesil işlemci serisi olarak ürettiği sınıftan, i3, i5 ve i7 olacak şekilde 3 işlemci seçilmiştir. Test sistemlerinin işlemcileri ve özellikleri şu şekildedir;

1. Intel® Core™2 Quad Processor Q6600 (8M Cache, 2.40 GHz, 1066 MHz FSB). Bu işlemci 2007 yılının ilk çeyreğinde üretilmiştir. (Q6600)
2. Intel® Core™ i3-370M Processor (3M cache, 2.40 GHz). Bu işlemci 2009 yılının üçüncü çeyreğinde üretilmiştir. (M370)
3. Intel® Core™ i5-460M Processor (3M Cache, 2.53 GHz). Bu işlemci 2009 yılının üçüncü çeyreğinde üretilmiştir. (M460)
4. Intel® Core™ i7-3720QM Processor (6M Cache, up to 3.60 GHz). Bu işlemci 2012 yılının ikinci çeyreğinde üretilmiştir. (3720QM)

5.2.Uygulama Özellikleri

Uygulamada, popülasyon boyutu 32 ve 64 olarak test edilmiştir. Kromozom uzunlukları bütün testlerde aynı ve 10 gen olarak uygulama gerçekleştirilmiştir. Çaprazlama olarak tek noktalı çaprazlama tercih edilmiş ve mutasyon için, rasgele alınan bir genin değerinde küçük bir değişiklik yapacak şekilde bir algoritma geliştirilmiştir.

Test sonuçları alınırken, tekrar sayısı, test sistemlerinin işlemci hızlarına göre belirlenmiştir ve sistemden sisteme değişiklik göstermektedir.

Yukarıda belirtilen değerler seçilirken, test sonuçlarının gözlenebilir olmasına dikkat edilmiştir ve ölçeklenebilir sonuçlar elde edilmiştir.

5.3.Uygulama Sonuçları

Uygulama sonuçları tek bir kriter fonksiyonu için detaylı olarak verilecek, diğer test sistemlerinin sadece süreleri verilecek.

Seri ve paralel performans artışlarında kullanılacak formül aşağıdaki şekildedir ;

$$\boldsymbol{S_{hızArtışı}} = \frac{\boldsymbol{T_s}}{\boldsymbol{T_p}} \tag{10}$$

$T_p = T_d + T_p + T_c$

$T_s = Seri\ çalışma\ zamanı$

$T_d = Veri\ dağıtım\ zamanı$

$T_c = Veri\ toplama\ zamanı$

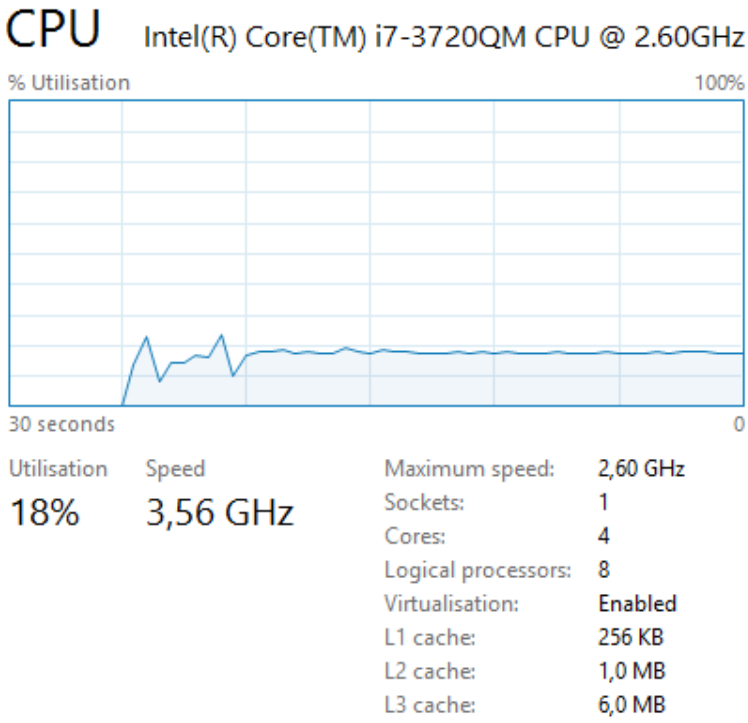
$T_p = Toplam\ paralel\ işlem\ süresi$

Bilgisayardaki test sonucunda alınan görev yöneticisi ekran görüntüleri şekil 5.1-5.8, şekil 5.18-5.23, şekil 5.33-5.38 ve şekil 5.48-5.53 verilmiştir. Ancak bütün fonksiyonların görev yöneticisi ekran görüntüleri eklenmeyecek, her bilgisayar için bir fonksiyondaki görev yöneticisi ekran görüntüleri verilecek ve diğer fonksiyonlar için çalışma zamanlarını gösteren ekran görüntüleri (şekil 5.9–5.17, 5.24–5.32, 5.39–5.47 ve 5.54–5.62) eklenecektir. En son olarak dört test sisteminin bütün fonksiyonlarda ve iki, dört ve eğer sistem destekliyorsa sekiz kanal çalışma zamanlarını tek bir grafik olarak verilecektir.

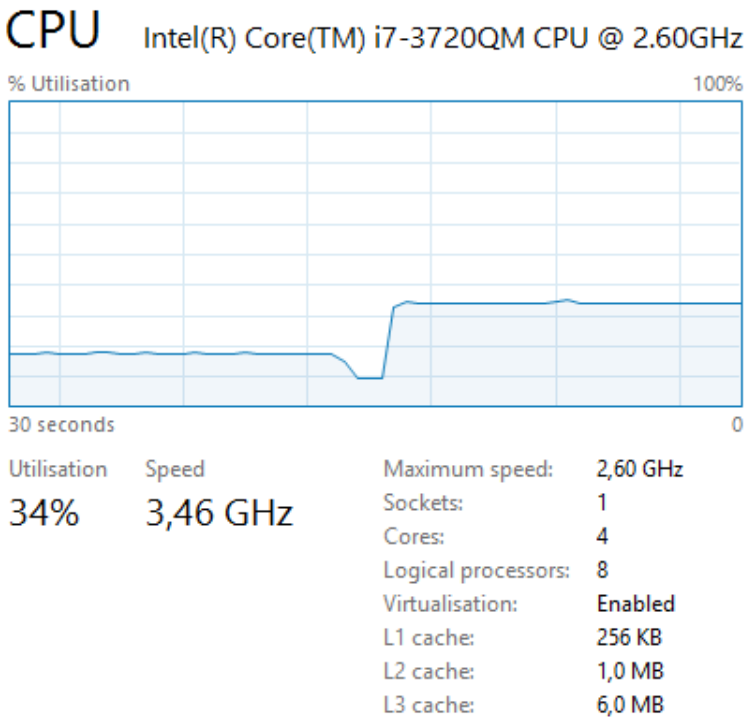
Test sonuçlarının bu şekilde verilmesinin sebebi, süreler ve oranlar değişmekle birlikte, bütün görev yöneticisi resimleri birbirine benzer çıkmaktadır. Şöyle ki, 2 çekirdek 4 kanal işlem yapabilen işlemcili sistemlerde yaklaşık olarak, seri çalışma sırasında %25, 2 kanallı paralel çalışma sırasında %50 ve 4 kanallı paralel programlama sırasında %100 kaynak kullanımı karakteristiği göstermektedir. Bu sebeple, buna örnek olarak bir kriter fonksiyonunun görev yöneticisi ekran görüntüsü verilmiştir.

5.3.1. Intel® Core™ i7-3720QM

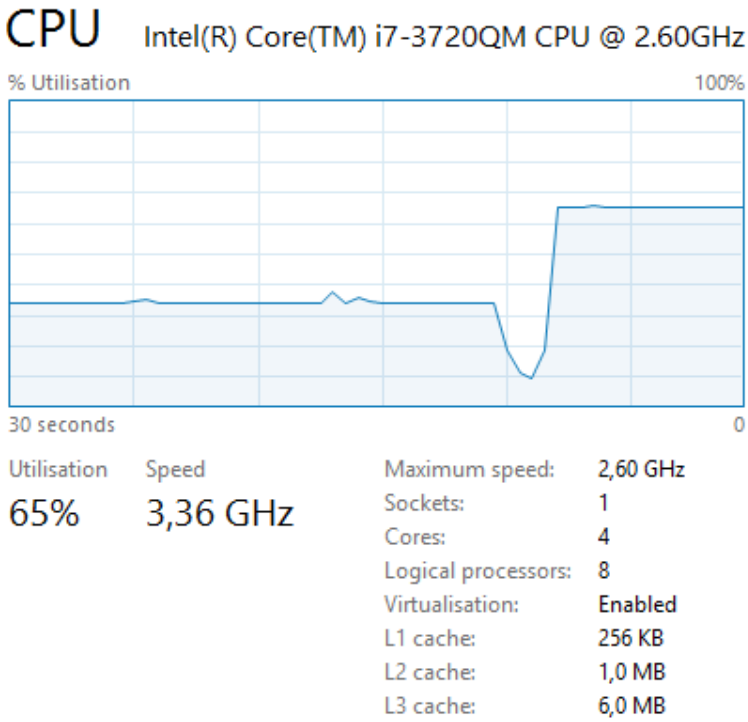
Bu işlemci, 4 gerçek çekirdek ve HT teknolojisi ile birlikte işletim sistemi tarafından 8 çekirdek olarak algılanan bir işlemcidir. Şekil 5.1-5.8' de, bu işlemcinin kaynak kullanımı görev yöneticisi ekran görüntüleri, örnek olarak bir fonksiyon için verilmekte ve bütün fonksiyonlar için çalışma zamanlarını gösteren ekran görüntüleri şekil 5.9-5.17' de verilmiştir.



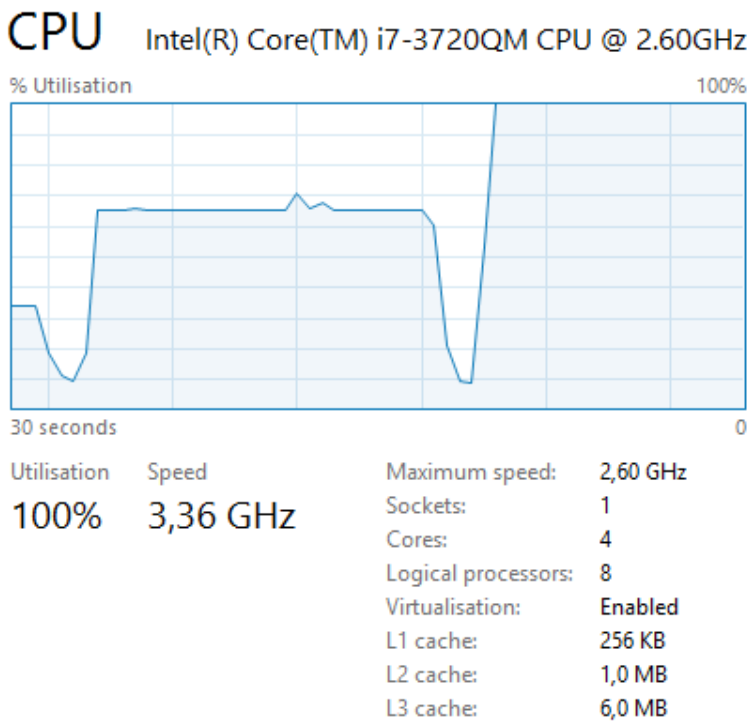
Şekil 5.1: Ackley fonksiyonu 32 popülasyon seri çalışma anı grafiği



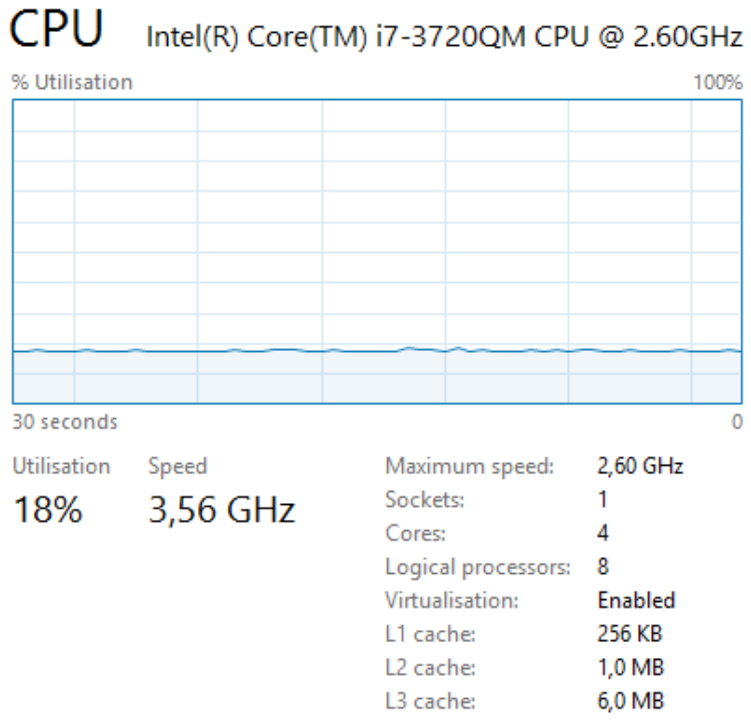
Şekil 5.2: Ackley fonksiyonu 32 popülasyon 2 kanal çalışma anı grafiği



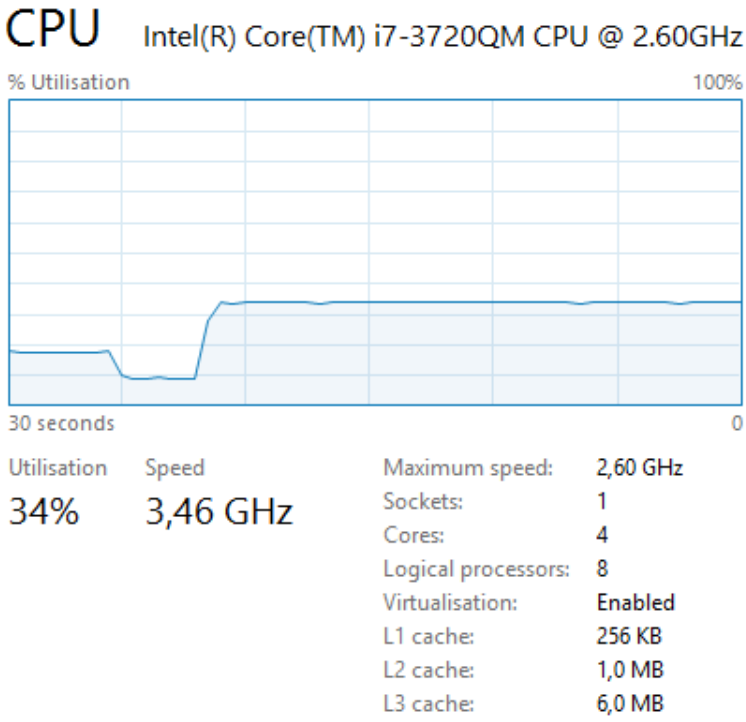
Şekil 5.3: Ackley fonksiyonu 32 popülasyon 4 kanal çalışma anı grafiği



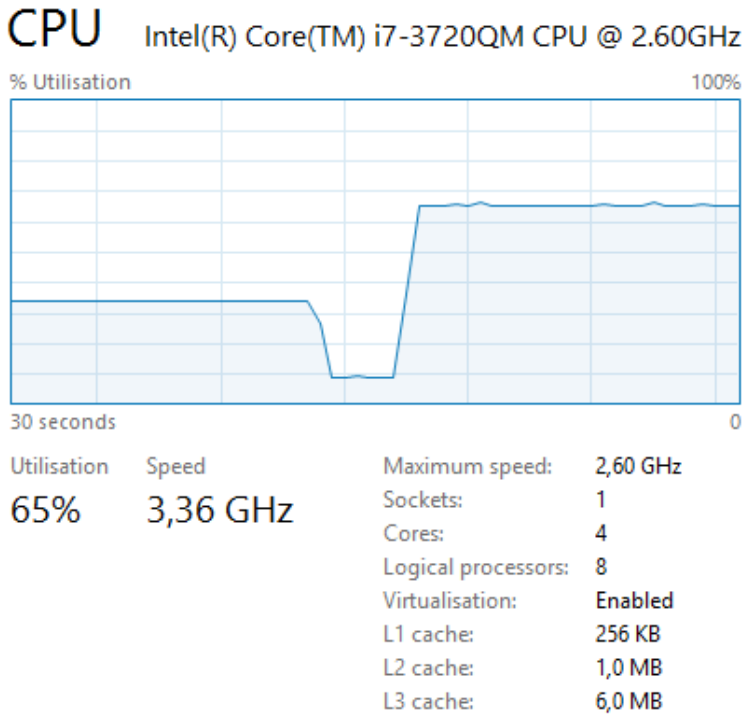
Şekil 5.4: Ackley fonksiyonu 32 popülasyon 8 kanal çalışma anı grafiği



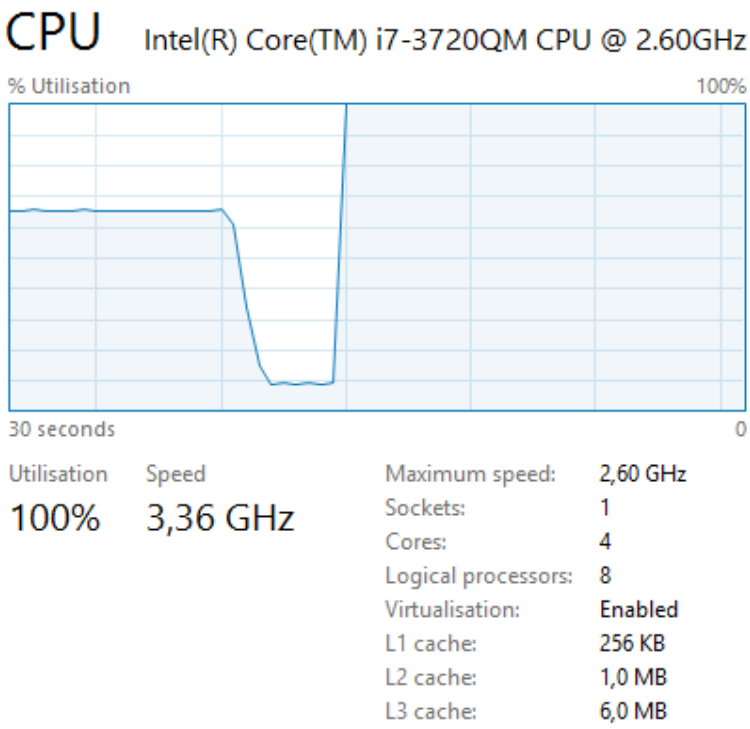
Şekil 5.5: Ackley fonksiyonu 64 popülasyon seri çalışma anı grafiği



Şekil 5.6: Ackley fonksiyonu 64 popülasyon 2 kanal çalışma anı grafiği



Şekil 5.7: Ackley fonksiyonu 64 popülasyon 4 kanal çalışma anı grafiği



Şekil 5.8: Ackley fonksiyonu 64 popülasyon 8 kanal çalışma anı grafiği

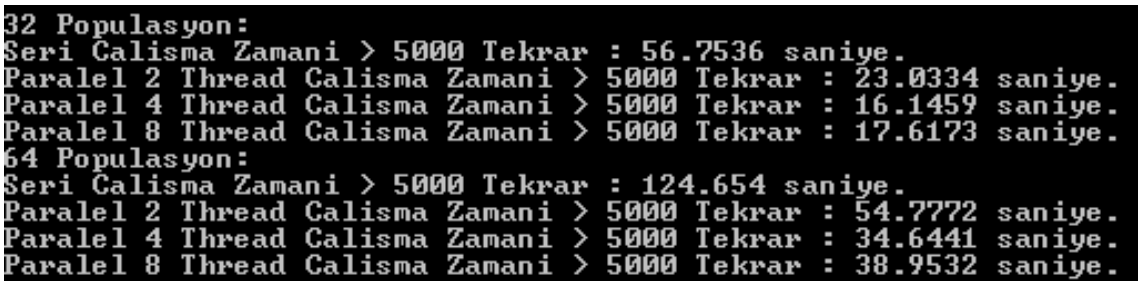
Şekil 5.9-5.17'de, i7-3720QM işlemcisinin bütün kriter fonksiyonlarındaki çalışma zamanlarını gösteren, uygulama çıktısı ekran görüntüleri verilmiştir.

```
32 Populasyon:
Seri Calisma Zamani > 5000 Tekrar : 50.4126 saniye.
Paralel 2 Thread Calisma Zamani > 5000 Tekrar : 23.3462 saniye.
Paralel 4 Thread Calisma Zamani > 5000 Tekrar : 15.9119 saniye.
Paralel 8 Thread Calisma Zamani > 5000 Tekrar : 17.1703 saniye.
64 Populasyon:
Seri Calisma Zamani > 5000 Tekrar : 92.7578 saniye.
Paralel 2 Thread Calisma Zamani > 5000 Tekrar : 44.1442 saniye.
Paralel 4 Thread Calisma Zamani > 5000 Tekrar : 32.3723 saniye.
Paralel 8 Thread Calisma Zamani > 5000 Tekrar : 35.3884 saniye.
```

Şekil 5.9: Ackley fonksiyonu çalışma zamanları

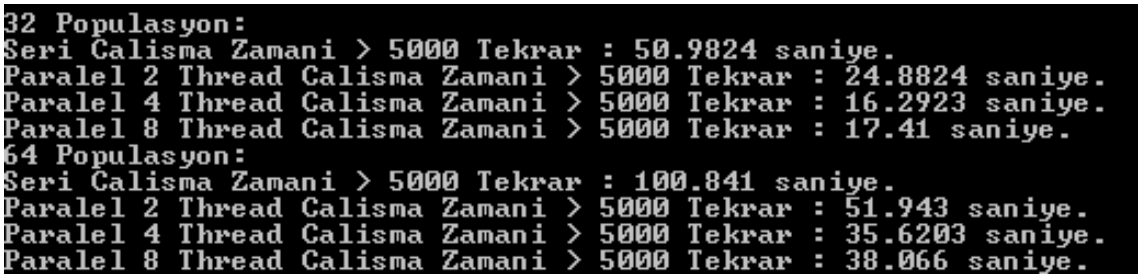
```
32 Populasyon:
Seri Calisma Zamani > 5000 Tekrar : 56.382 saniye.
Paralel 2 Thread Calisma Zamani > 5000 Tekrar : 23.0111 saniye.
Paralel 4 Thread Calisma Zamani > 5000 Tekrar : 16.1349 saniye.
Paralel 8 Thread Calisma Zamani > 5000 Tekrar : 17.5155 saniye.
64 Populasyon:
Seri Calisma Zamani > 5000 Tekrar : 123.331 saniye.
Paralel 2 Thread Calisma Zamani > 5000 Tekrar : 53.8671 saniye.
Paralel 4 Thread Calisma Zamani > 5000 Tekrar : 34.128 saniye.
Paralel 8 Thread Calisma Zamani > 5000 Tekrar : 37.3301 saniye.
```

Şekil 5.10: Kosinüs Karma Fonksiyonu çalışma zamanları

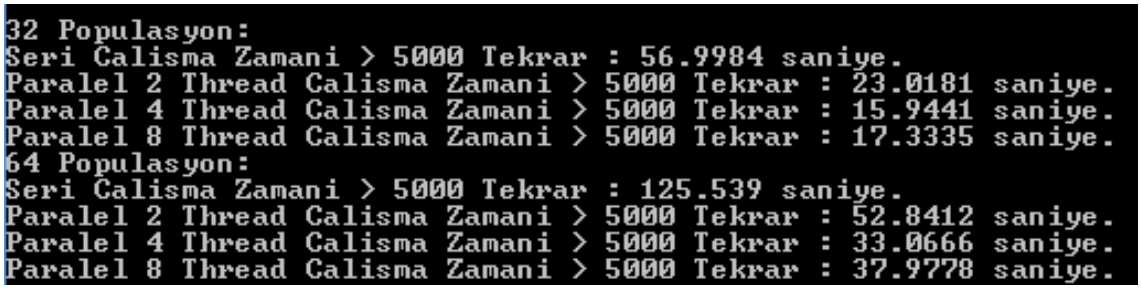


Şekil 5.11: Hiper Elipsoid Fonksiyonu çalışma zamanları

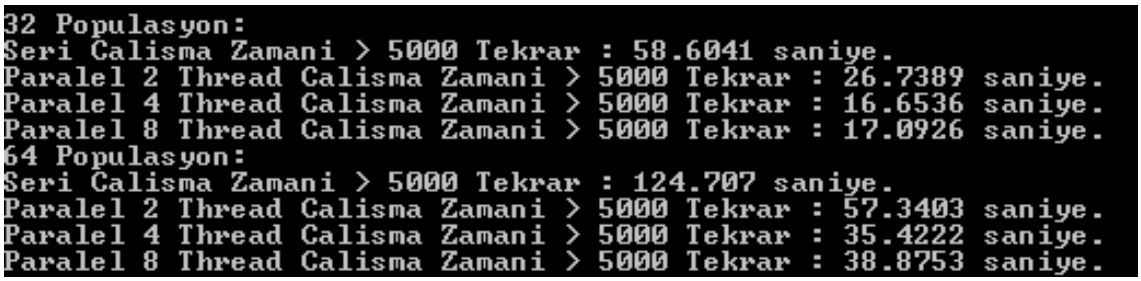
Şekil 5.12: Üssel fonksiyon çalışma zamanları



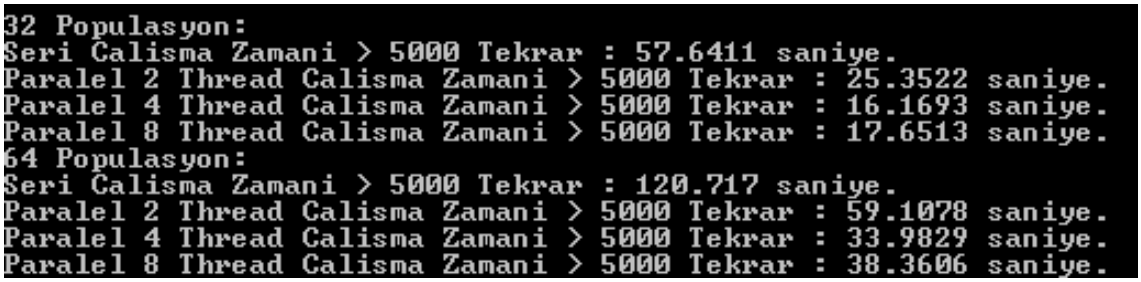
Şekil 5.13: Griewank fonksiyonu çalışma zamanları



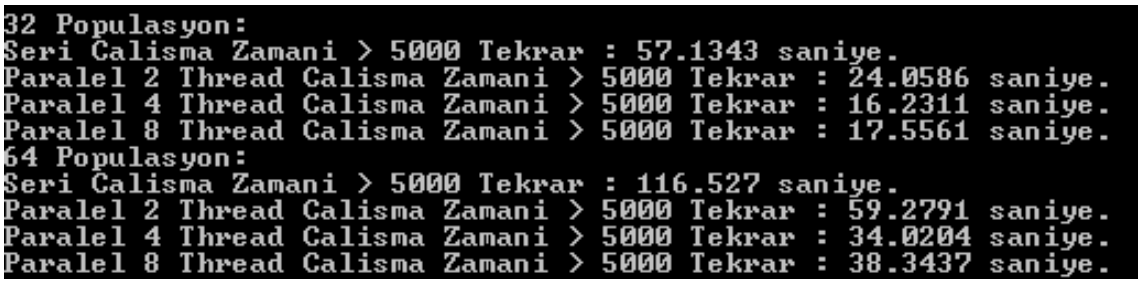
Şekil 5.14: Neumaier fonksiyonu çalışma zamanları



Şekil 5.15: Rastrigin fonksiyonu çalışma zamanları



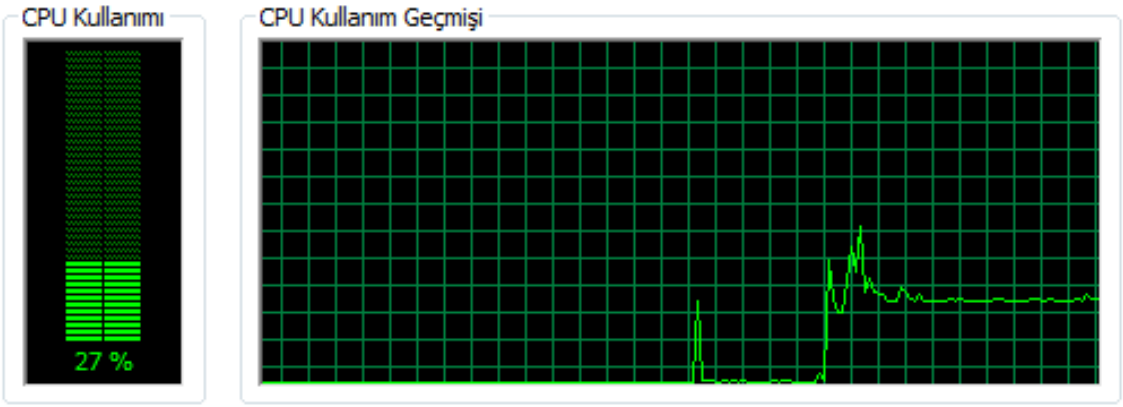
Şekil 5.16: Sphere fonksiyonu çalışma zamanları



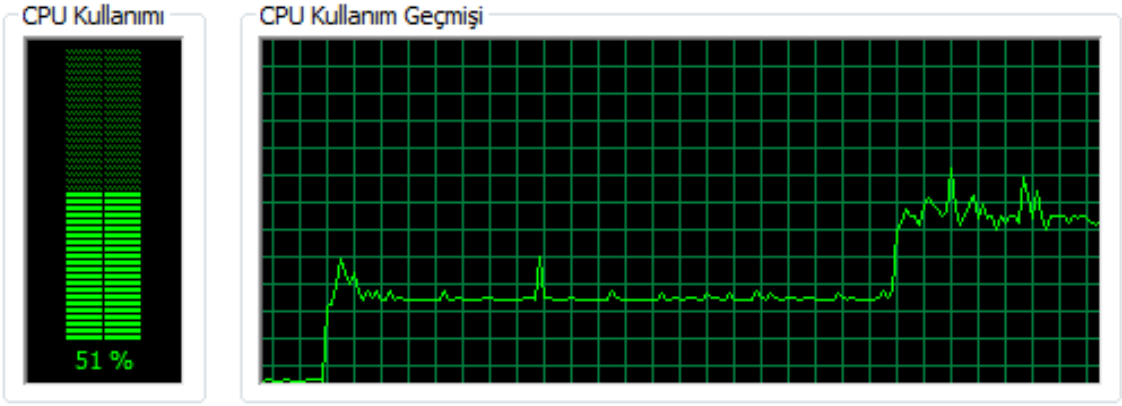
Şekil 5.17: Step fonksiyonu çalışma zamanları

5.3.2. Intel (R) Core(TM) i5 CPU M 460

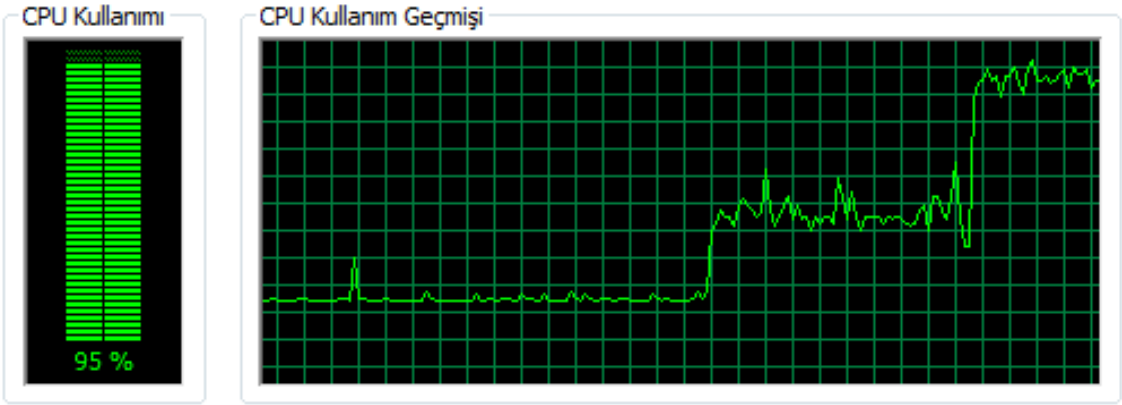
Bu işlemci, 2 gerçek çekirdek ve HT teknolojisi ile birlikte işletim sistemi tarafından 4 çekirdek olarak algılanan bir işlemcidir. Şekil 5.18-5.23' de, bu işlemcinin kaynak kullanımı görev yöneticisi ekran görüntüleri, örnek olarak bir fonksiyon için verilmekte ve bütün fonksiyonlar için çalışma zamanlarını gösteren ekran görüntüleri şekil 5.24-5.32' de verilmiştir.



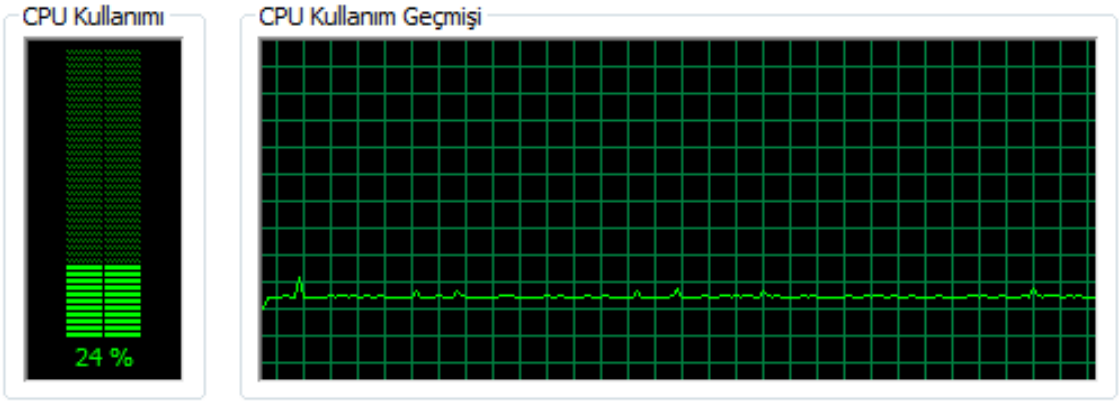
Şekil 5.18: Ackeley fonksiyonu 32 popülasyon seri çalışma ani grafiği



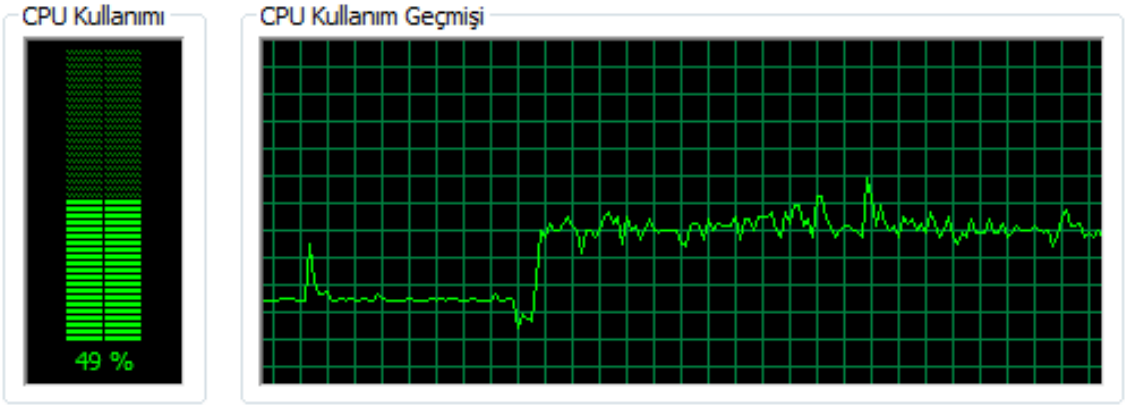
Şekil 5.19: Ackeley fonksiyonu 32 popülasyon 2 kanal paralel çalışma ani grafiği



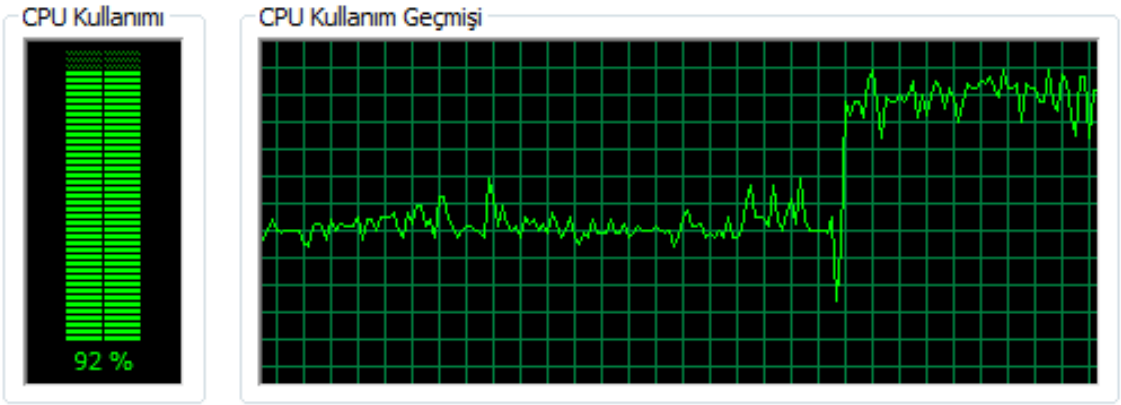
Şekil 5.20: Ackeley fonksiyonu 32 popülasyon 4 kanal paralel çalışma ani grafiği



Şekil 5.21: Ackeley fonksiyonu 64 popülasyon seri çalışma ani grafiği

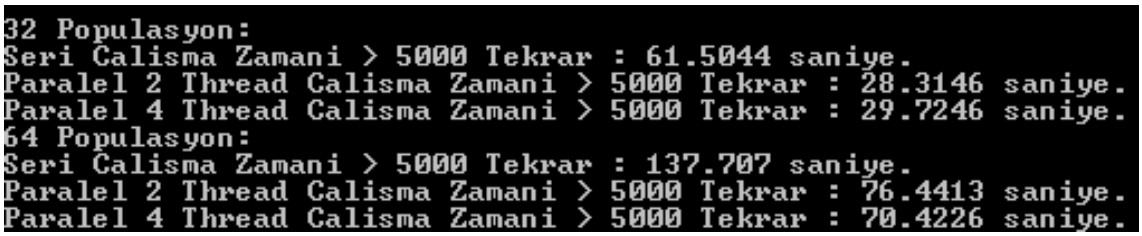


Şekil 5.22: Ackeley fonksiyonu 64 popülasyon 2 kanal paralel çalışma ani grafiği

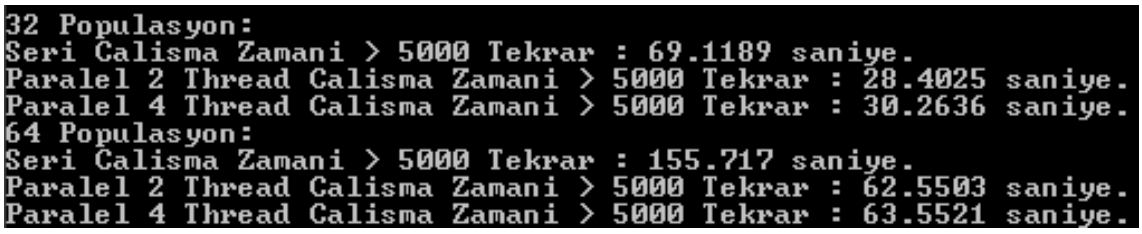


Şekil 5.23: Ackeley fonksiyonu 64 popülasyon 4 kanal paralel çalışma ani grafiği

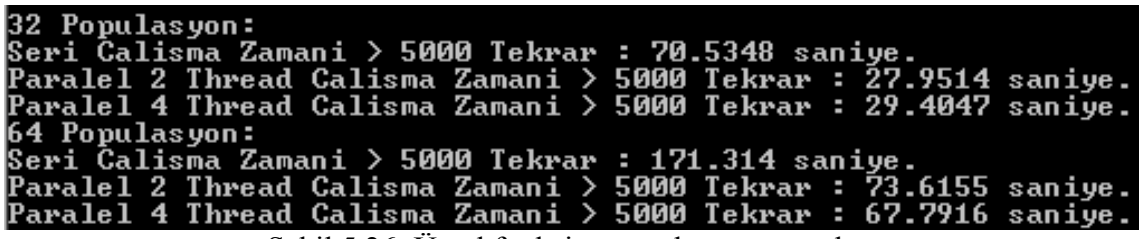
Şekil 5.24-5.32'de, i5-M 460 işlemcisinin bütün kriter fonksiyonlarındaki çalışma zamanlarını gösteren, uygulama çıktısı ekran görüntüleri verilmiştir.



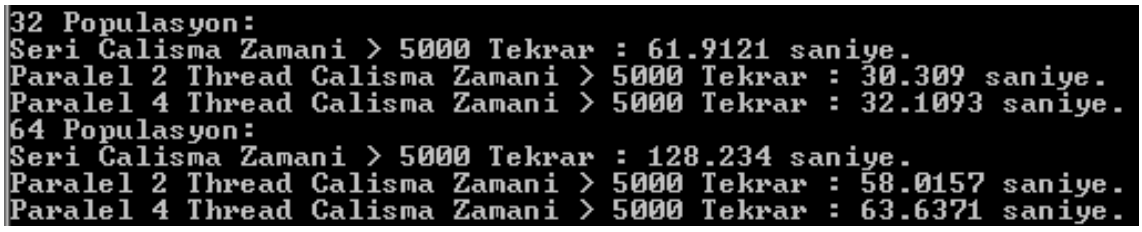
Şekil 5.24: Ackeley fonksiyonu çalışma zamanları



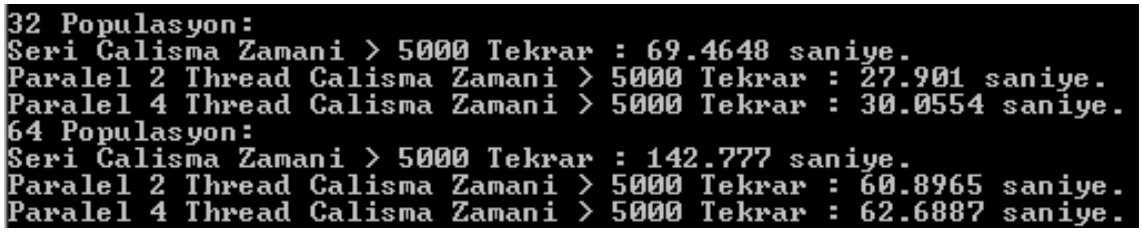
Şekil 5.25: Kosinüs karma fonksiyonu çalışma zamanları



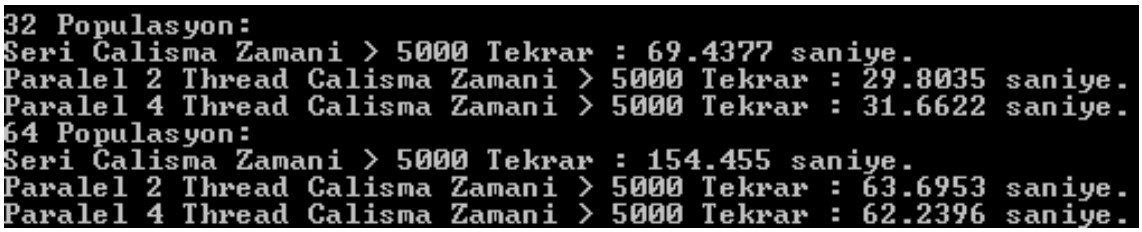
Şekil 5.26: Üssel fonksiyonu çalışma zamanları



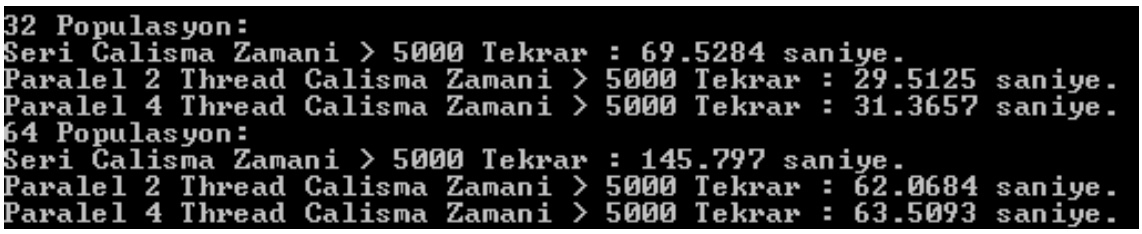
Şekil 5.27: Griewank fonksiyonu çalışma zamanları



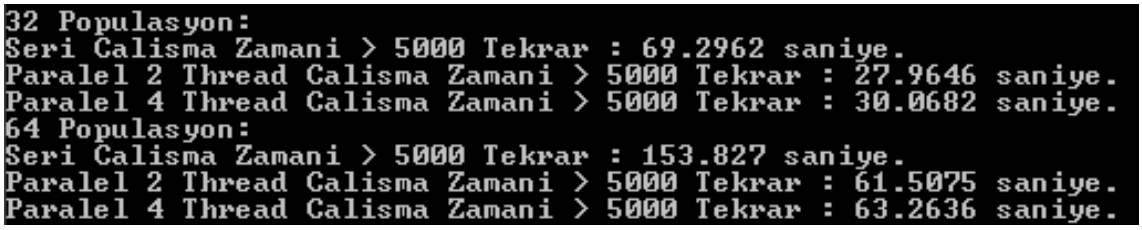
Şekil 5.28: Hiper elipsoid fonksiyonu çalışma zamanları



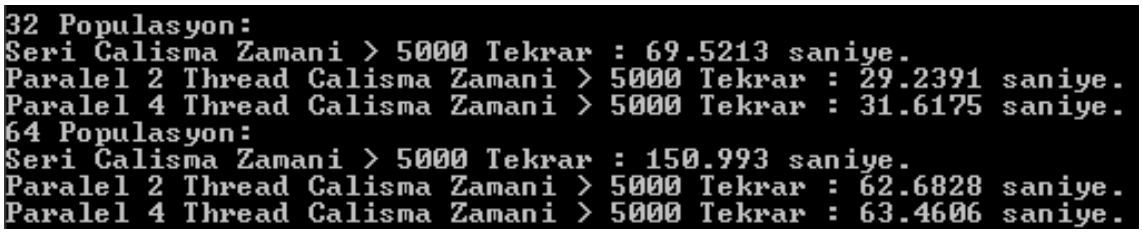
Şekil 5.29: Neumaier fonksiyonu çalışma zamanları



Şekil 5.30: Rastrigin fonksiyonu çalışma zamanları



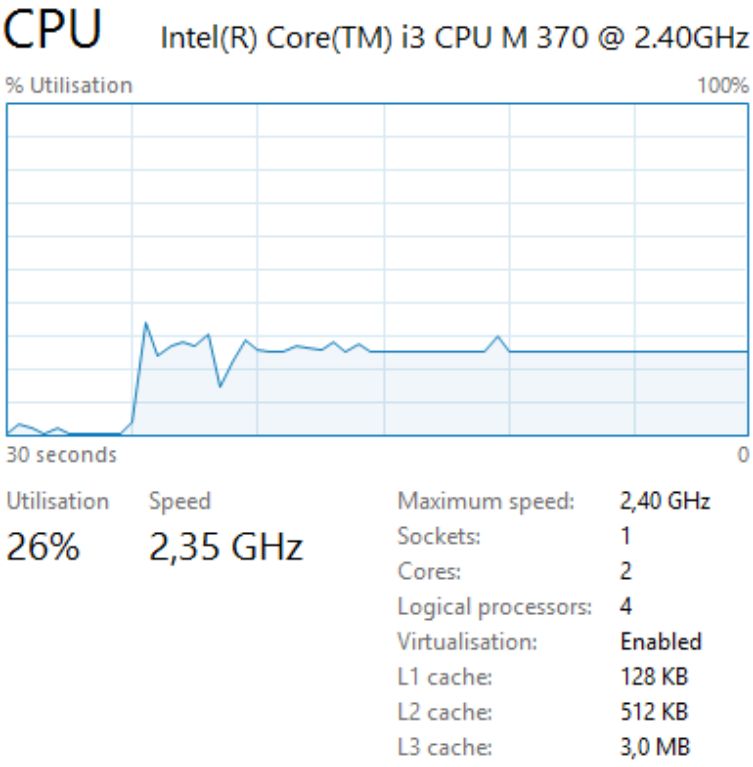
Şekil 5.31: Sphere fonksiyonu çalışma zamanları



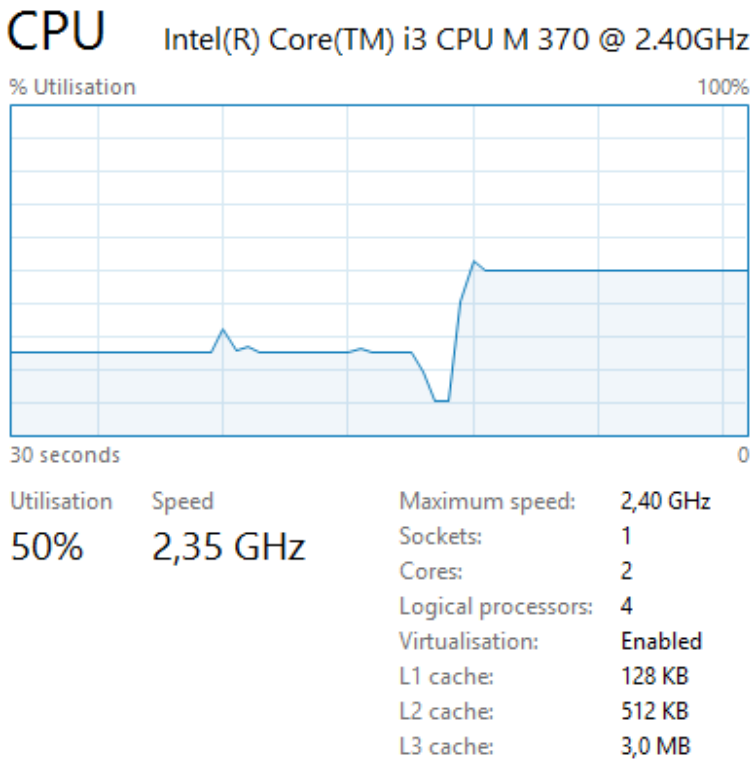
Şekil 5.32: Step fonksiyonu çalışma zamanları

5.3.3. Intel(R) Core (TM) i3 CPU M 370

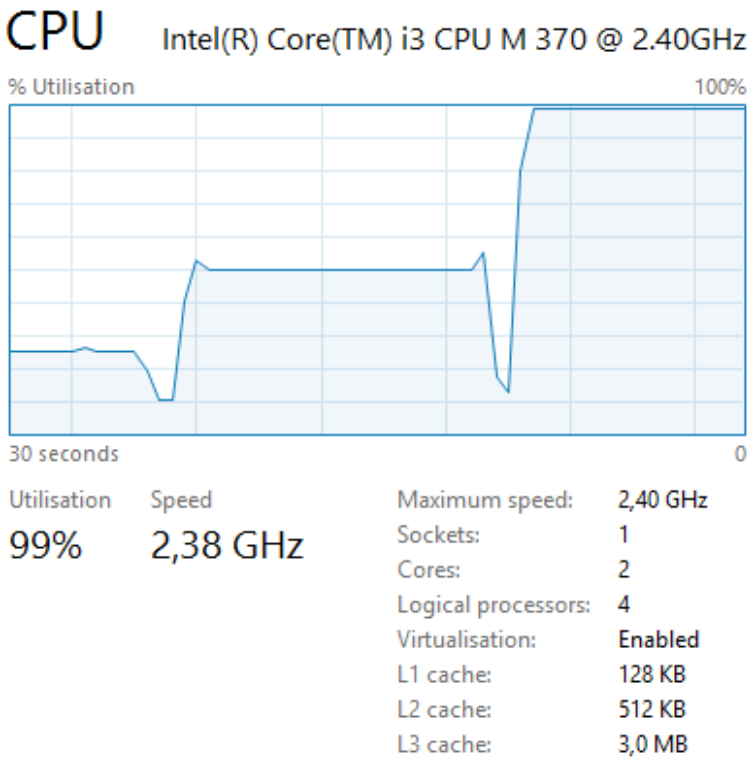
Bu işlemci, 2 gerçek çekirdek ve HT teknolojisi ile birlikte işletim sistemi tarafından 4 çekirdek olarak algılanan bir işlemcidir. Şekil 5.33-5.38' de, bu işlemcinin kaynak kullanımı görev yöneticisi ekran görüntüleri, örnek olarak bir fonksiyon için verilmekte ve bütün fonksiyonlar için çalışma zamanlarını gösteren ekran görüntüleri şekil 5.39-5.47' de verilmiştir.



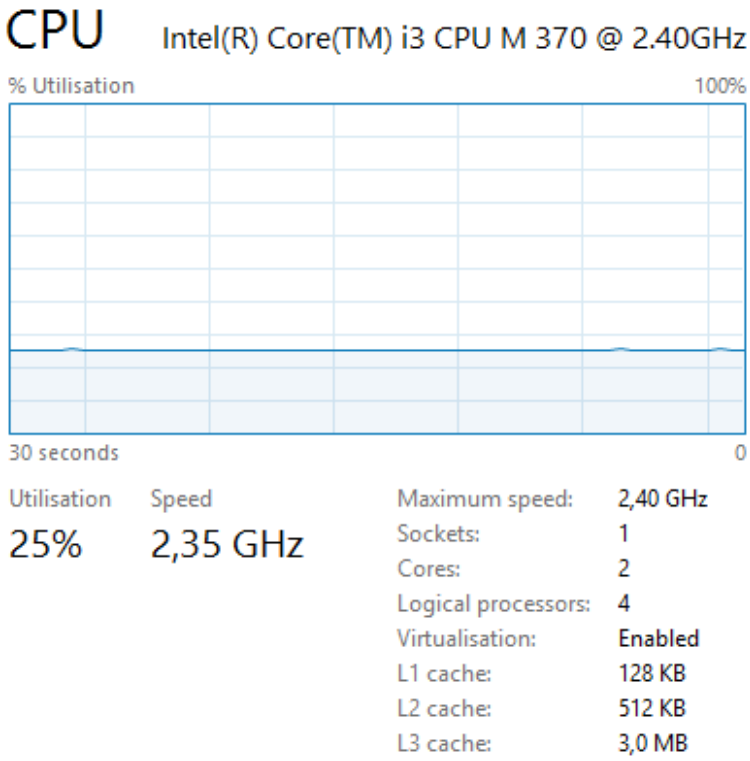
Şekil 5.33: Ackeley fonksiyonu 32 popülasyon seri çalışma ani grafiği



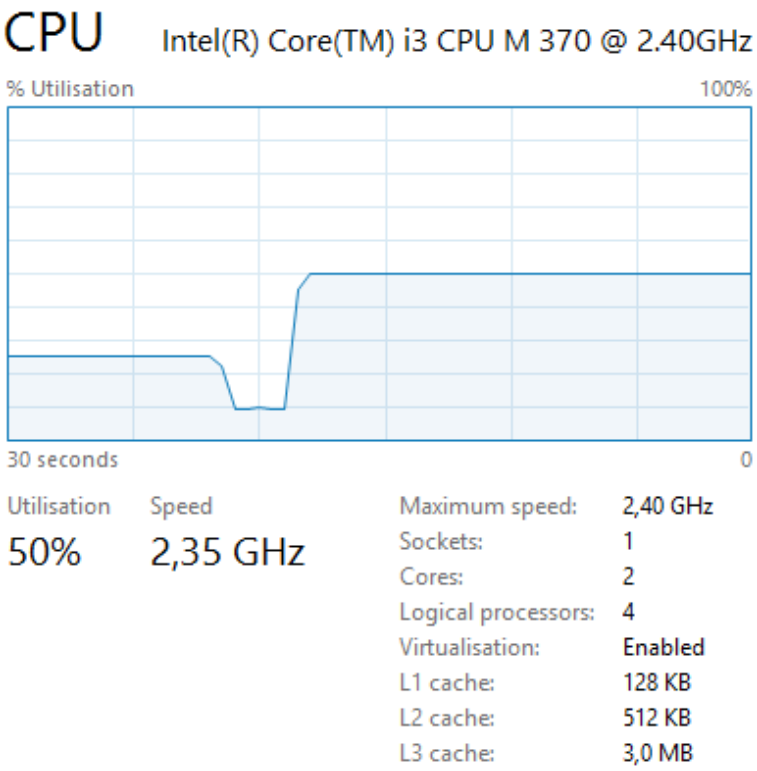
Şekil 5.34: Ackeley fonksiyonu 32 popülasyon 2 kanal paralel çalışma ani grafiği



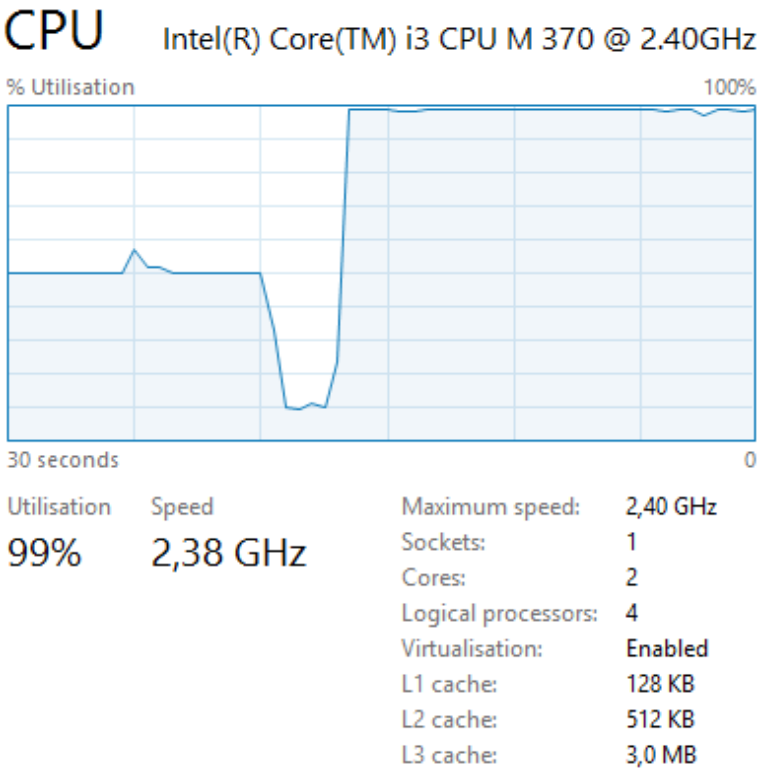
Şekil 5.35: Ackeley fonksiyonu 32 popülasyon 4 kanal paralel çalışma ani grafiği



Şekil 5.36: Ackeley fonksiyonu 64 popülasyon seri çalışma ani grafiği

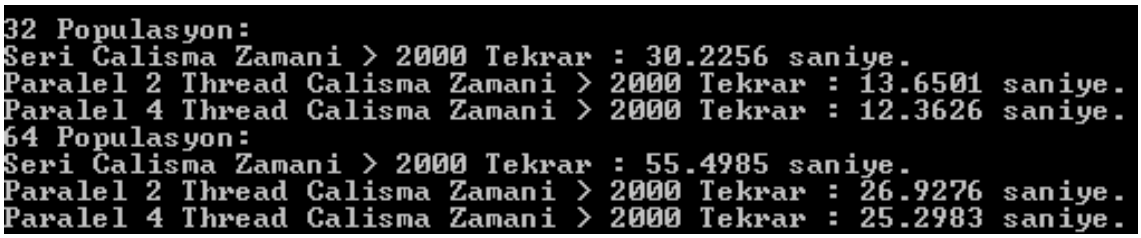


Şekil 5.37: Ackeley fonksiyonu 64 popülasyon 2 kanal paralel çalışma ani grafiği

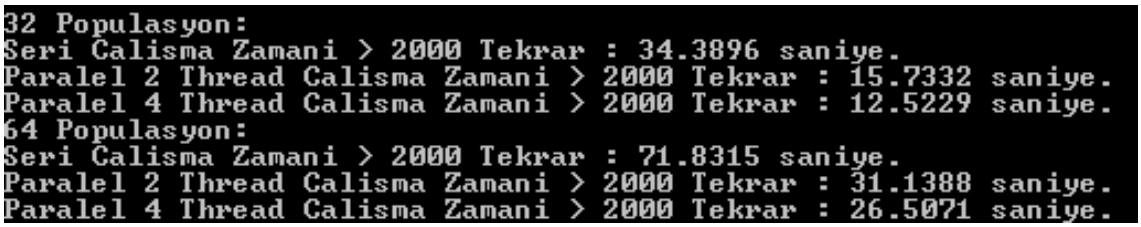


Şekil 5.38: Ackeley fonksiyonu 64 popülasyon 4 kanal paralel çalışma ani grafiği

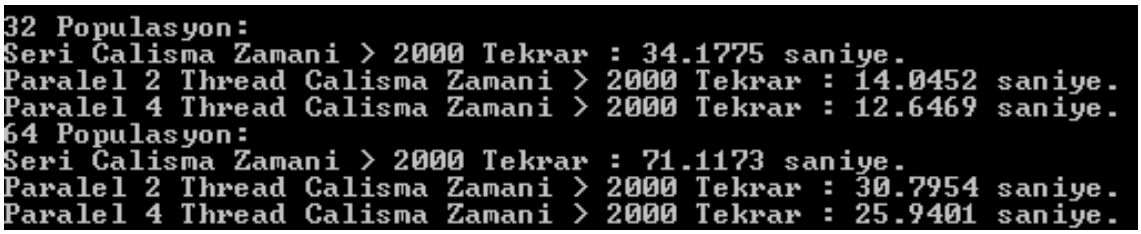
Şekil 5.39-5.47'de, i3-M 370 işlemcisinin bütün kriter fonksiyonlarındaki çalışma zamanlarını gösteren, uygulama çıktısı ekran görüntüleri verilmiştir.



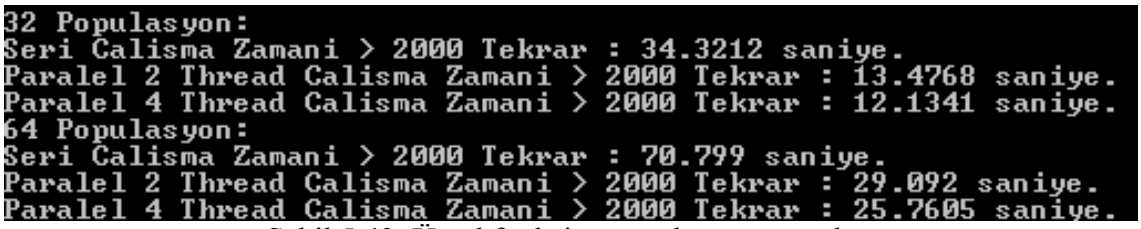
Şekil 5.39: Ackeley fonksiyonu çalışma zamanları



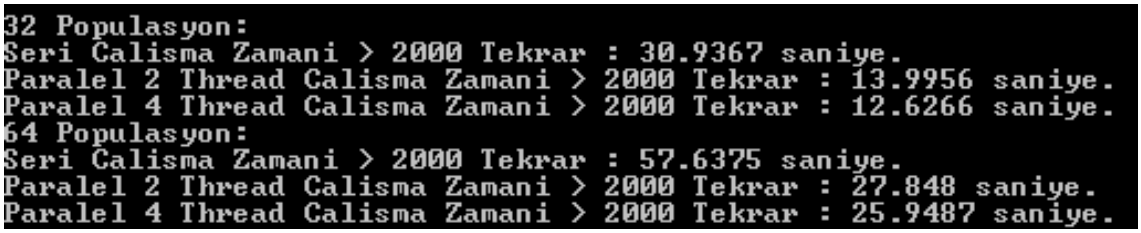
Şekil 5.40: Kosinüs Karma fonksiyonu çalışma zamanları



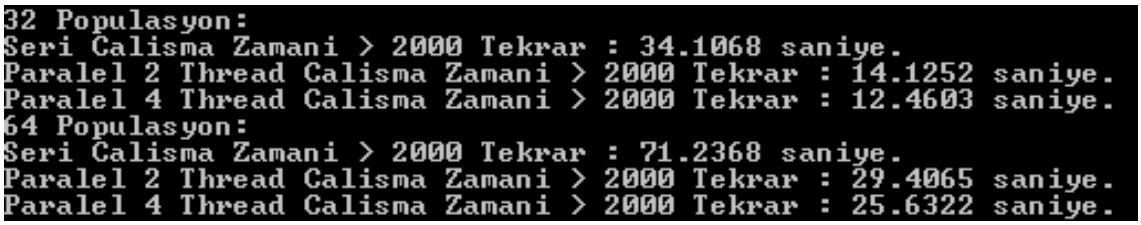
Şekil 5.41: Hiper Elipsoid fonksiyonu çalışma zamanları



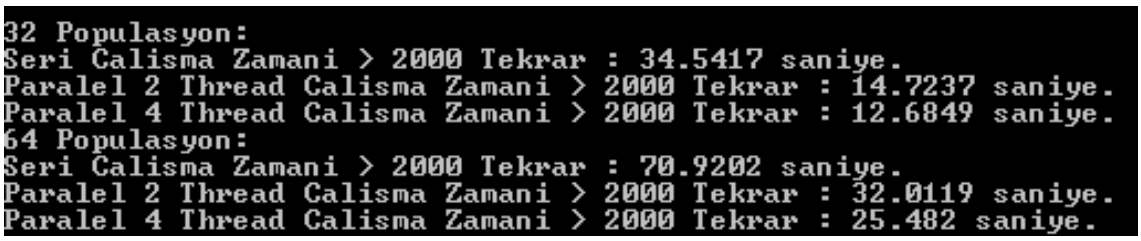
Şekil 5.42: Üssel fonksiyonu çalışma zamanları



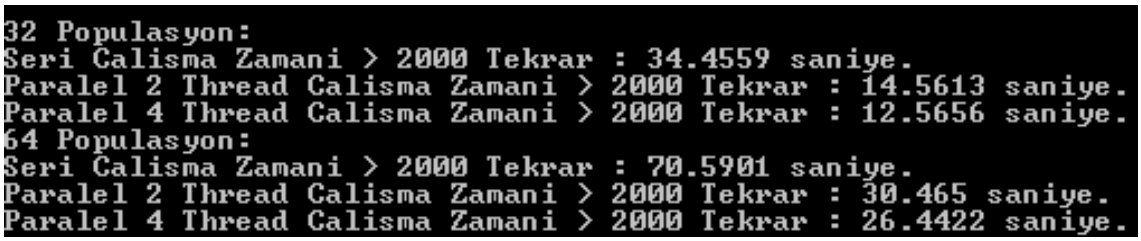
Şekil 5.43: Griewank fonksiyonu çalışma zamanları



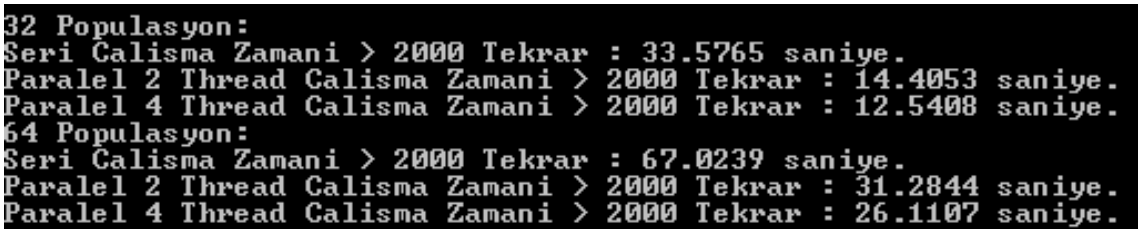
Şekil 5.44: Neumaier fonksiyonu çalışma zamanları



Şekil 5.45: Rastrigin fonksiyonu çalışma zamanları



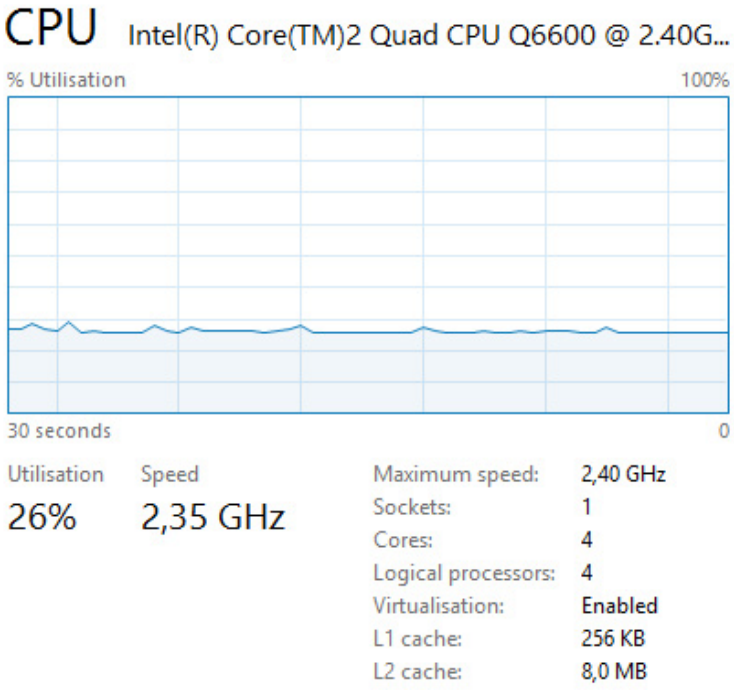
Şekil 5.46: Sphere fonksiyonu çalışma zamanları



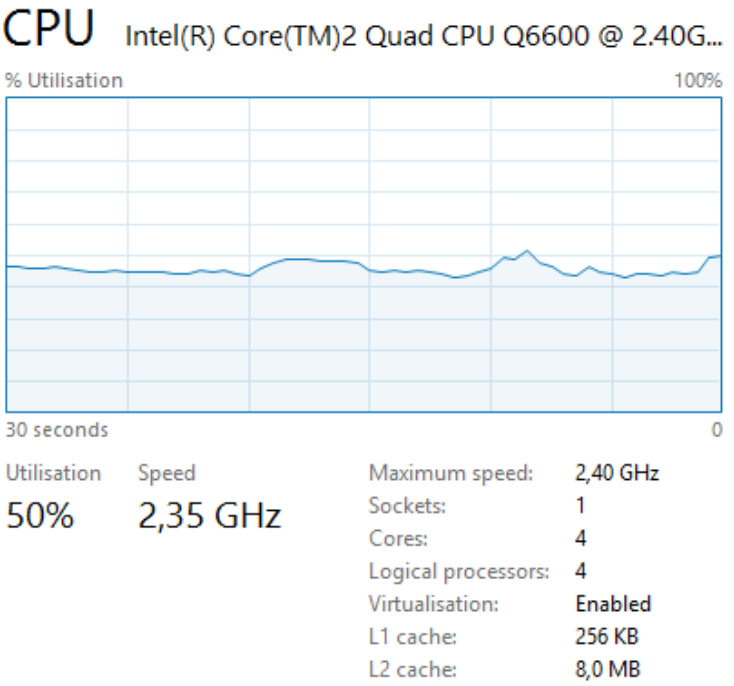
Şekil 5.47: Step fonksiyonu çalışma zamanları

5.3.4. Intel(R) Core (TM)2 Quad CPU Q6600

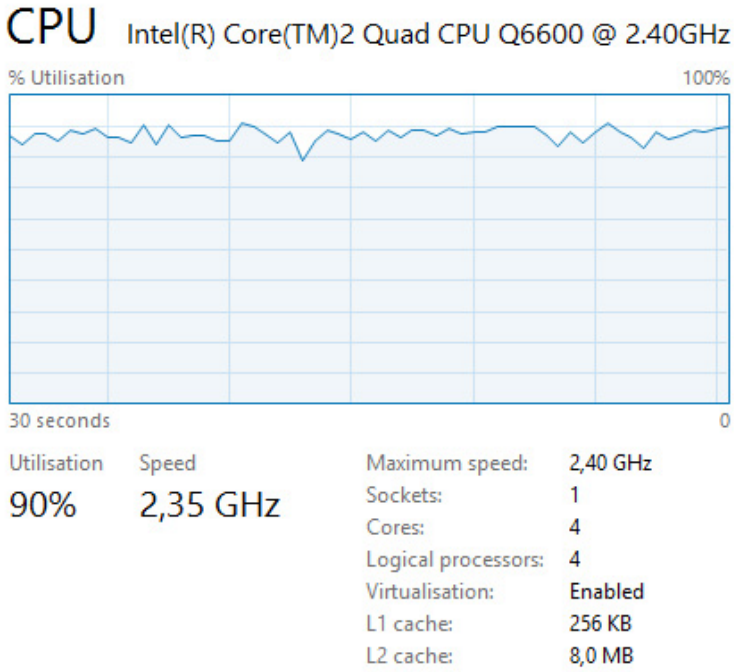
Bu işlemci, 4 gerçek çekirdek barındıran, HT teknolojisi bulundurmayan ve doğal olarak işletim sistemi tarafından da 4 çekirdek olarak algılanan bir işlemcidir. Şekil 5.48-5.53' de, bu işlemcinin kaynak kullanımı görev yöneticisi ekran görüntüleri, örnek olarak bir fonksiyon için verilmekte ve bütün fonksiyonlar için çalışma zamanlarını gösteren ekran görüntüleri şekil 5.54-5.62' de verilmiştir.



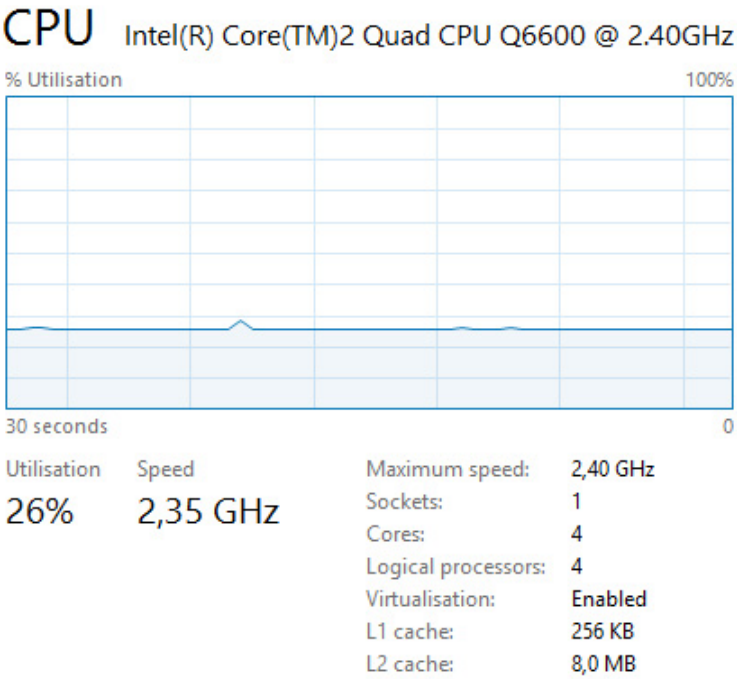
Şekil 5.48: Ackeley fonksiyonu 32 popülasyon seri çalışma ani grafiği



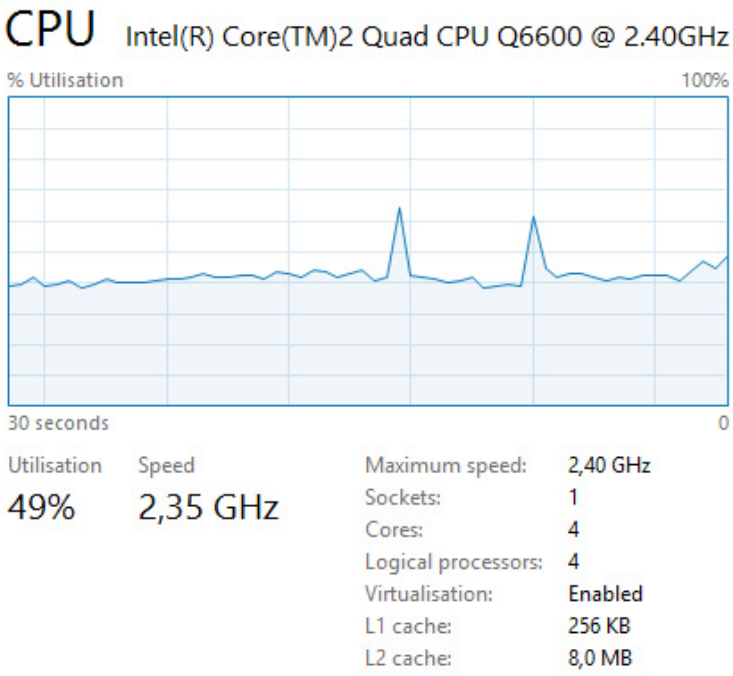
Şekil 5.49: Ackeley fonksiyonu 32 popülasyon 2 kanal paralel çalışma ani grafiği



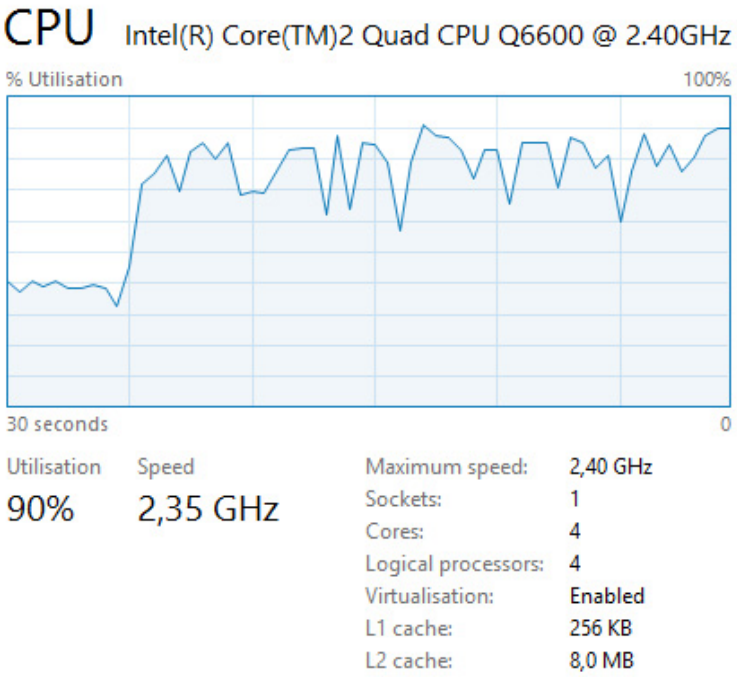
Şekil 5.50: Ackeley fonksiyonu 32 popülasyon 4 kanal paralel çalışma ani grafiği



Şekil 5.51: Ackeley fonksiyonu 64 popülasyon seri çalışma ani grafiği

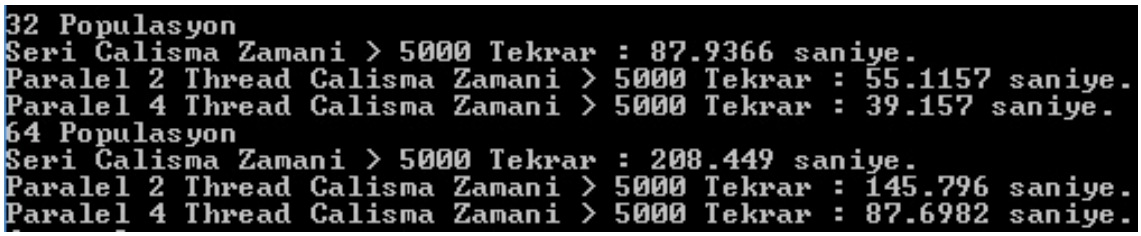


Şekil 5.52: Ackeley fonksiyonu 64 popülasyon 2 kanal paralel çalışma ani grafiği

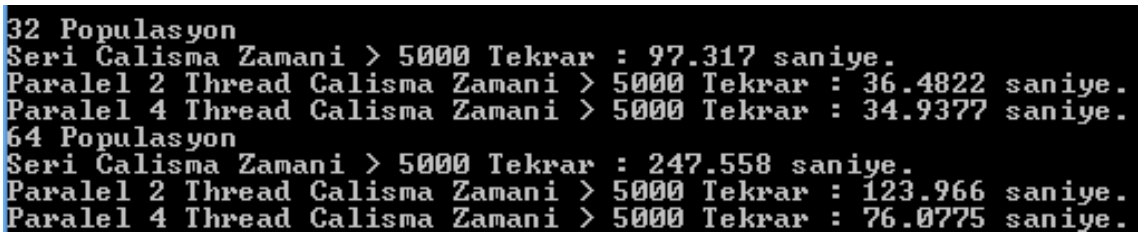


Şekil 5.53: Ackeley fonksiyonu 64 popülasyon 4 kanal paralel çalışma ani grafiği

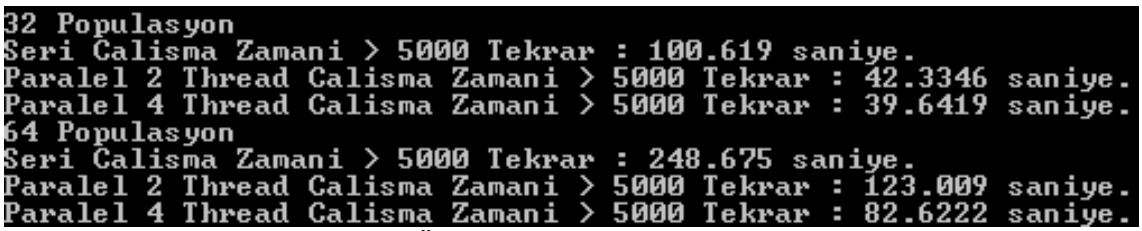
Şekil 5.54-5.62'de, Q6600 işlemcisinin bütün kriter fonksiyonlarındaki çalışma zamanlarını gösteren, uygulama çıktısı ekran görüntüleri verilmiştir.



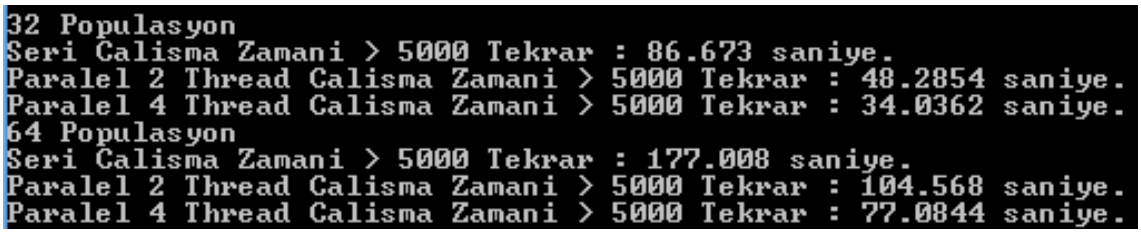
Şekil 5.54: Ackeley fonksiyonu çalışma zamanları



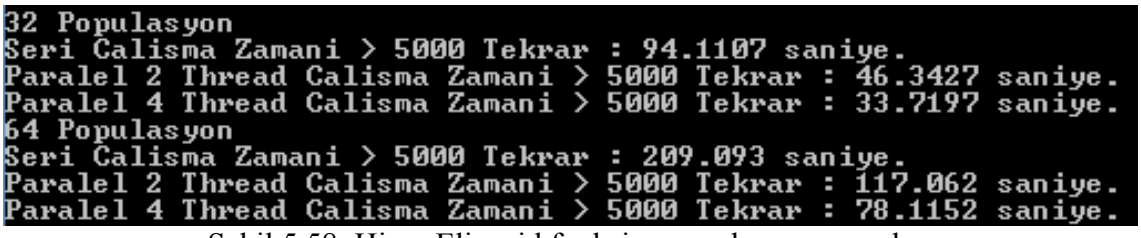
Şekil 5.55: Kosinüs Karma fonksiyonu çalışma zamanları



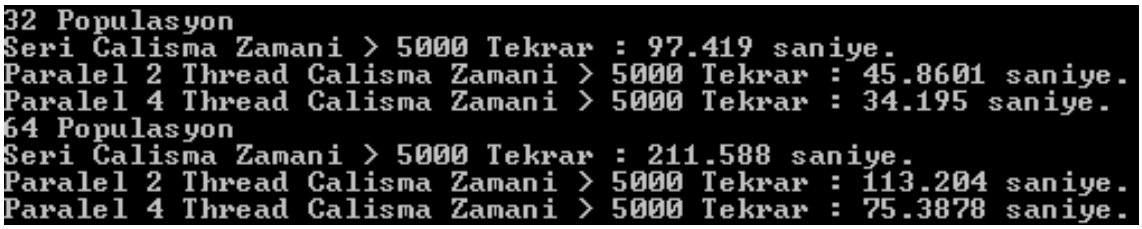
Şekil 5.56: Üssel fonksiyonu çalışma zamanları



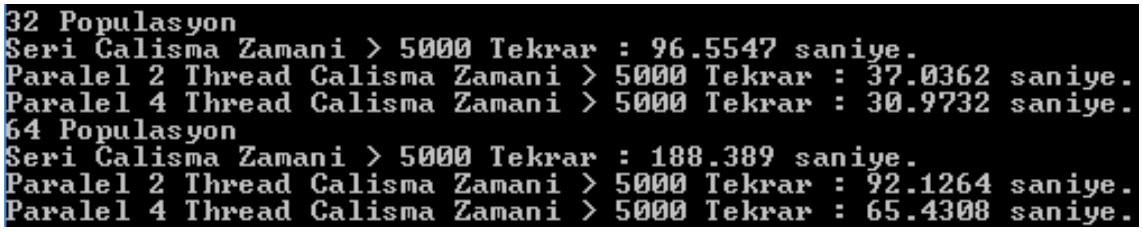
Şekil 5.57: Griewank fonksiyonu çalışma zamanları



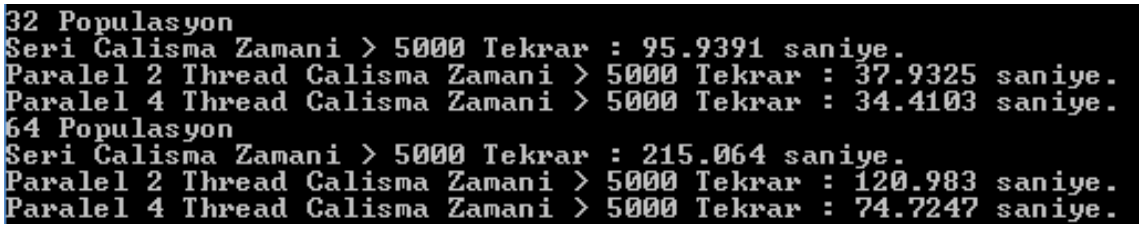
Şekil 5.58: Hiper Elipsoid fonksiyonu çalışma zamanları



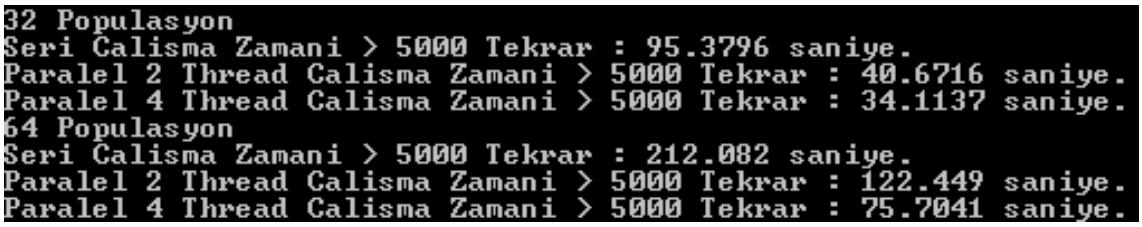
Şekil 5.59: Neumaier fonksiyonu çalışma zamanları



Şekil 5.60: Rastrigin fonksiyonu çalışma zamanları



Şekil 5.61: Sphere fonksiyonu çalışma zamanları



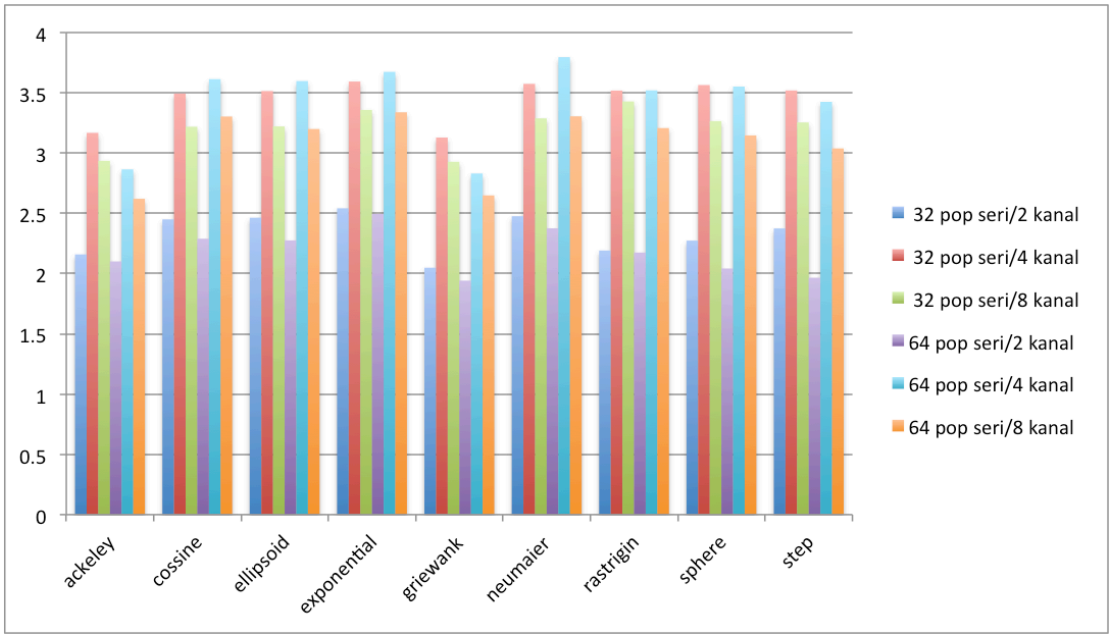
Şekil 5.62: Step fonksiyonu çalışma zamanları

5.4.Uygulama Sonuçlarının Değerlendirilmesi

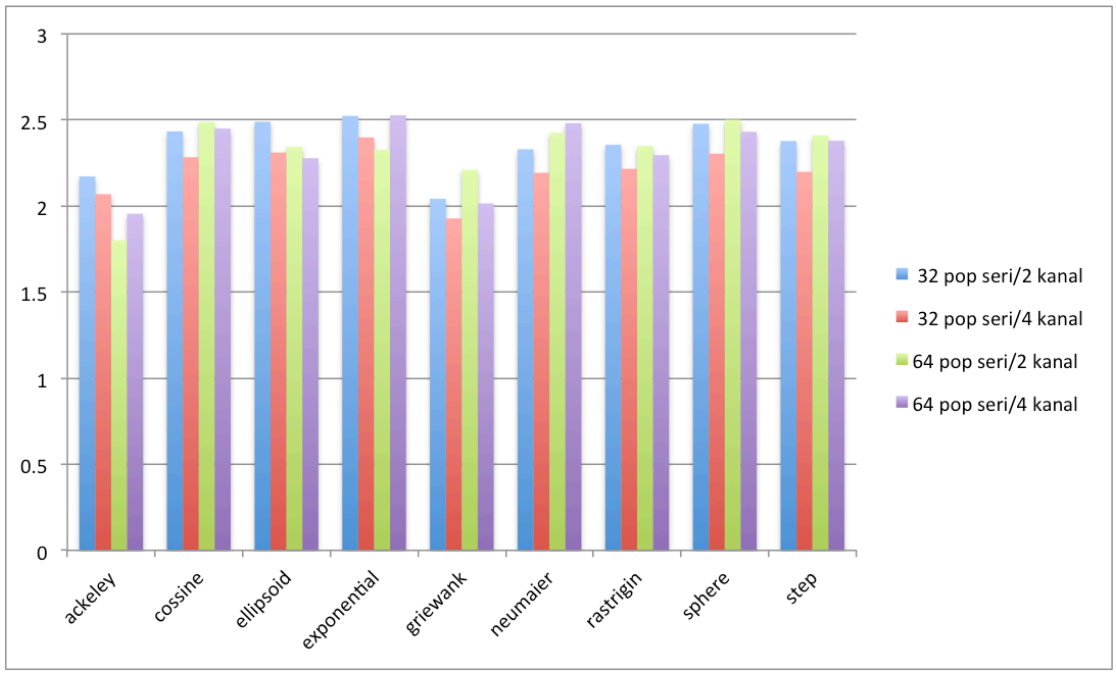
Bütün ekran görüntülerinden görülebileceği üzere, tezde geliştirilen uygulama ölçeklenebilir bir uygulamadır. Yani verinin belli bir oranda değiştirilmesi sonucu, işlem süreside yaklaşık olarak benzer oranda değişmektedir. Aynı şekilde, veri boyutunun sabit tutulup, kanal sayısı arttırılmıştır. Bunun sonucu olarak bütün test sistemlerinde benzer sonuçlar alınmıştır. Ancak HT bulunduran sistemlerde bu ölçeklenebilirlik, HT bölümlerde gözlenememekte, hatta veri dağıtma ve veri toplama

sürelerinden dolayı süre artmaktadır. Bunun sebebi, bölüm 4 de anlatıldığı gibi, HT'nin gerçek çekirdek olmayıp, bir sanal çekirdek teknolojisi olmasından kaynaklanmaktadır.

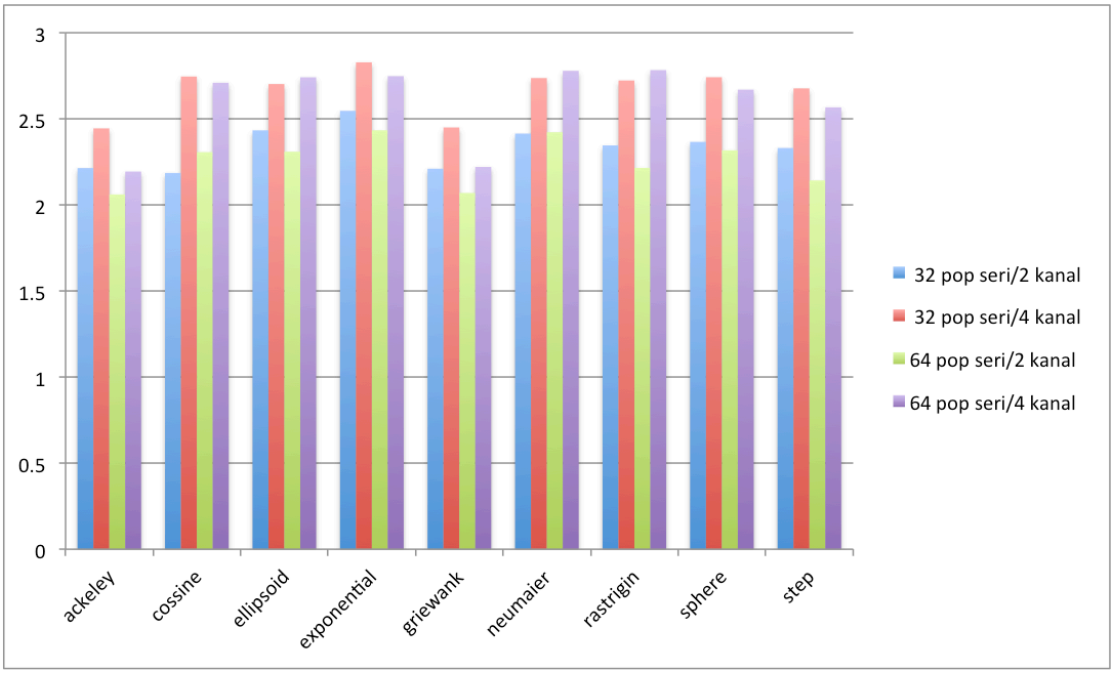
Test sistemlerinden, bütün kriter fonksiyonlarında elde edilen hız artışları grafiği şekil 5.63-5.66' de verilmiştir.



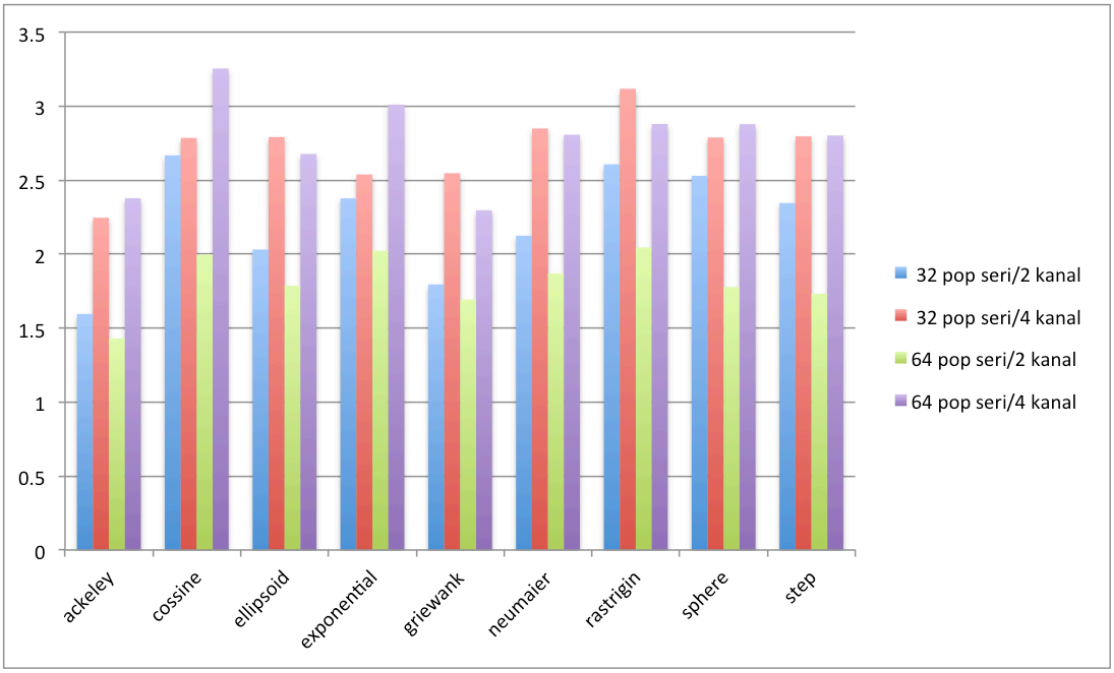
Şekil 5.63: Intel® Core™ i7-3720QM sisteminin hız artış grafiği



Şekil 5.64: Intel (R) Core(TM) i5 CPU M 460 sisteminin hız artış grafiği



Şekil 5.65: Intel(R) Core (TM) i3 CPU M 370 sisteminin hız artış grafiği



Şekil 5.66: Intel(R) Core (TM)2 Quad CPU Q6600 sisteminin hız artış grafiği

Yukarıdaki şekillerden i7, i5 ve i3 işlemcili sistemlerde görülebileceği gibi, bu sistemler HT teknolojisi barındırdığından dolayı, bu sistemlerin gerçek çekirdek

sayısına kadar olan kanal sayılarındaki parelelleştirilmesinde (i7 için 8 kanal, i3 ve i5 için 4 kanal paralel programlama) hız artışı elde edilirken, HT teknolojisinin dahil edildiğinde, işletim sistemi tarafından görünen kadar kanal sayısı ile paralelleştirme durumunda, hız düşüşü olmaktadır. Bunun sebebi Bölüm 5.3' te anlatılan, veri dağıtım ve veri toplama sürelerinin, paralelleştirmeden elde edilen kazançtan fazla olmasından kaynaklanmaktadır. Ancak Q6600 işlemcisi, HT teknolojisi barındırmaz ve 4 gerçek çekirdeği bulunmaktadır. Bu sebeple bu işlemcide bu problem ile karşılaşılmamıştır.

6. SONUÇ

Bu tezdeki çalışmanın amacı, günlük hayatta kullanılan bilgisayarlarda, paralel programlama ile elde edilebilecek performans artışının gösterilmesidir. Uygulama dört farklı maliyet aralığında seçilen test sistemlerinde test edilmiştir ve bunlarda ölçeklenebilir benzer karakteristikte sonuçlar elde edilmiştir.

Bilgisayarların tarihine kısaca bakılacak olunursa, ilk olarak tek işlemcili bilgisayarlar üretilmiştir. Bu bilgisayarlar zamanında masaüstü uygulamaları doğal olarak tek işlemciye göre yazılan, her döngüde tek bir işlem yapabilen bu işlemcilere uygun seri olarak yazılmış yazılımlardı. Ancak fazla işlem yükü barındıran işlemlerin çözülebilmesi için, bu sistemler küme halinde, farklı topolojiler kullanılarak birbirine bağlanması ile, paralel bilgisayar sistemleri kurulmuş ve işlem yükü fazla olan problemler bu sistemler üzerinde çözüme ulaştırılmaya çalışılmıştır. Daha sonraları, her birinde tek çekirdek barındıran ama anakart üzerinde birden fazla işlemci barındıran, ancak maliyetinden dolayı daha çok sunucu sistemlerde kullanılan hibrid diyebileceğimiz sistemler üretilmiştir. Bu sistemler ile paralel programlama biraz daha kullanıcı seviyesine inmiş olsa da, her evde olabilecek şekilde yaygınlaştığı söylenemez.

2000'li yılların başından itibaren, tek işlemci ile birlikte çok çekirdek barındıran sistemler üretilmeye başlanmıştır ve günümüzde kullanılan bilgisayarların hemen hemen tamamı bu mimarideki bilgisayarlar haline gelmiştir. Doğal olarak paralel programlamanın her bireyin bilgisayarında çalıştırılabileceği bir zaman gelmiştir denilebilir.

Donanım alanındaki bu gelişmelere rağmen, yazılım alanında paralel gelişme gözlenmiştir demek yanlış olur. Bilimsel araştırmalar ve bazı oyun firmaları haricinde, ki bu oyun firmaları GPU paralelleştirme teknolojisini kullanmaktadır, paralel çalışabilen uygulamalar olması gereken seviyede değildir.

Bu tezde yapılan çalışma ile, çok düşük maliyetli bir bilgisayarda bile paralel programlama ile elde edilebilecek hız artışının gösterilmesi hedeflenmiştir.

Uygulama sonucunda görülmüştür ki, en düşük test sisteminde ve en karmaşık kriter fonksiyonunda dahi en az 2 kat hız artışı elde edilmiştir. Buda paralel programlamanın en basit sistemlerde bile uygulanabileceğini ve hatta uygulanması gerektiğini göstermektedir. Bu hız artışı paralel programlama literatüründe, algoritma ve kriter fonksiyonlarının karmaşıklığı düşünüldüğünde kabul edilebilir bir sonuçtur. Örneğin Panagiotis D. Michailidis ve Konstantinos G. Margaritis (Michailidis, Margaritis, 2011) 2011 yılında yaptıkları çalışmada kütüphane olarak OpenMP (OpenMP, 2012) kullanmışlardır ve 2 kanal paralel çalışma sonucunda 1.3 ile 2.1 kat hız artışı, 4 kanal çalışma sonucunda da 1.2 ile 3.6 kat hız artışı elde etmişlerdir. Başka bir çok çekirdekli mimarilerde yapılmış çalışma olarak, Virginie Maris ve Philip E. Wannamaker (Maris ve Wannamaker, 2010) 2010 yılında yaptıkları çalışmada 2 çekirdek için yaklaşık olarak 2 kat, 4 çekirdek için yaklaşık 3.6 kat hız artışı elde etmişlerdir. Bir başka çalışma olarak 2012 yılında yapılmış olan E. Yamada, T. Shimada ve A.K. Hayashi'nin (Yamada vd. 2012) Java kullanarak yaptıkları çalışmada, 2 kanal için yaklaşık 2 kat, 4 kanal için yaklaşık 3.5 kat hız artışı elde etmişlerdir.

Bu tezde yapılmış olan çalışmada da literatürdeki sonuçlara benzer sonuçlar elde edilmiştir. 2 kanal çalışmada en düşük 1.6 kat, en yüksek 2.6 kat hız artışı, 4 kanal çalışmada ise en düşük 2.1, en yüksek 3.6 kat hız artışı elde edilmiştir. Tezde yapılan çalışmanın literatürdeki çalışmalardan farkı, büyük parçacık paralelleştirme yapılmıştır. Bu tekniğin seçilmesinin sebebi, her çekirdeğin bir problemi baştan sona kendi yapması sonucu elde edilebilecek hız artışını gözlemlemek, bunun sonucu olarak da, geliştirilebilecek uygulamalara veriyi bölmeden, sadece işlemleri dağıtarak elde edilebilecek hız artışını göstermektir.

7. KAYNAKLAR

3720QM: http://ark.intel.com/products/64891 , 12.12.2012.

Abed, Majida Ali; Ismail, Ahmad Nasser; Hazi, Zubadi Matiz 2010, *Pattern Recognition Using Genetic Algorithm*. International Journal of Computer and Electrical Engineering, Col. 2, No. 3.

Ayala, Helon Vicente Hultmann; Coelho, Leandro dos Santos 2012, *Tuning of PID controller based on a multiobjective genetic algorithm applied to a robotic manipulator*. Expert Systems with Applications 39, 2012.

Baker, J. E. 1985, "*Adaptive selection methods for genetic algorithms*", Proceedings of an International Conference on Genetic Algorithms and their applications, pp 101-111.

BGL,2012: http://www.boost.org/doc/libs/1_52_0/libs/graph/doc/ , 2012.

Boost, 2012: http://www.boost.org/doc/libs/1_50_0/libs/graph_parallel/doc/html/index.html , 2012.

Chandra, R.; Dagum, L.; Kohr, D.; Maydan, D; McDonald, D; McDonald, J; Menon, R 2001, *Parallel Programming in OpenMP*, Academic Press, ISBN 1-55860-671-8.

Chipperfield, Andrew; Fleming, Peter 1996, *Multiobjective Gas Turbine Engine Controller Desing Using Genetic Algorithms.* IEEE Transactions on Industrial Electronics, Vol. 43, No. 5.

CUDA,2012. http://www.nvidia.com/object/cuda_home_new.html , 08.12.2012.

Duato, Jose; Yalamanchili, Sudhakar; Ni, Lionel 2003, *Interconnection Networks, An Engineering Approach*. Morgan Kaufmann Publishers An Imprint of Elsevier Science. ISBN 1-55860-852-4.

Erben, Wilhelm; Burke, E. 2001, *A Grouping Genetic Algorithm for Graph Colouring and Exam Timetabling*. PATAT'00 Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III.

Falahiazar, Leila; Teshnehlab, Mohammed; Falahiazar. Alireza 2012, "Parallel Genetic Algorithm based on a new migration strategy," *Recent Advances in Computing and Software Systems (RACSS), 2012 International Conference on* , vol., no., pp.37-41, 25-27 April 2012 doi: 10.1109/RACSS.2012.6212694.

Gebali, Fayez 2011, *Algorithms And Parallel Computing*, A John Wiley & Sons, Inc., Publication ISBN 978-0-470-90210-3,Hardcover.

Gen, Mitsio; Cheng, Runwei 1997, *Genetic Algorithms and Engineering Desing.* John Wiley & Sons, Inc. ISBN 0-471-12741-8.

Glover, Fred; Kochenberger, Gary A. 2003, *Handbook of Metaheuristics,* Kluwer Academic Publishers, ISBN 1-40207263-5.

Goldberg, David E. 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1 edition.

Goldberg, David E.; Deb, Kalyanmoy 1989, *A comparative analysis of selection schemes used in genetic algorithms.* Foundation of Genetic Algorithms, edited by Gregory J.E. Rawlins, ISBN 1-55860-170-8.

Guinness,2011: *http://www.amd.com/us/press-releases/Pages/amd-showcases-worlds-2011sept13.aspx* , 08.12.2012.

Haupt, Randy L.; Haupt, Sue Ellen 2004, *Practical Genetic Algorithms*, Randy L. Haupt, Sue Ellen Haupt. ISBN: 0-471-45565-2. 2nd Ed.

Intel Atom 230: *http://ark.intel.com/products/35635/Intel-Atom-Processor-230-512K-Cache-1_60-GHz-533-MHz-FSB* , 09.12.2012.

Hockney, R.W.; Jesshope, C.R. 1988, Parallel Computers 2 Architecture, Programming and Algorithms. IOP Publishing Ltd. ISBN 0-85274-811-6, 2nd Ed.

Holland, John H. 1992, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan; re-issued by MIT Press.

Karaboğa, Derviş 2011, Yapay Zeka Optimizasyon Algoritmaları. Nobel Yayın Dağıtım Tic. Ltd. Şti. ISBN 978-605-395-434-7.

Karcı, Ali; Arslan, Ahmet, 2002 , 'Uniform Population in Genetic Algorithms' , Journal of Electrical and Electronics , Vol:2, pp:495-504.

Kumar, Rakesh 2012, *Blending Roulette Wheel Selection & Rank Selection in Genetic Algorithms*, International Journal of Machine Learning and Computing, Vol. 2, No. 4.

Kumar, Vipin; Grama, Ananth; Gupta, Ananth; Karypis, George 1994, Introduction to Parallel Computing. The Benjamin/Cummings Publishing Company, Inc. ISBN 0-8053-3170-0.

Langton, Christopher G. 1988, *Artificial Life An Overview*, edited by. A Bradford Book The MIT Press. ISBN 0-262-12189, 4. Printing.

Leung, Yiu-Wing; Wang, Yuping 2001 , "An orthogonal genetic algorithm with quantization for global numerical optimization," *Evolutionary Computation, IEEE Transactions on* , vol.5, no.1, pp.41-53, Feb 2001 doi: 10.1109/4235.910464

Li, Jianming; Lv, Ximeng; Liu, Linlin 2011, "A Parallel Genetic Algorithm with GPU Accelerated for Large-scale MDVRP in Emergency Logistics," *Computational Science and Engineering (CSE), 2011 IEEE 14th International Conference on* , vol., no., pp.602-605, 24-26 Aug. 2011 doi: 10.1109/CSE.2011.106.

M370: *http://ark.intel.com/products/49020/Intel-Core-i3-370M-Processor-3M-cache-2_40-GHz* , 12.12.2012

M460: *http://ark.intel.com/products/50179/Intel-Core-i5-460M-Processor-3M-Cache-2_53-GHz* , 12.12.2012.

Maris, Virginie; Wannamaker, Philip E. 2010, Parallelizing a 3D finite difference MT inversion algorithm on a multicore PC using OpenMP, Computers & Geosciences, Volume 36, Issue 10, October 2010, Pages 1384-1387, ISSN 0098-3004, 10.1016/j.cageo.2010.03.011.

Melanie, Mitchell 1999, *An Introduction to Genetic Algorithms*. A Bradford Book The MIT Press. ISBN 0-262-13316-4 5. Printing.

Michailidis, Panagiotis D.; Margaritis, Konstantinos G. 2011, Parallel direct methods for solving the system of linear equations with pipelining on a multicore using OpenMP, Journal of Computational and Applied Mathematics, Volume 236, Issue 3, 1 September 2011, Pages 326-341, ISSN 0377-0427, 10.1016/j.cam.2011.07.023.

Michalewicz, Z. 1992, *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlang, 252.

MPI,2012: *http://www.mcs.anl.gov/research/projects/mpi/* , 10.12.2012.

Pthreads, 2012: *https://computing.llnl.gov/tutorials/pthreads/* , 2012.

Q6600: *http://ark.intel.com/products/29765/Intel-Core2-Quad-Processor-Q6600-8M-Cache-2_40-GHz-1066-MHz-FSB* , 12.12.2012.

Qiancheng, Zhao; Qianjin, Tan; Zengyu, Cai 2011, "Application and research of logistics vehicles dispatching system based on parallel genetic algorithm," *Transportation, Mechanical, and Electrical Engineering (TMEE), 2011 International Conference on* , vol., no., pp.833-836, 16-18 Dec. 2011 doi: 10.1109/TMEE.2011.6199331.

Q6600: *http://ark.intel.com/products/29765/Intel-Core2-Quad-Processor-Q6600-8M-Cache-2_40-GHz-1066-MHz-FSB* , 12.12.2012.

Quinn, Michael J. 2003; *Paralel Programming in C with MPI and OpenMP*. Oregon State University, ISBN 007-282256-2, 1st Ed.

OpenMP,2012: *http://openmp.org/wp/* , 09.12.2012.

Rechenberg, Ingo 1973, *Evolutionsstrategie – Optimierung technisher Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, GER.

Salomon, Ralf 1996, Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. A survey of some theoretical and practical aspects of genetic algorithms, Biosystems, Volume 39, Issue 3, 1996, Pages 263-278, ISSN 0303-2647, 10.1016/0303-2647(96)01621-8.

Schewefel, H.P. 1984, *Evolution Strategies: A Family of Non-Linear Optimization Techniques Based on Imitating Some Principles of Organic Evolution*. Annals of Operations Research 1(1984)165-167.

Tang, Ke; Li, Xiaodong; Suganthan, P.N.; Yang, Zhenyu; Weise, Thomas 2009, Benchmark Functions for the CEC'2010 Special Session and Competition on Large-Scale Global Optimization.

TBB,2012: *http://threadingbuildingblocks.org/* , 08.12.2012.

Tsuji, Miwako; Sato, Mitsuhisa; Tanabe, Akifumi S.; Inagaki, Yuji; Hashimoto, Tetsuo 2012, "An asynchronous parallel genetic algorithm for the maximum likelihood phylogenetic tree search," *Evolutionary Computation (CEC), 2012 IEEE Congress on* , vol., no., pp.1-8, 10-15 June 2012 doi: 10.1109/CEC.2012.6256560.

Valdez, F.; Melin, P.; Parra, H. 2011, "Parallel genetic algorithms for optimization of Modular Neural Networks in pattern recognition," *Neural Networks (IJCNN), The 2011 International Joint Conference on* , vol., no., pp.314-319, July 31 2011-Aug. 5 2011 doi: 10.1109/IJCNN.2011.6033237

Vose, M. 1999, *The Simple Genetic Algorithm: Foundation and Theory*. MIT Press, 251.

Whitley, L.Darrell; Vose, Michael D. 1995, Foundations of Genetic Algorithms-3, Edited by. Morgan Kaufmann Publishers, Inc. ISBN 1-55860-356-5.

Wahib, Mohamed; Munawar, Asim; Munetomo, Masaharu; Akama, Kiyoshi 2011, Optimization of Parallel Genetic Algorithms for nVidia GPUs. *Evolutionary Computation (CEC), 2011 IEEE Congress on* , vol., no., pp.803-811, 5-8 June 2011 doi: 10.1109/CEC.2011.5949701

Wolpert, David H.; Macready, William G. 1997, "No free lunch theorems for optimization," *Evolutionary Computation, IEEE Transactions on* , vol.1, no.1, pp.67-82, Apr 1997 doi: 10.1109/4235.585893

Yamada, E.; Shimada, T.; Hayashi, A.K. 2012, *Development of Java multi-threaded simulation for chemical reacting flow of ethanol, Advances in Engineering Software, Volume 54, December 2012, Pages 17-23, ISSN 0965-9978, 10.1016/j.advengsoft.2012.08.006.*

Yang, Xin-She 2010, *Nature-Inspired Metaheuristic Algorithms*, Luniver Press, ISDB 1-905986-28-9, 2nd. Ed.

Zhu-rong, Wang; Tao, Ju; Du-wu, Cui; Xin-hong, Hei 2011, "A study of hybrid parallel genetic algorithm model," *Natural Computation (ICNC), 2011 Seventh International Conference on* , vol.2, no., pp.1038-1042, 26-28 July 2011 doi: 10.1109/ICNC.2011.6022186

Zomaya, Albert Y. 1996, Parallel Computing: Paradigms and Applications, Edited By. International Thomson Computer Press, ISBN 1-85032-188-4.

ÖZGEÇMİŞ

Doğum Tarihi : 16.06.1982

Doğum Yeri : Malatya

Lisans : İzmir Yüksek Teknoloji Enstitüsü Bilgisayar Mühendisliği Bölümü

Yüksek Lisans : Şubat 2011 – Ocak 2013, İnönü Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı

Kurum : 2010 – devam ediyor, İnönü Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü