T.C
DÜZCE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ELEKTRİK EĞİTİMİ ANABİLİM DALI

PARALEL GENETİK ALGORİTMA İLE SAYISAL FİLTRE OPTİMİZASYONUNUN KARŞILAŞTIRMALI ANALİZİ

YÜKSEK LİSANS TEZİ

HÜSREV YILDIZ

AĞUSTOS 2013

DÜZCE

KABUL VE ONAY BELGESİ

Hüsrev YILDIZ tarafından hazırlanan PARALEL GENETİK ALGORİTMA İLE SAYISAL FİLTRE OPTİMİZASYONUNUN KARŞILAŞTIRMALI ANALİZİ isimli lisansüstü tez çalışması, Düzce Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 27.08.2013 tarih ve 2013-359 sayılı kararı ile oluşturulan jüri tarafından Elektrik Eğitimi Anabilim Dalı'nda Yüksek Lisans Tezi olarak kabul edilmiştir.

Üye
(Tez Danışmanı)
Yrd. Doç. Dr. Devrim AKGÜN
Düzce Üniversitesi

Üye
Doç. Dr. İbrahim YÜCEDAĞ
Düzce Üniversitesi

Üye
Doç. Dr. Resul KARA
Düzce Üniversitesi

Tezin Savunulduğu Tarih : 28.08.2013

ONAY

Bu tez ile Düzce Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu Hüsrev YILDIZ'ın Elektrik Eğitimi Anabilim Dalı'nda Yüksek Lisans derecesini almasını onamıştır.

Prof. Dr. Haldun MÜDERRİSOĞLU
Fen Bilimleri Enstitüsü Müdürü

BEYAN

Bu tez çalışmasının kendi çalışmam olduğunu, tezin planlanmasından yazımına kadar bütün aşamalarda etik dışı davranışımın olmadığını, bu tezdeki bütün bilgileri akademik ve etik kurallar içinde elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara kaynak gösterdiğimi ve bu kaynakları da kaynaklar listesine aldığımı, yine bu tezin çalışılması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını beyan ederim.

06Agustos 2013

Hüsrev YILDIZ

TEŞEKKÜR

Yüksek lisans ve bu tezin hazırlanması süresince desteğini esirgemeyen Ali KARASU'ya ve gösterdiği her türlü yardımdan dolayı çok değerli hocam Yrd. Doç. Dr. Devrim AKGÜN'e en içten dileklerimle teşekkür ederim.

06 Ağustos 2013 **Hüsrev Yıldız**

İÇİNDEKİLER **Sayfa**

ŞEKİL LİSTESİ

Sayfa No

ÇİZELGE LİSTESİ

SİMGELER VE KISALTMALAR

GA	Genetik algoritma
PGA	Paralel genetik algoritma
FIR	Sonlu dürtü yanıtı
IIR	Sonsuz dürtü yanıtı
x(n)	Giriş sinyali
y(n)	Çıkış sinyali
f(x)	Uygunluk fonksliyonu

ÖZET

PARALEL GENETİK ALGORİTMA İLE SAYISAL FİLTRE OPTİMİZASYONUNUN KARŞILAŞTIRMALI ANALİZİ

Hüsrev YILDIZ
Düzce Üniversitesi
Fen Bilimleri Enstitüsü, Elektrik Eğitimi Anabilim Dalı
Yüksek Lisans Tezi
Danışman: Yrd. Doç. Dr. Devrim AKGÜN
Ağustos 2013, 55 sayfa

Sayısal filtreler, sayısal işaretlerin bazı özelliklerini bir dizi çarpma ve toplama işlemine tabi tutularak istenilen şekilde değiştirmek için kullanılır. Filtre optimizasyonu, filtre karakteristiklerini sağlayacak en uygun filtre katsayılarının belirlenmesi için gerçekleştirilir. Geleneksel hesaplama teknikleri ile gerçekleştirilen filtre tasarımında katsayıların belirlenmesi için yapılan arama yerel minimum noktalarına takılıp kalabilmektedir. Genetik algoritma da ise arama işlemine farklı noktalardan devam edilerek genel minimum bulunabilir ve böylece en uygun değerlere ulaşılabilir. Genetik algoritmalarda çözülecek problemin hesaplama yükü arttığı zaman, algoritmanın hızlandırılması için en etkin seçeneklerden biri olan paralel formda gerçekleştirme işlemine başvurulmaktadır. Bu tez çalışmasında sayısal filtre optimizasyonunun paralel genetik algoritmalar kullanılarak, çok-çekirdekli bilgisayar üzerinde başarım analizi gerçekleştirilmiştir. Bu amaçla, paralel genetik algoritmalar C# programlama dili ile kodlanmış ve paralel hesaplamalar için yerleşik Parallel kütüphanesi kullanılmıştır. Farklı filtre yapılarının optimizasyonu için elde edilen başarım değerleri deneysel olarak incelenmiş ve sıralı algoritmada gereken çalışma süresinin, paralel algoritma ile azaltılması sağlanmıştır. Geliştirilen arayüz kullanılarak dört ve altı çekirdekli işlemcilerle yapılan deneysel ölçümler sıralı hesaplama ile karşılaştırıldığında başarımın kullanılan çekirdek adedine bağlı olarak arttığı gözlenmiştir.

Anahtar sözcükler: Sayısal filtre, Genetik Algoritma, Paralel programlama, C#

ABSTRACT

COMPARATIVE ANALYSIS OF DIGITAL FILTER OPTIMIZATION USING PARALLEL GENETIC ALGORITHM

Hüsrev YILDIZ
Duzce University
Graduate School of Natural and Applied Sciences, Departmant of Electrical Education
Master of Science Thesis
Supervisor: Assist. Prof. Dr. Devrim AKGÜN
August 2013, 55 pages

Digital filters are used to modify some characteristics of digital signals by means of a series of multiplication and addition operations. Optimization of the filter is realized to determine the optimal filter coefficients that provide the desired characteristics. Filter design with traditional searching techniques to achieve coefficients may remain trapped in a local minima point. In Genetic algorithm, global minimum can be found via continuing search operation from different points and thus optimal values can be determined. In Genetic algorithms, when the size of the problem to be solved increases, parallel realization which is one of the most effective ways applied for the acceleration of the algorithm. In this study, performance analysis of digital filters optimization using parallel genetic algorithms was carried out on multi-core computer. For this purpose, parallel genetic algorithm was coded with C# programming language and built in Parallel library was used for parallel computations. Performance for different filter structures was examined experimentally and the duration of the sequential optimization algorithm is reduced by parallel algorithm. When the experimental measurements with four and six cores processors using developed interface are compared with sequential implementation it is observed that performance is increased depending on the number of cores.

Keywords: Digital filter, Genetic algorithm, Parallel programming, C#

EXTENDED ABSTRACT

COMPARATIVE ANALYSIS OF DIGITAL FILTER OPTIMIZATION USING PARALLEL GENETIC ALGORITHM

Hüsrev YILDIZ
Duzce University
Graduate School of Natural and Applied Sciences, Departmant of Electrical Education
Master of Science Thesis
Supervisor: Assist. Prof. Dr. Devrim AKGÜN
August 2013, 55 pages

1. INTRODUCTION:

Filters are implemented in software by defining as functions that contain a series of arithmetic operations. The coefficients in these functions constitute the basic structure of the filters. In filter optimization, the process of determining the most relevant values of filter coefficients is carried out. A genetic algorithm (GA) is a powerful optimization tool that mimics the process of natural evolution by using heuristic calculation method. This heuristic method (also sometimes called a metaheuristic) is routinely used to generate useful solutions to optimization and search problems. GA, as used in many areas, can be utilized in the calculation of filter coefficients and promising results can be achieved, as well. In this thesis, the digital filter optimization has been aimed to realize faster by using parallel genetic algorithms on a multi-core PC and has been carried out by means of C#.Net programming language in a PC environment.

2. MATERIAL AND METHODS:

Signal processing, if defined in general, is the process in which a signal sequence is created in the desired form after various operations are done. This mentioned structure of digital signal processing is called a *digital filter*. In this thesis, the performance analysis of calculating digital filter coefficients is performed by using the IIR filter model in the carried out experimental studies. In the GA structure, the fitness of each chromosome is evaluated by using the performance criteria of Mean of the Squared Errors (abbreviated as MSE). In the used structure of GA, the real value encoding

method is used and, crossover and mutation operations are performed. Elitism operator is also used in the optimization stage. As for the parallel GA models, the method of master-slave model is applied to the problem and in the parallelization process, the parallelization is done by dividing the fitness calculation part, which is the most time-consuming part in the sequential GA, to the threads in the number of cores in the processor. Object-oriented programming has become quite widespread nowadays and in general, studies have been started to be designed by any of object-oriented programming language. In this thesis, sub-classes related to the parallelization are benefited with C#.Net visual programming language and .Net framework of 4.5 version using Microsoft Visual Studio 2012 compiler.

3. RESULTS AND DISCUSSIONS:

In the experimental studies, an interface that provides a quick and easy adjustment for the GA parameters has been developed. The interface makes it also easy to interpret the parallelization analysis results. Moreover, by coding various filter models, it is provided to monitor the parallelization performance regarding the degree of difficulty of the problem. From the results obtained by the experimental studies, it is observed that the aimed parallelization process is dependent on the number of cores and the GA running in the parallel configuration has been speeded up considerably compared to the sequential GA depending on the number of cores. Approximately 4.3 fold acceleration for six cores processor and approximately 3.5 fold acceleration four cores processor with hyperthreading support have been achieved.

4. CONCLUSION AND OUTLOOK:

In this thesis, the digital filter optimization has been provided to be performed faster by means of parallel genetic algorithms on a multi-core PC. Parallel performance analysis has been obtained by using the processors Intel® Core™ i7-4950HQ and AMD Phenom™ 1055T. From the experimental results, as a result of the parallelization of the GA based on the number of cores, successful results have been achieved. Additionally, it is observed that the feature of Hyper-Treading from the processor technologies provides significant benefits in the parallelization process. Furthermore, using different GA parallelization methods and different programming languages for the parallelism structures, speed analysis of the parallelization method and programming languages can be done in the parallelization process.

1. GİRİŞ

1.1. AMAÇ VE KAPSAM

Filtreler, yazılımda temel olarak çarpma ve toplama işlemlerinin yapılmasıyla gerçekleştirilmektedir. Filtrenin girişine uygulanan işaret bir dizi çarpma ve toplama işlemine tabi tutularak filtreleme işlemi gerçekleştirilir. İşaretin işleme tabi tutulduğu matematiksel fonksiyondaki katsayılar filtrenin temel yapısını oluşturmaktadır. Bu katsayıların değeri filtrenin istenilen özellikleri gerçekleştirmesini oluşturmaktadır. Filtre optimizasyonu işleminde, filtrenin temel yapısını oluşturan bu katsayıların istenilen amaca en uygun değerlerde belirlenmesi işlemi gerçekleştirilmektedir[1,2]. Sezgisel yaklaşım yöntemleri kullanılarak yapılan programlama yöntemlerine, genel olarak Evrimsel Hesaplama (evalutionary computing) yöntemleri denir. Yapay zeka alanında kullanılan genetik algoritmalar doğal evrim sürecini taklit eden güçlü bir optimizasyon aracıdır [3,4,5]. Evrimsel Hesaplama yöntemlerinden biri olan Genetik algoritmalar, birçok alanda uygulanma imkânı bulmaktadır. Bu uygulama alanları içerisinde, genellikle bilinen hesaplama yöntemlerinin yetersiz olduğu veya bu yöntemlerin uygulanmalarının uzun vakit aldığı alanlar fazlaca görülmektedir. Bu yöntemlerden biriside filtre katsayılarının hesaplanmasıdır[6]. Geleneksel hesaplama teknikleri ile yapılan filtre tasarımında arama genelde yerel minimum noktalarına takılıp kalmaktadır. Genetik algoritmalar da ise bu sorun algoritma gerçekleştirilirken mutasyon işlemi ile aşılmakta ve çözüme farklı noktalardan devam edilebilmekte en uygun değerlere ulaşılabilmektedir[7,8,9].

Genetik algoritmalar genelde makul zamanda iyi sonuçlar üretebilir fakat çözülecek problemin büyüklüğü arttığı zaman çözümleme zamanı oldukça artmaktadır. Bundan dolayı genetik algoritmaların hızlandırılması için en etkin seçeneklerden biri olan paralel formda gerçekleştirme işlemine başvurulmaktadır [10,11,12,13]. Paralel genetik algoritmalar birçok mühendislik alanında yaygın olarak kullanılmaktadır. Örneğin genetik ve gen tarama [14], inşaat projeleri [15], Küme (Cluster) OpenMP'nin Fortran ile gerçekleştirilmesi için Intel tarafından simetrik çok işlemcili uygulama programlama

arayüzü (API) geliştirilmiştir[16]. Daha sonra bu çalışma ile elde edilen sonuçlardan ters problemlerin (inverse problems) çözümünde başarılı seviyede hızlanma sağlamıştır[17,19]. Yer altı suları çalışmasında kullanılan paralel genetik algoritmaların gerçekleştirilmesinde paralelleştirmenin çalışma anında (runtime parallelization) gerçekleştiği Mesaj İletim Arayüzü (Message Passing Interface -MPI) iletişim protokolü kullanılmıştır[18].

Ülkemizde de paralel optimizasyon yöntemi yaygın bir şekilde kullanılmaktadır. Genetik algoritmada iyi bireyleri iyi alt popülasyonlara toplayarak düşük göç maliyeti ile başarılı sonuçlar elde etmeyi sağlayan Elit Göç Metodu (EGM) geliştirilmiştir [20]. 2006 yılında Turgay Kaya tarafından yapılan bir çalışmada GA kullanılarak sayısal filtre tasarımı gerçekleştirilmiş ve genetik algoritma ile filtre katsayılarının hesaplanması adı verilen yazılım geliştirilmiştir. Elde edilen sonuçlar MATLAB ta elde edilen katsayılarla karşılaştırılarak sonuçların aynı olduğu gözlemlenmiştir[2]. 2007 yılında İlker Ozan Koç tarafından yapılan bir doktora tezi çalışmasında, gezgin satıcı problemi için çok popülasyonlu paralel bir genetik algoritma geliştirilmiş ve tek popülasyonlu GA çalışmaları ile de analizi yapılmıştır[21]. 2010 yılında Selim Onur Uşkay tarafından Paralel Hibrit Genetik Algoritmalarla Katı Atık Taşınması İçin Rota Optimizasyonu isimli yapılan çalışmada, PHGA ile katı atık taşınması için atık toplama rotalarını optimize eden ve araç yükü ile yol eğiminin yakıt tüketimine etkisini de hesaba katan bir bilgisayar yazılımı geliştirilmiştir [22]. 2008 yılında Özgür Aksu tarafından yapılan çalışmada baskın gen seçimi operatörüne ve popülasyondaki değişimin sürekliliğini sağlayacak yeni genetik algoritma metodu geliştirilmiş ve bu algoritma analog devre tasarımına uygulanarak başarılı sonuçlar elde edilmiştir. Geliştirilen metot da hesaplama süresinin kısaltılması amacıyla donanımsal paralellik içeren GA yapısı oluşturulmuştur ve devre tasarımlarında başarıyla kullanılabileceği görülmüştür[23].1994 yılında Nurhan Karaboğa tarafından yapılan çalışmada GA kullanılarak değişik tip sayılar için FIR filtre katsayılarının kuantalanması ile ilgili benzetim çalışmaları yapılmıştır. Sonuçlar el ile yuvarlatılma yapıldığında elde edilen sonuçlarla karşılaştırılmış ve FIR filtre katsayılarının yuvarlatılmasında başarıyla kullanılabileceği sonucuna varılmıştır[24]. 2008 yılında Ayşe Kara tarafından yapılan çalışmada, şebekeden alınan güç işaretindeki gürültüler tespit edilerek filtrelenmiş, güç kalitesini arttıran bir sayısal filtre tasarımı gerçekleştirilmiş ve problemin çözümüne en uygun filtrenin yüksek geçiren filtre olduğu sonucuna varılmıştır[25].

2011 yılında da Zeynep Batık tarafından yapılan yüksek lisans tez çalışmasında, GA kullanılarak sayısal filtre tasarımı gerçekleştirilmiş ve GA operatörlerinin filtre tasarımına etkileri açıklanmıştır Ayrıca yapılan bu çalışma MATLAB ve ASP.NET programlarında çalışması sağlanarak eğitim materyali hazırlanmıştır[26].

Yapılan bu tez çalışması ile sayısal filtre optimizasyonunun çok-çekirdekli bilgisayar üzerinde paralel genetik algoritmalar kullanılarak daha hızlı gerçekleştirilmesi hedeflenmiştir. Bu çalışma, günümüzde yaygın olarak kullanılan çok çekirdekli bilgisayarlar üzerinde yaygın kullanıma sahip C# programlama dili ile gerçekleştirilmesi ile sağlanan başarımın analiz edilmesi ve bu konuda ki literatür boşluğunun doldurulması açısından önem taşımaktadır. Sayısal filtre parametrelerinin sıralı algoritma ile optimizasyonu sonucu geçen hesaplama zamanları referans alınarak, çekirdek adedine bağlı olarak elde edilen hesaplama zamanlarından belirlenen hızlanma sonuçlarının karşılaştırılması amaçlanmıştır.

2. MATERYAL VE YÖNTEM

2.1. SAYISAL FİLTRELER

Fiziksel bir olay hakkında veri taşıyan ve o olay hakkında bize bilgiler veren, bir veya daha fazla sayıda değişken içeren fonksiyonlara işaret denir. İstenilen özelliklere sahip işaret üreten veya girişine uygulanan işarete göre çıkışında istenilen özelliklere sahip çıkış üreten yapılara ise sistem denir. İşaret işleme ise bilgisayar yada özel olarak üretilmiş olan sayısal işaret işleme donanımları sayesinde, sistemleri meydana getiren yapıdır. İşaret işleme genel olarak tanımlanacak olursa, bir işaret dizisinin çeşitli işlemler yapıldıktan sonra istenilen hale getirilmesi işlemidir[26].

Sayısal işaret işlemenin amacı, bir işaretin frekans spekturumu üzerinde belirli frekanslarda istenilen işlemleri yapması ve bu işlemlerden sonra istenilen özellikte sonuç alınmasını sağlamaktır. Yapılan bu sayısal işaret işleme yapısına sayısal filtre adı verilmektedir. Sayısal filtreler yazılım veya donanımla gerçekleştirilebilirler. Sayısal filtreler impals cevaplarına göre ikiye ayrılmaktadırlar. Bunlar FIR(sonlu impuls cevaplı) ve IIR(sonsuz impals cevaplı) filtrelerdir[27].

Rasyonel bir sistem fonksiyonu ile doğrusal ötelemeyle değişmez bir sistem için x(n) girişi ve y(n) çıkışı doğrusal sabit katsayılı fark denklemiyle ilişkilendirilir. Örneğin $h(n) = a^n\, u(n)$birim örnek cevabına sahip olan bir sistem Denklem 2.1'deki denklem ile tanımlanmaktadır.

$$y(n) = \sum_{k=0}^{\infty} a^k x(n-k) \tag{2.1}$$

Bu denklem herhangi bir $x(n)$ girişi için $y(n)$ çıkışının hesaplanmasına izin vermesine rağmen, bu gösterim hesaplama bakış açısından çok etkin değildir. Bazı durumlarda girişin mevcut ve geçmiş değerlerine ek olarak çıkışında geçmiş değerler cinsinden ifade edilmesi mümkün olabilir. Örneğin önceki sistem Denklem 2.2'deki gibi kısaca tanımlanabilir.

$$y(n) = ay(n-1) + x(n) \tag{2.2}$$

Denklem 2.2'de verilen denklem doğrusal sabit katsayılı fark denklemi olarak bilinir ve genel biçimi Denklem 2.3'teki gibidir.

$$y[n] = \frac{1}{a_0}(b_0 x[n] + b_1 x[n-1] + \cdots + b_p x[n-P] \\ -a_1 y[n-1] - a_2 y[n-2] - \cdots - a_Q y[n-Q] \tag{2.3}$$

- P : İleri besleme filtre
- b_i: İleri besleme filtre katsayıları
- Q: Geri besleme filtre
- a_i: Geri besleme filtre katsayıları
- $x[n]$: Giriş sinyali
- $y[n]$: Çıkış sinyali

Bu denklem toplam sembolleri kullanılarak Denklem 2.4'deki gibi ifade edilir.

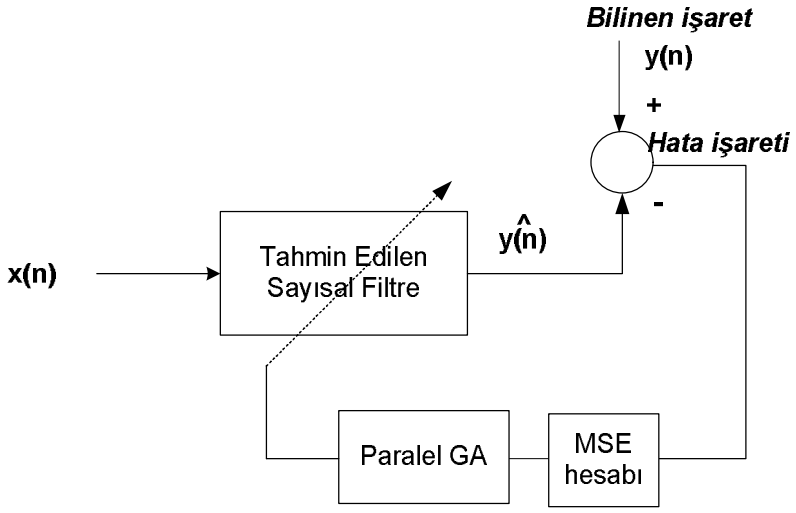
$$y(n) = \sum_{k=0}^{\infty} b(k)x(n-k) \; - \sum_{k=0}^{\infty} a(k)y(n-k) \tag{2.4}$$

Burada $a(k) ve\ b(k)$ katsayıları sistemi tanımlayan sabitlerdir. Eğer fark denklemi bir veya daha fazla sıfır olmayan $a(k)$ terimine sahip ise, fark denkleminin yinelemeli olduğu söylenir. Diğer yandan tüm $a(k)$'lar sıfır ise fark denkleminin yinelemeli olmadığı söylenir. Böylece denklem 2.2 birinci derece yinelemeli fark denkleminin bir örneğiyken, Denklem 2.1 sonsuz dereceden yinelemeli olmayan bir fark denklemidir [28].

2.2. SAYISAL FİLTRE KATSAYILARININ BELİRLENMESİ

Sayısal filtrelerin katsayıları genelde istenilen frekans cevabı özelliklerine göre belirlenir. Filtre katsayıları alçak geçiren, yüksek geçiren veya bant geçiren filtre gibi filtre yapılarına göre farklı değerler alır [26,28].

Uygulamada çoğu zaman filtre katsayıları gürültülü ve orijinal işaretlerin yapısına göre belirlenmesi ihtiyacı ortaya çıkmaktadır. Bu amaçla referans işaretler kullanılarak çeşitli optimizasyon yöntemleriyle katsayılar belirlenmektedir.



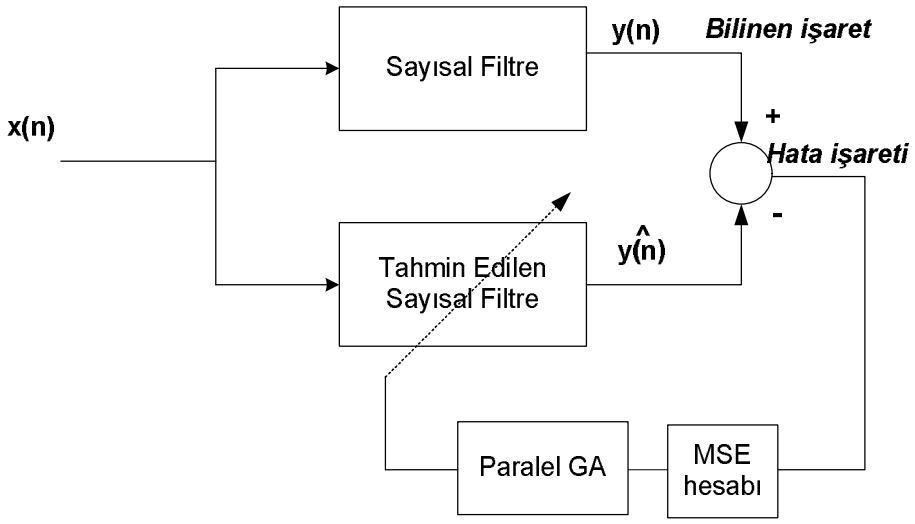
Şekil 2.1. Filtre katsayılarının genetik algoritmalar ile tahmini.

x(n) bilgi sinyali katsayıları tahmin edilen sayısal filtre girişine uygulanır. Elde edilen sonuçlar daha önceden elde edilmiş olan bilgi sinyali ile karşılaştırılarak hata sinyali elde edilir. Hata sinyalinden elde edilen sonuçların MSE değeri Denklem 2.5'deki şekilde hesaplanır.

$$MSE = \left(\sum_{i=1}^{n} (y_i - \hat{y})^2 \right) \frac{1}{n} \tag{2.5}$$

Elde edilen MSE sonuçları Paralel GA ya uygulanır ve MSE hata değerine göre yeni katsayılar elde edilerek, filtre katsayılarının optimizasyon işlemi gerçekleştirilir.

Gerçekleştirilen tez çalışmasında bilinen işaret katsayıları bilinen filtre kullanılarak Şekil 2.2'deki gibi elde edilmiştir. Böylece optimizasyon sonucunda elde edilecek katsayı değerleri açıkça belirtilmiştir.



Şekil 2.2. Filtre katsayılarının bilinen işaret üzerinden GA'lar ile tahmini.

2.3. GENETİK ALGORİTMALAR

Genetik Algoritma, rastlantısal arama yöntemleriyle çözüm bulmaya çalışan, parametre kodlama esasına dayanan bir arama tekniğidir. Genetik algoritmalar çözümü bulmak için seçme, çaprazlama ve mutasyon operatörlerini kullanan bir optimizasyon yöntemidir. Bu yöntemde problemin çözümüne ait rastgele birçok çözüm üretilir. Üretilen bu çözümlerden iyi olanlar bir sonraki nesle (generation) aktarılırken kötü olanlar elenir. Nasıl ki doğada bulunan bazı bireyler zamanla yok olurken diğer bireyler yaşamlarını sürdürür ve ayrıca yeni bir nesil üretir ise, genetik algoritmada da aynı yaklaşım vardır. Bir nesildeki iyi bireyler bir sonraki nesle aktarılır ve yeni bireyler üretilir. Bu sayede kötü bireyler elenmiş olur. Bir çözümün iyimi kötümü olduğuna karar verilmesi işlemini sağlayan kritere uygunluk ya da amaç fonksiyonu denir. Ayrıca her problemin kendine özgü bir uygunluk fonksiyonu vardır. Elde edilen nesillerde her çözüme ait uygunluk değeri hesaplanır, en iyi bireyler seçilir ve kötü olan bireyler çözüm kümesinden silinir. Silinen bireylerin yerine ise yeni bireyler üretilir. GA da elde edilen çözümlerin her birine birey veya kromozom adı verilir. Uygunluk değerine dayanarak bir sonraki nesle aktarılacak olan kromozomların belirlenmesi işlemine seçme işlemi denir. GA da yeni nesillerin oluşturulması ve nesiller arası farklılıkların sağlanması içinde çaprazlama ve mutasyon operatörleri kullanılır[29].

2.3.1 Genetik Algoritmaların Çalışması ve Adımları

Genetik algoritmalar da ilk üretilen çözüm kümesine başlangıç popülasyonu denir.

Genetik algoritmaların ilk adımı, başlangıç popülasyonunun üretilmesi ve uygunluk değerlerinin hesaplanmasıdır. Daha sonra diğer genetik operatörler uygulanır. Elde edilen her nesil için uygunluk değeri hesaplanır. Bu durum sonlandırma kriteri sağlanana kadar devam eder. Şekil 2.3'te genetik algoritmanın akış diyagramı verilmiştir. Bu akış şemasının elemanlarına ait açıklamalar aşağıdaki gibi maddeler halinde sıralanabilir.

Akış şemasının adım adım açıklanması:

1. [Başlangıç]: n kromozomdan oluşan rasgele bir popülasyon oluşturulur (problemin olası çözümleri)

2. [Uygunluk]: f(x) fonksiyonunda popülasyonun her bir kromozomu değerlendirilir.

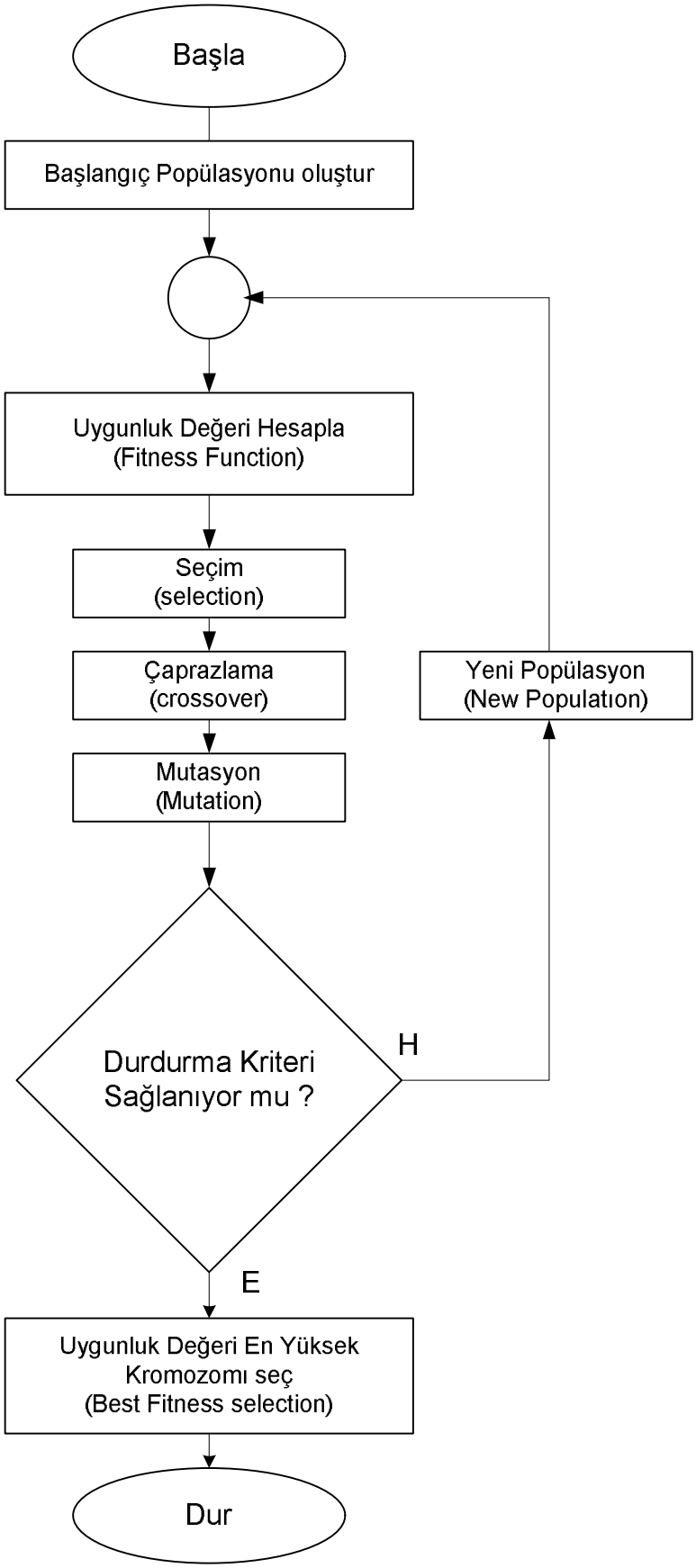
3.[Seçilim(Seleksiyon)]: Popülasyondan iki ebeveyn kromozom seçilir.(daha uygun olanın seçilme sansı daha fazladır)

4.[Çaprazlama]: Çaprazlama olasılığıyla yeni bireyler oluşturulur.

5.[Mutasyon]: Mutasyon olasılığı ile yeni nesli mutasyona uğratır.

6.[Durdurma Kriteri]: Eğer son durum yeterliyse en iyi kromozom seçilir veya popülasyona yeni bireyler eklenmesi için döngünün başına döner.

7.[Yeni Popülasyon]: 2. adıma geri dönülür.



Şekil 2.3. Genetik algoritma akış şeması.

2.3.2 Başlangıç Popülasyonu

Genetik algoritmada ki başlangıç popülasyonları genellikle rasgele üretilir. Fakat problemle ilgili başlangıçta bazı çözümler biliniyorsa bu durumda, popülasyon tamamen rasgele olarak değil de bir kısmı bilinen değerlerden oluşturulur ya da başlangıç popülasyonu belirli kısıtlar aralığında üretilir. Bu durum da programın en iyi çözümü bulmasında zaman yönünden tasarruf sağlanır [30].

Genetik algoritmanın performansını etkileyen kriterlerin başında başlangıç popülasyonu gelir. Popülasyon büyüklüğünün gereğinden fazla seçilmesi, işlemlerin karmaşıklığını ve aramanın derinliğini arttıracaktır. Bu durum da en iyi çözümü bulmada geçen zamanın artmasına neden olacaktır. Buna karşılık popülasyonun az sayıda seçilmesi ise kromozomların çeşitliliğinde azalmaya neden olacak ve çözümün istenilen hassasiyette olması sağlanamayacaktır. Başlangıç popülasyonun büyüklüğü yanı sıra GA programında uygulanacak olan jenerasyon sayısı da önemlidir. Jenerasyon sayısı ile popülasyon büyüklüğünün en iyi şekilde dengelenmesi gerekmektedir[31].

Genetik algoritmalarda başlangıç popülasyonu oluşturulurken kromozomlar rastgele değerler seçilerek oluşturulur. Ancak bu durum arama uzayında geniş bir alanı kaplama durumuna yol açmakta ve çözüme ulaşmada geçen süre artmaktadır. Bu durumu ortadan kaldırmak için başlangıçta oluşturulacak olan kromozom değerlerine kısıt konulmalıdır. Gerçek kodlu GA kullanılarak yapılan bu çalışmada, kısıtlamayı sağlayan program kodunda kullanılan formül Denklem 2.6'daki gibidir.

$$kromozom = (randomsayı(\quad) * (UB - LB) + LB) \tag{2.6}$$

$$UB = \text{Üst sınır}$$

$$LB = Alt\ sınır$$

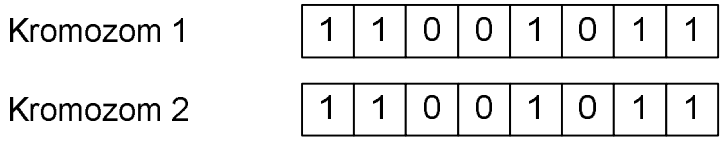
Genetik algoritmalarda problemlerin çözümüne ulaşmadaki en önemli kriterler den biride kromozomların kodlanması işlemidir. Bu durum GA'nın gerçek çözüme ulaşabilmesini ve çalışma hızını belirler. Kromozomların kodlanmasında problem türüne göre farklı yöntemler kullanılmaktadır.

2.3.3 Kromozomların Kodlanması

Bir GA yazılımının gerçekleştirilmesi işleminde belirlenmesi gereken ilk husus kromozomların kodlanmasıdır. Bu işlem problemin türüne göre değişim göstermektedir. Bazı problemlerde en iyi sonucu binary (ikili) kodlama verirken, bazen de permutasyon veya değer kodlama yöntemleri vermektedir.

2.3.3.1. İkili (Binary) Kodlama

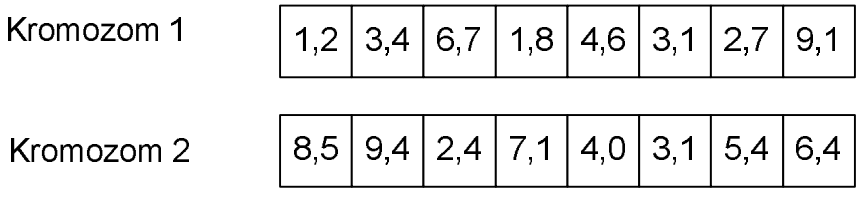
GA programlarının ilk kodlanmasından yana kullanılan bu yöntem, günümüzde de en çok kullanılan yöntemidir. Bu kodlamada, popülasyonu oluşturan kromozomların her biri 0 ve 1'ler ile ifade edilerek kodlanırlar. Sonuç olarak kodlanan her kromozom bir sayıya karşılık gelmektedir [32].



Şekil 2.4. İkili kodlama için kromozom örneği.

2.3.3.2. Değer Kodlama - Gerçek Kod

Gerçek gibi karmaşık sayıların da yer aldığı problemlerde genellikle değer kodlama (real code) yöntemi kullanılır. Karmaşık sayıların ikili kodlanması oldukça zordur. Burada her kromozom, bazı değerler dizisidir ve bu değerler problemle ilişkilidir. Örneğin gerçek sayı, karakter veya karmaşık sayılar olabilir. Bu tür kodlama, yapay sinir ağları için ağırlıkların hesaplanmasında yaygın olarak kullanılmaktadır[32].



Şekil 2.5. Değer kodlama için kromozom örneği.

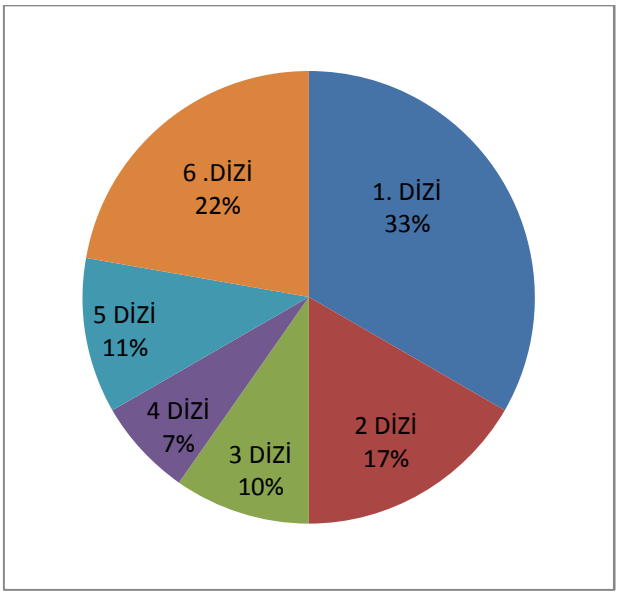
2.3.4 Uygunluk Fonksiyonu ve Seçme

Popülasyon içerisindeki her bireyi temsil eden kromozomların ne kadar iyi olduklarını bulan fonksiyona uygunluk fonksiyonu (fitness function) denir. Bu kısım her probleme göre farklılık gösterdiğinden GA programında özel çalışan tek kısımdır. Uygunluk elde eline çözümlerin her birini uygunluk fonksiyonuna tek tek uygulayarak bu kromozomların uygunluklarını hesaplar. Elde edilen sonuçlara göre kromozomların yeni jenerasyon da olup olmayacaklarına karar verilir. Aynı zamanda uygunluğun hesaplanmasıyla sonlandırma kriterleri de kontrol edilerek istenilen çözüme ulaşılıp ulaşılmadığına da bakılır[33]. GA'nın sonlanması için gerekli jenerasyon ve en iyi sonucu bulma durumu gerçekleşmemiş ise genetik operatörler olan seçme, çaprazlama ve kopyalama işlemine devam edilir.

Kromozomlardan yeni ve daha iyi kromozomlar oluşturmak için çaprazlama işlemine geçilir. Çaprazlama uygulanacak olan bireylerin seçilmesinde ise birden fazla yöntem kullanılmaktadır. Bu yöntemlerden en çok kullanılanları Rulet tekerleği seçimi (Roulette Wheel Selection), ve Turnuva seçimi (Tournament Selection) yöntemleridir.

2.3.4.1. Rulet Tekerleği Yöntemi

Bu yöntem en basit seçim yöntemi olarak bilinmektedir ve yaygın olarak kullanılan yöntemdir. Bu seçim yönteminde uygunluk değeri en yüksek olan bireyin seçilme ihtimali daha yüksek olması esasına dayanmaktadır. Her kromozom uygunluk değeri ile orantılı bir olasılık değeri hesaplanır ve rulet tekerleği dilimleri oluşturulur. Rulet tekerleği N defa döndürülür ve her seferde bir kromozom eşleme havuzuna atılır. Daha iyi diziler tekerleğin daha büyük bir bölümünü kapladığından iyi bireylerin seçilme olasılıkları daha yüksektir. Şekil 2.6'da örnek bir rulet tekerleği gösterilmiştir. Görüldüğü gibi kromozomlar uygunluklarına göre tekerlek üzerinde belirtilmiştir. Buradan 1. Kromozom dizisinin seçilme olasılığının en yüksek, 4. Kromozom dizisinin ise en düşük olduğu görülmektedir. Minimizasyon için tercih edilmez[34].



Şekil 2.6. Rulet tekerleği.

2.3.4.2. Turnuva Yöntemi

Turnuva secim yönteminde popülasyondan rastgele n adet birey seçilir. Seçilen n adet birey turnuvaya katılarak aralarından en iyi uygunluktaki birey seçilir. Turnuva sonucunda seçilen en iyi uygunluktaki bireyler çaprazlamaya girerler.

Seçme yöntemleri problemin türüne ve yapısına göre farklılıklar göstermektedir. Burada çözüme ulaşmak için en iyi yöntemin hangisinin olduğu iyi belirlenmelidir. Ayrıca kullanılan yöntemler bir birleri arasında problemin çeşidine göre farklı performans sonuçları üretebilmektedir. Paralel çalışmalarda turnuva yöntemi tercih edilir [34].

2.3.5 Çaprazlama

Genetik algoritmalarda çaprazlama operatörü, kromozomlardan çocuk kromozomlar üretmek ve yeni bireyler elde etmektir. Elde edilen çocuk kromozomların, ebeveynlerinden daha iyi sonuçlar elde edileceği tahmin edilerek çaprazlama yapılır. Çaprazlamada bireyler n adet rast gele seçilirler ve çaprazlama işlemine tabi tutulurlar. Buradaki önemli bir noktada çaprazlamada kullanılacak olan yöntemin kromozomlar üzerine etkisidir. Bu noktada problem iyi tanımlanmalı ve çözüm için en uygun çaprazlama yöntemi seçilmelidir. Günümüzde genetik algoritma problemlerinde kullanılan çok çeşitli çaprazlama türleri vardır. Uygulamada en çok kullanılan yöntemlerden ikili kodlamada ve değer kodlamada çaprazlama aşağıda açıklanmıştır.

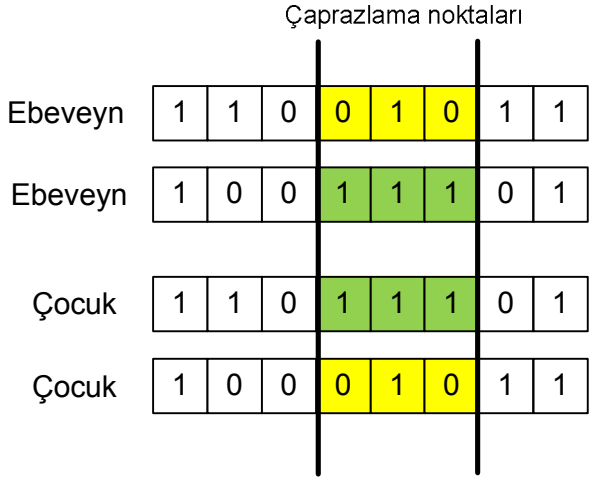
2.3.5.1. İkili Kodlamada Çaprazlama

İkili kodlamada bir çok çaprazlama yöntemi kullanılmaktadır. Bunlardan en yaygın olarak kullanılanı tek noktalı, iki noktalı, çok noktalı ve aritmetiksel çaprazlamadır.

Tek noktalı çaprazlamada kromozomlar rastgele bir noktadan kesilirler ve elde edilen bölümlerden her biri karşılıklı olarak yer değiştirir. Elde edilen yeni bireylerin daha iyi bireyler olması ihtimali olduğu gibi daha kötü olabilme ihtimalide vardır. Buradaki önemli husus çaprazlama yapılacak noktanın seçimidir. Tek noktalı çaprazlamaya örnek Şekil 2.7'de gösterildiği gibidir.

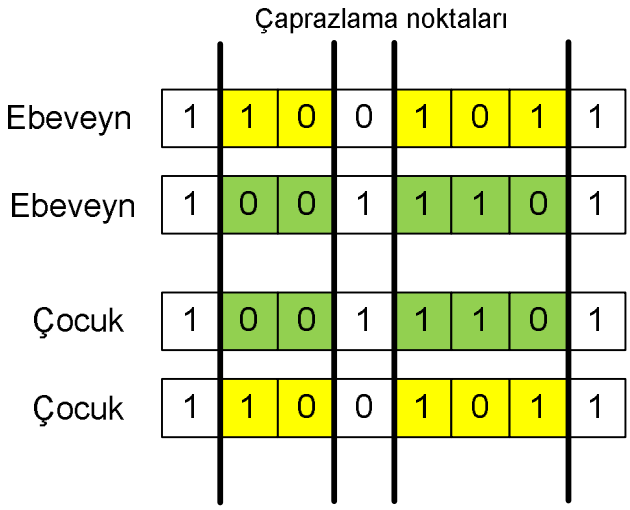
Şekil 2.7. Tek noktalı çaprazlama.

İki noktalı çaprazlamada ise rastgele iki nokta seçilir ve bu iki nokta arsasında kalan bölgeler yer değiştirilir. İki noktalı çaprazlamaya örnek Şekil 2.8'de gösterildiği gibidir.



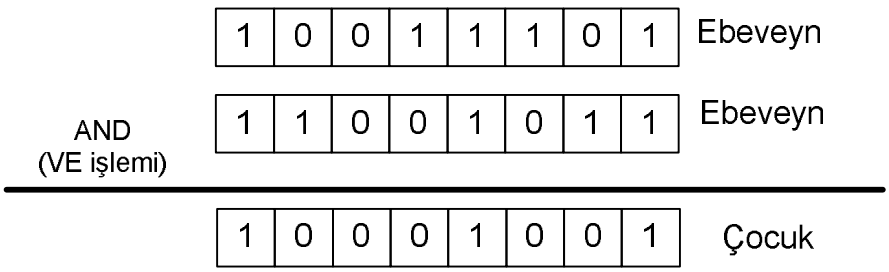
Şekil 2.8. İki noktalı çaprazlama.

Çok noktalı çaprazlama da ise kromozomlar daha fazla sayıda parçalara ayrılırlar ve parça araları sırayla yer değiştirilir[32,26].



Şekil 2.9. Çok noktalı çaprazlama.

Aritmetiksel çaprazlamada yeni nesiller elde edebilmek için seçilen kromozomlar bir takım aritmetik işlemlere tabi tutulur ve yeni bireyler elde edilir[23,26,32].



Şekil 2.10. İkili kodlamada aritmetiksel çaprazlama.

2.3.5.2. Değer Kodlamada Çaprazlama

Değer kodlama çaprazlamada, ikili kodlamada kullanılan tek noktalı, iki noktalı ve aritmetiksel gibi çaprazlama yöntemleri kullanılabilir.

Tek ve iki noktalı çaprazlamada ikili kodlamadaki gibi rastgele noktalar belirlenir ve kromozomu oluşturan genler karşılıklı olarak yer değiştirilir.

Aritmetiksel çaprazlama işleminde ise kromozomlar bir takım aritmetiksel işlemlere tabi tutularak yeni bireyler elde edilir.

Yapılan bu tez çalışmasında da aritmetiksel çaprazlama yöntemi kullanılmıştır. 0 - 1 aralığında rastgele c değeri belirlenmiş, bu değer çaprazlama oranı Pc den büyük ise yine rastgele iki kromozom seçilmiş, MSE değerleri kıyaslanmış ve bireyler Denklem 2.7' deki yöntemle çaprazlanarak yeni bireyler elde edilmiştir[6].

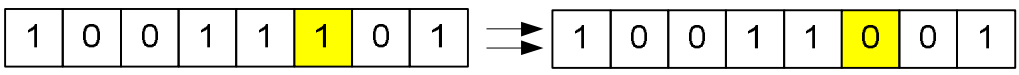
$$
\begin{aligned}
&if\ c > pc \\
&if\ MSE(\theta_1) < MSE(\theta_2) \\
&\qquad \theta_1' = \theta_1 + r(\theta_1 - \theta_2)\,, \theta_2' = \theta_2 + r(\theta_1 - \theta_2) \\
&else \\
&\qquad \theta_1' = \theta_1 + r(\theta_2 - \theta_1)\,, \theta_2' = \theta_2 + r(\theta_2 - \theta_1)
\end{aligned}
\tag{2.7}
$$

2.3.6 Mutasyon

2.3.5.3. İkili Kodlamada Mutasyon

Çaprazlama işleminden sonra genetik algoritmada uygulanan operatör mutasyondur. Mutasyonun amacı genetik çeşitliliği sağlamaktır. Ayrıca çözüm kümesinin sürekli aynı kromozomlardan oluşması gibi durumları ortadan kaldırmak için kullanılır. Yerel optimuma yakınsamayı engelleyebilir. Uyum değeri yüksek bireyleri kaybetmemek amacıyla mutasyon oranı oldukça çok düşük tutulur.

Mutasyon işlemi ikili sayı sisteminde kodlanmış bir bireyin, rasgele seçilmiş bir veya birkaç bit, 0'dan 1'e veya 1'den 0'a değiştirilir.

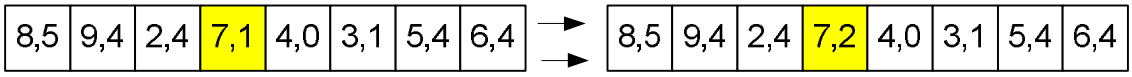


Şekil 2.11. İkili kodlamada mutasyon.

2.3.6.1. Değer Kodlamada Mutasyon

Değer kodlama yönteminde mutasyon işlemi için kromozomu oluşturan bireylerden

küçük bir değeri ekleme ve çıkartma yaparak elde edilir. Yapılan bu çalışmada da bu yöntem kullanılmıştır.



Şekil 2.12. Değer kodlamada mutasyon işlemi.

Mutasyon işlemi için Denklem 2.8 kullanılmıştır. Denklemdeki σ mutasyon işleminde bireyin işleme tabi tutulacağı değeri göstermektedir[6].

$$yeni\ birey = birey + \sigma * \Phi \tag{2.8}$$

2.3.7 Elitizm

Genetik algoritmalarda bir nesilde elde edilen bireylerin en iyilerinin seçilerek, genetik algoritma operatörlerinde hiçbir işleme tabi tutulmadan bir sonraki nesle aktarılmasına elitizm denir. Elitizmin amacı en iyi bireylerin bir sonraki nesillere aktarılarak korunmasını sağlamaktır [26,35].

2.3.8 Genetik Algoritmada Kullanılan Parametreler

2.3.8.1. Popülasyon Büyüklüğü (N)

Genetik Algoritmalarda ilk karar verilmesi gereken hususların başında popülasyon büyüklüğü gelmektedir. Bu parametre genetik algoritmanın etkinliğini ve performansını önemli derecede etkilemektedir. Seçilen değer küçük olursa, optimum değere ulaşmada yerel bir minimum değerine takılarak sonuca ulaşmasını engelleyebilir. ayrıca popülasyon büyüklüğünün sonuca ulaşmak için gerekli olan ilterasyon sayısının artmasına neden olabilmektedir. Diğer yandan popülasyon büyüklüğünün gereğinden büyük seçilmesiyle arama derinleşir ve işlem karmaşıklıkları ortaya çıkabilir, sonuca ulaşma süresi arttırabilir. Sonuç olarak topluluk büyüklüğü, ele alınan probleme göre dengeli bir değer seçilmedir [29].

2.3.8.2. Çaprazlama Oranı (Pc)

Çaprazlamanın amacı mevcut kromozomların özelliklerini birleştirerek çözüme uygun daha iyi bireyler elde etmektir. Çaprazlama oranı çaprazlanacak olan bireylerin sayısını

kontrol eder. GA'da her nesilde Pc x N adet diziye çaprazlama işlemine tabi tutulur. Büyük çaprazlama oranı, neslin değişikliğini hızlandırıp, istenilen optimum sonuçlardan uzaklaşılmasına neden olabilir. Çaprazlama oranının seçimi problemin türlerine göre uygun bir değerde seçilmelidir[26,36,34].

2.3.8.3. Mutasyon Oranı (Pm)

Mutasyonun amacı popülasyondaki genetik çeşitliliği sağlamaktır. Yeni popülasyonlardaki her dizinin her elemanı mutasyon oranına eşit bir olasılıkla rastgele olarak değişime uğratılır. Mutasyon oranı çözüme yakın bireylerin bozulmaması için oldukça düşük tutulur. Mutasyon oranının genetik algoritma üzerindeki etkisini araştırmak için yapılan çalışmalarda bu oranın 0,1 ile 0,001 aralığında tutulması ile sonuca ulaşmada olumlu etki yaptığı belirtilmiştir[26,36,34].

2.3.8.4. Elitizm Oranı (Pe)

Elitizm bir popülasyon da, seçilen en iyi bireylerin bir sonraki nesle aktarılmasıdır. En iyi bireylerin adedini oluşturan değere elitizm oranı denir. Elitizm oranı Pe x N şeklinde belirlenir. Elitizm oranın düşük seçilmesi çözüme ulaşmada geçen süreyi arttıracağı gibi, bu oranın yüksek seçilmesi de popülasyonda ki kötü bireylerinde bir sonra ki nesle aktarılmasını sağlayacaktır[35].

2.4. PARALEL GENETİK ALGORİTMALAR

Klasik sıralı genetik algoritmalar küçük boyutlu problemlerde oldukça iyi sonuçlar vermesine rağmen, problemin boyutu büyüdüğünde iyi bir performansa sahip değildir. Bu nedenler genetik algoritmaların performans arttırılmasına yönelik çeşitli çalışmalar yapılmıştır. Bu çalışmalar genel olarak yeni ve geliştirilmiş operatörlerin önerilmesi, kontrol parametrelerinin optimum düzeyde belirlenmesi, hibrid ve paralel yapılar içeren genetik algoritma modelleri üzerine olmuştur[37].

Genetik algoritmalarda işlemciye en fazla ihtiyaç duyulan kısım, üretilen kromozomların problemin sonucuna yönelik değerlendirilmelerinin yapıldığı uygunluk fonksiyonunun çalıştırılmasıdır. Seri GA'lar çok farklı alanlarda başarıyla kullanılmasına rağmen bazı dezavantajları vardır. Bunların başında popülasyonun büyüklüğü gelmektedir. Bazı problemlerin çözümünde büyük popülasyon seçilmesi gerekmekte ve bu durumda büyük hafıza alanları ve oldukça fazla işlem karmaşıklığını

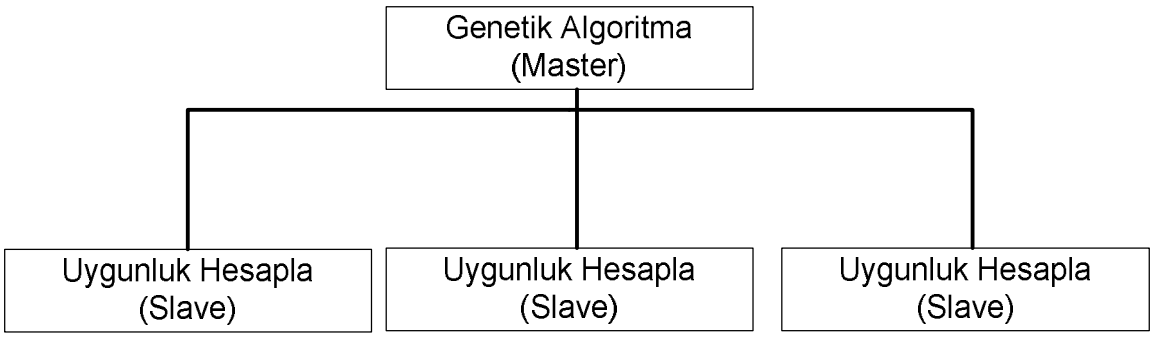
arttırmaktadır. Bu yüzden genetik algoritmaların bazı paralel modelleri gerekmektedir. GA'nın paralelleşmesini gerektiren ikinci husus, uygunluk değerlendirilmesinin yapılması işlemidir. Bazı çalışmalarda, bu işlemin oldukça uzun zaman dilimleri gerektiği gözlemlenmiştir[38,39]. GA'nın Paralelleşmesini gerektiren diğer bir hususta seri GA'lardaki arama uzayının yeteri kadar taranmamasına bağlı olarak, aramanın bir noktaya takılıp kalmasıdır. Paralel genetik algoritmalarda ise arama uzayını farklı alt bölgelerle yapılmasıyla, alt optimal çözüme takılma gibi riskli durumlar ortadan kaldırılmıştır.

Yapılan çalışmalarda Paralel Genetik Algoritmalarla ilgili birçok PGA modeli bulunmaktadır. Tüm PGA modellerini içeren bir sınıflandırma yapmak oldukça zordur. Ancak birçok PGA'nın içerdiği teknik, yapılması gereken çok sayıdaki işi daha küçük iş parçacıklarına bölerek işlemlerin eş zamanlı olarak gerçekleşmesini sağlamaktır. Bu yaklaşım bir çok farklı modelle uygulanabilmekte ve birden fazla işlemci üzerinde PGA farklı metotlarla gerçekleştirilmektedir. PGA'lar dört ana sınıfta Yönetici-yardımcı PGA'lar, Çok popülasyonlu PGA'lar, İnce-taneli PGA'lar Hiyerarşik melez PGA'lar şeklinde incelenmektedir [39].

Gerçekleştirilen bu tez çalışmasında da çok çekirdekli işlemci ile gerçekleştirme kolaylığı bakımından Yönetici-Yardımcı PGA metodu kullanılmıştır.

2.4.1 Yönetici-Yardımcı Paralel Genetik Algoritmalar

Tek bir popülasyona sahip olan Yönetici-Yardımcı (master-slave) PGA lar da bir yönetici genetik algoritmayı çalıştırır. Kromozomların uygunluk değerlerinin hesaplanması işlemini ise çeşitli işlemcilere dağıtarak işlerin paralel şekilde eş zamanlı olarak gerçekleşmesini sağlar[39]. İşlemciler yöneticinin verdiği işlemleri yerine getirirler ve sonucu geriye döndürdüler. Böylece çözüm kümesini oluşturan kromozamlar eş zamanlı olarak değerlendirilmiş olurlar. Bu tip PGAlarda genetik operatörler (seçim, çaprazlama ve mutasyon)popülasyon bir bütün olarak ele alındıklarından, bu yönteme "glaobal" PGA'lar da denilmektedir[21]. Şekil 2.13'de yönetici-yardımcı PGAların yapısını gösterilmektedir.



Şekil 2.13 Yönetici-Yardımcı (Master-Slave) paralel genetik algoritma yapısı.

Yönetici-yardımcı PGA'ların çalışması temel olarak incelendiğinde standart GA yaklaşımından farkı yoktur. Aralarındaki tek fark, yardımcı PGA'ların işlem yükünü farklı işlemcilere dağıtması ve bu şekilde daha hızlı çalışmasıdır.

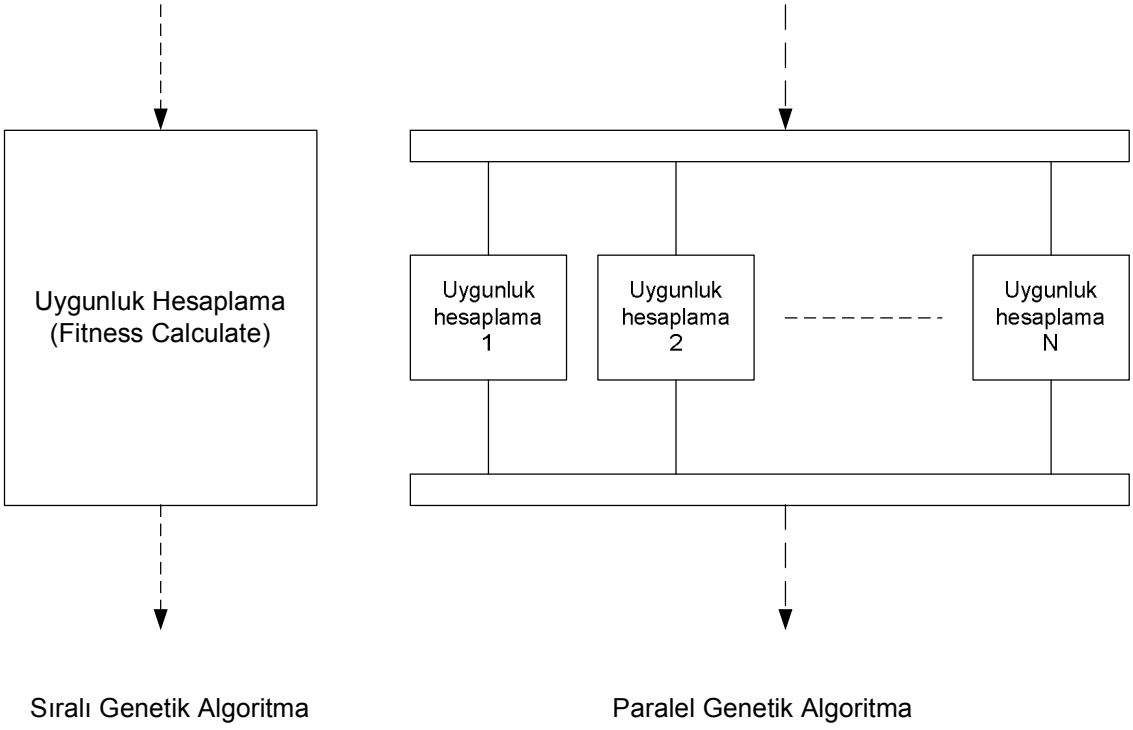
Bu yöntemde çalışma süresi değerlendirilmesi iki bileşenden oluşmaktadır. Bunlar işlerin yapılması için harcanan süre ve işlemciler arsındaki iletişim için harcanan süredir. İşlemci sayısı arttıkça uygunluk hesaplanması için harcanan süre azalırken, iletişim süresi artacaktır[21].

GA'ların paralel çalışması üzerine yapılmış ilk çalışma 1976 yılında Bethke tarafından yapılmıştır. Bu çalışmada uygunluk değerlerinin belirlenmesi için harcanan süre, genetik operatörler için harcanan süreye göre fazla olduğunda önemli kazanımlar sağlanabileceği gösterilmiştir. Ancak iletişim süresi göz ardı edilmiştir[40]. Grefenstette 1981 yılında yaptığı bir çalışmada, dört farklı PGA önermiştir. Bunlardan ilk üçü yönetici-yardımcı PGA, dördüncüsü ise çok popülasyonlu PGA'dır. Grefenstette bu çalışmasında, çok popülasyonlu PGA'ların yönetici-yardımcı PGA'lar ile birleştirilebileceğini belirtmiştir[41].

Yönetici-yardımcı PGA çalışmalarının diğer bir örneği de Fogarty ve Huang'ın 1991 yaptığı bir çalışmadır. Bu çalışmada işlemciler arasındaki iletişimi tarif eden farklı topolojiler denenmiştir ve işlem süresinin iletişim süresine baskın geldiği durumlarda işlemciler için kullanılan farklı topolojilerin önemsiz olduğu vurgulanmıştır. Ayrıca işlemci sayısı arttıkça iletişim süresinin hızlı bir artış gösterdiği vurgulanarak, belirli bir noktadan sonra işlemci sayısını arttırmanın işlem süresindeki kazanımları engelleyebileceği belirtilmiştir[42]. Genel olarak uygunluk değerlerinin belirlenmesi

için gereken işlem süresi arttıkça, yönetici-yardımcı PGAların etkinliği de artmaktadır[43, 39]. Yönetici-yardımcı PGA'lardaki süre kazanımının her bir işlemci için (işin yapılma süresi – iletişim süresi) olduğu düşünülecek olursa, işin yapılma süresi arttıkça yönetici PGAnın verimliliğinin de artması beklenen bir durumdur.

Gerçekleştirilen bu tez çalışmasında, PGA yapılarından Yönetici-Yardımcı yöntemi kullanılmıştır. Sıralı çalışan genetik algoritmada hesap karmaşıklığının olduğu ve uzun zaman alan uygunluk hesabının yapıldığı bölüm, işlemci çekirdek adedinde iş parçalarına bölünerek paralelleştirilme sağlanmıştır. Şekil 2.14'te sıralı çalışan GA'daki uygunluk hesabının, PGA'daki çalışma yapısı gösterilmektedir.



Şekil 2.14. GA'daki paralelleştirilen bölge yapısı.

Yukarıdaki uygunluk hesaplama işleminin seri gerçekleştirilmesinde klasik for yapısı, paralel biçimde kodlamasında ise paralel for yapısı kullanılmıştır. Gerçekleştirilen bu iki yöntemde kullanılan for yapılarına ait C# kod bloğu da aşağıdaki gibi gerçekleştirilmiştir.

```
for (int i = 0; i < pop.GetLength(0); i++)
        {
          Uygunluk[i] = uygunlukfonksiyonu(katsayı1,katsayı2, katsayı3, katsayı4);
        }
```

Şekil 2.15. Seri for yapısı C# kod bloğu.

```
Parallel.For(0, pop.GetLength(0), i =>
        {
        Uygunluk[i] = uygunlukfonksiyonu(katsayı1,katsayı2, katsayı3, katsayı4);
        });
```

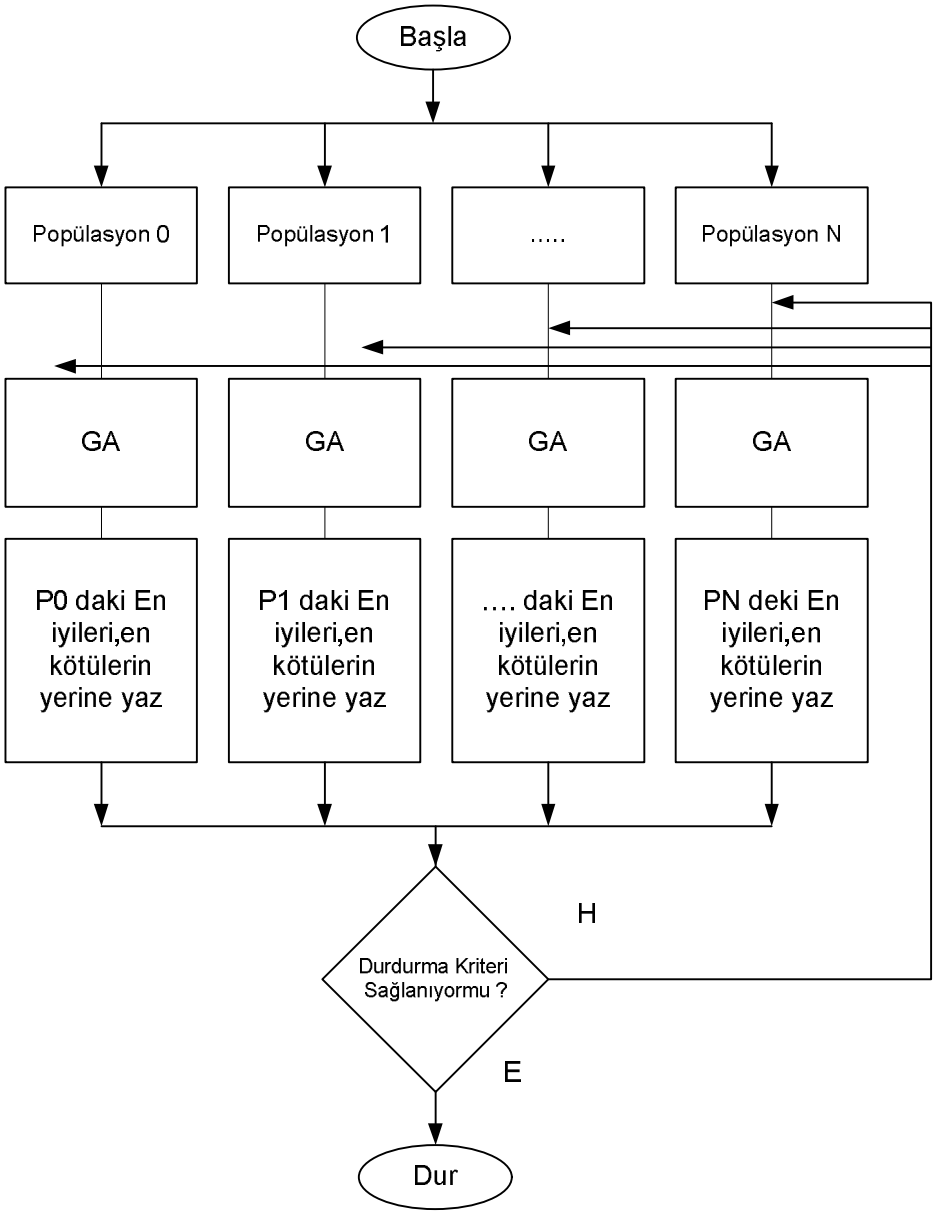
Şekil 2.16. Paralel for yapısı C# kod bloğu.

2.4.2 Çok Popülasyonlu Paralel Genetik Algoritmalar

Çok popülasyonlu PGA'lar, GA'ların çalışma sistemlerine, birden fazla popülasyonla çalışmak ve popülasyonlar arası bilgi alışverişi sağlamak olmak üzere iki yenilik katmıştır.

Çok popülasyonlu GA'lar birden fazla popülasyon üzerinde çalışmaktadır. Her popülasyon üzerinde bir GA çalışır ve her popülasyonun arama uzayında farklı bölgeleri örneklemesi beklenir. Bu farklı popülasyonların aralarında haberleştirilmesi yoluyla da daha kaliteli çözümlere ulaşacak kromozomların elde edilmesi sağlanmaktadır.

Çok popülasyonlu PGAlar, popülasyonlar arası bilgi alışverişini sağlamak için bir topolojiden faydalanırlar. Topoloji hangi popülasyonların birbirleriyle iletişimde bulunacaklarını tanımlayan bir yapıdır. Bu sayede topolojiye uygun olan iki popülasyon arasında bilgi alış verişi gerçekleştirilir. Ayrıca GA'lar arası iletişim için bir genetik operatörde kullanılmaktadır. Şekil 2.17'de örnek çok popülasyonlu GA yapısı verilmiştir[39].



Şekil 2.17. Çok popülasyonlu genetik algoritma genel yapısı.

Çok popülasyonlu evrimsel algoritmalara ilişkin bulunabilen ilk çalışma 1967 yılında Bossert tarafında yapılmıştır. Bossert'in çalışmasında popülasyonlar yaşamlarını sürdürebilmek için birbirleri ile rekabet etmektedirler. En kötü popülasyonu rastgele belirlenen periyotlarla yok etmiş, yok edilen popülasyon yerine rastgele bir popülasyon oluşturmuştur[44].

Çok popülasyonlu PGA'larda ise, Grefenstette popülasyonlar arasındaki bilgi akışını saglamak için her bir popülasyondaki en iyi bireyleri her jenerasyonda tüm popülasyonlara dağıtmıştır. Gerçekleştirilen bu yaklaşımda popülasyonlar arası iletişim

oldukça yüksektir. Bu sayede popülasyonların hızlı bir şekilde birbirine yakınsaması söz konusudur. Bunun bir sonucu olarak ta Grefenstette, popülasyonlar arası bilgi alıverişinin sıklığı, kromozomların topolojide hangi yöne aktarılması ve bu işlemin erken yakınsama üzerindeki etkileri gibi konular üzerine çalışılması gerektiğini belirtmiştir[45].

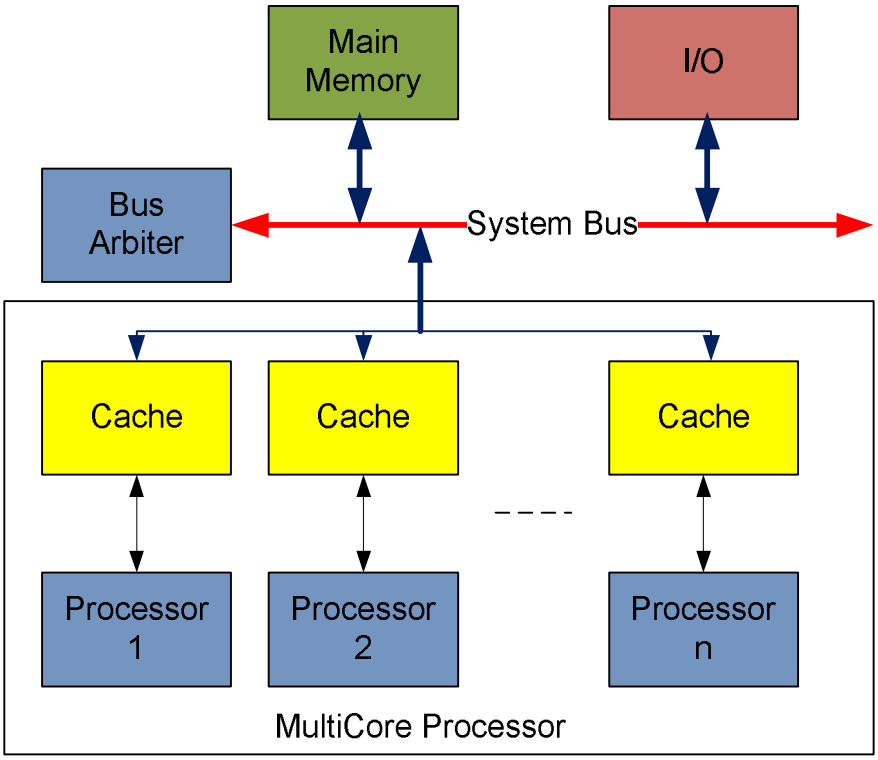
1985 yılında Grosso tarafından yapılan bir çalışmada, tek popülasyonlu bir yapının, alt popülasyonlara bölünmüş yapıya göre uygunluk değerlerindeki artış oranının daha yüksek olduğu gözlemlenmiştir. Ayrıca bölünen popülasyonların birbirinden bağımsız çalışması durumunda alt popülasyonlarda ki çeşitlilik az olduğundan, elde edilen çözümler toplamda aynı kromozom sayısına sahip tekbir popülasyonlu yapıda elde edilen sonuçlardan daha kötü olduğu belirlenmiştir. Ayrıca Grosso tarafından yapılan bu çalışmada popülasyonlar arası iletişimin sıklığı üzerine de çeşitli yaklaşımlar izlenerek denemeler yapılmıştır. İlk olarak, göç edecek kromozomlar ve hangi popülasyona göç edecekleri rastgele belirlenerek popülasyonlar arası göçe her jenerasyonda izin verilmiştir. İkinci denemede ise "geciktirilmiş göç" operatörünü kullanarak popülasyonların kendi içlerinde ki yakınsamalarını tamamlamalarını beklemiş ve daha sonra yüksek bir oranda iletişim oranı ile iletişimde bulunmalarına izin vermiştir. Yapılan bu iki denemede elde dilen sonuçlara göre iki durumda da çözüm kalitelerinin aynı olduğu belirtilmiştir[43].

Munetomo tarafından 1993 yılında yapılan bir çalışmada da, Grosso'nun önerdiği "geciktirilmiş göç" operatörünü farklı bir yaklaşımla ele alınarak, popülasyon içerisindeki kromozomların çeşitliliği belirli bir değerin altına düştüğünde popülasyonlar arasında iletişim kurulmasına izin verilmiştir. Ayrıca uygulanan bu yöntem ile dinamik topoloji yapısı kullanılmış, çeşitliliği en düşük popülasyona göç işlemi yapılmıştır[46].

Starkweather 1991 yılında yaptığı bir çalışmada, kısmi bir şekilde bağımsız çalışan popülasyonların arama uzayının farklı bölgelerini taradığını farketmiş ve popülasyonlar arası göç operatörü ile bilgi alışverişi sağlayarak daha iyi çözümlerin elde edildiği yüksek kaliteli kromozomlar elde edildiği belirtilmiştir[47]. Elde edilen bu sonuçlar bulgular daha sonra 1991 yılında Rudolph[48] ve 1999 yılında Whitley[49] tarafından yapılan çalışmalar la tutarlılık göstermiş ve onaylanmıştır.

2.5. PARALEL PROGRAMLAMA

Günümüzde hızla gelişen teknolojiyle, üretilen donanımlar yazılımların ihtiyaçlarını karşılamakta zorlanmaktadırlar. Moore yasasına göre transistör sayısı her iki yılda iki atına çıkacak denilmiştir. Ancak Transistör sayısındaki bu artış güç güç problemini ortaya çıkarmıştır[50]. Ayrıca donanımdaki hafıza ve bit çözünürlüğü rahatlıkla arttırılabilirken işlemci hızı ise fiziksel limitlere ulaşmıştır. Donanım üreticileri bu durum karşısında işlemci hızı yerine kullanılan işlemcideki çekirdek(core) sayısını arttırmıştır. Şekil 2.18'de görüldüğü gibi çok çekirdekli işlemci içinde çekirdek olarak adlandırılan bağımsız işlemciler tek bir paket içerisinde toplanmış olup bu paket üzerinden sistem yoluna bağlanmıştır.



Şekil 2.18. Çok çekirdekli bilgisayar donanımı.

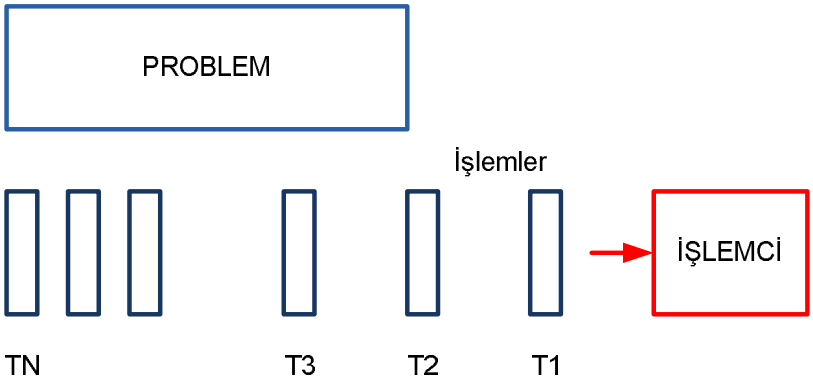
Çok çekirdekli işlemciler sadece sunucularda yada yüksek performans gerektiren yerlerde değil evlerdeki kişisel bilgisayarlarda, taşınabilir bilgisayarlarda, cep telefonlarında dahi yaygınlaşması paralel programlamaya duyulan ihtiyacı ve bu alanda yapılan araştırmaların önemini her geçen gün arttırmaktadır[51].

Bilgisayar yazılımları çok çekirdekli işlemcilerden yararlanabilmesi için paralel program yazılımlarını kullanmalıdır. Bir yazılımın nasıl paralelleştirileceği ise yazılımın

kendisiyle ilgilidir. Genelllikle her yazılım çeşitli algoritmalarla paralelleştirilebilir. Günümüzde en çok kullanılan 5 temel algoritma bulunmaktadır. Bunlar böl ve yönet, paralel işaretçi, randomizasyon, simetri kırılması ve yük dengeleme algoritmalarıdır [52,53]. Sonuç olarak her paralel programlama algoritmasının kendine özgü kullanım alanları mevcuttur. Yazılımın yapısına göre hangi paralelleştirme algoritmasının kullanılacağı da farklılık göstermektedir.

Çok çekirdekli bilgisayarlar da yazılımların paralelleştirilmesi için kullanılan algoritmaların yanında paralelleştirme yazılımları da oldukça fazla önem taşımaktadır. Son yıllarda en çok tercih edilen paralelleştirme yazılımları nesneye dayalı programlama yöntemi elde edilmektedir. Paralel nesneye dayalı yazılım geliştirmek için günümüzde en çok tercih edilen yazılımlar Cilk++, MPI, OpenMP gibi araçlardır. Bu araçların yanında yazılım firmalarının ürettikleri Therading.Task gibi paralelleştirme kütüphaneleri de kullanılmaktadır. Bu araçlar çok işparçacıklı (multithreading) yapı ile paralelliği sağlamaktadır.

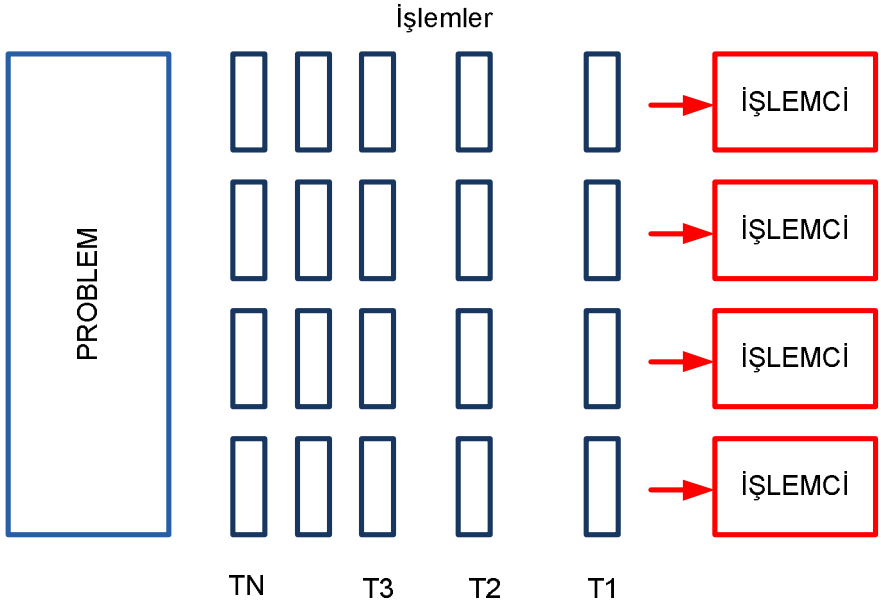
Gerçekleştirilecek olan işlemin tekbir işlemci üzerinde çalıştırılması işlemine "Seri Programla" denir. Bu işlemde aynı anda sadece tekbir işlem yapılabilmektedir. Bir işlem bitmeden diğer bir işlemin çalışması söz konusu değildir[54]. Seri programlama mantığı Şekil 2.19'da gösterilmiştir.



Şekil 2.19. Seri programlama çalışma mantığı.

Seri programlamada her işlem sırayla işleme tabi tutulduğundan her bir adım için ayrı bir zaman gerekmektedir. Bu durumda yapılan işlemin sonuçlanmasında performans ve verimlilik gibi kısıtlamaları da beraberinde getirmektedir.

Paralel programlama ise birden fazla işlemcinin, bir işlemin gerçekleştirilmesine yönelik aynı anda çalıştırılmasıdır. Bu işlem çoklu çekirdek teknolojisine sahip bir işlemci üzerinde gerçekleştirilir. Gerçekleştirilecek olan işlem parçalara ayrılarak, her parça farklı bir işlemci üzerinde çalıştırılarak tüm parçalar aynı zamanda çözülebilir[54]. Paralel programlamanın çalışma mantığı, Şekil 2.20'de gösterilmiştir.



Şekil 2.20. Paralel programlama çalışma mantığı.

Paralel programlamada işlemin parçalara bölünmesiyle, işlemin tamamlanması için geçen sürede parçalara bölmekte ve her bir parça eş zamanlı olarak farklı çekirdekler üzerinde çalıştırılmaktadır. Böylece her bir işlemciye düşen işlem sayısı azaltılarak işlem süresini azaltmak mümkün olmaktadır [55].

2.5.1 Temel Paralelleştirme Yapıları

Paralel programlama mantığında paralellik işlemci içinde ve işlemci dışında da gerçekleştirilebilir. Bilgisayar sistemlerinde paralellik (Bit-level parallelism), komut düzeyinde paralellik (Instruction-level parallelism), veri paralelliği (Data parallelism) ve görev paralelliği (Task parallelism) olmak üzere 4 sınıfa ayrılmaktadır[56]. İşlemci içindeki paralellik bir komutun işlenme aşamaları ve komutlar arası ölçekte olmaktadır. Budan dolayı bit ve komut düzeyi paralellik işlemci içinde gerçekleştirilir. Veri ve görev paylaşımlı paralellik ise işlemci dışında gerçekleşmektedir. İşlemci içi paralellik maliyet ve kullanım sınırlılıkları acısında esnek ve ekonomik değildir.

2.5.1.1. Bit düzeyinde paralellik (Bit Level Parallelism)

İşlemci içerisindeki bir saat darbesinde işlenebilen bit miktarı bit düzeyindeki paralellik düzeyini belirtmektedir. Eski nesil işlemcilerde bir saat darbesinde 4 bit veri işlenebiliyorken günümüzdeki işlemcilerde aynı anda 64 bitlik veri işlenebilmektedir. Günümüzde bu düzeyde işlemciler yaygın olarak kullanılmaktadır.

2.5.1.2. Komut Düzeyinde Paralellik (Instruction-Level Parallelism)

Bir yazılım gerçekte, bir işlemci tarafından yürütülen komut adımları akışıdır. Bu akış şekliyle yazılımın sonucu değiştirilmeden komutlar paralel olarak yürütülebilir. Komut düzeyinde paralellik birden fazla komutun aynı anda işlenmesi temeline dayanır. Bu işlemci içinde iş hattı (pipeline) teknolojisi ile gerçekleştirilir. Bu iş hattı teknolojisi sayesinde işlemci içerisinde aynı anda 5 adet komut çalıştırılabilmektedir.

2.5.1.3. Veri Düzeyinde Paralellik (Data Parallelism)

Veri düzeyinde paralellik, paralel olarak işlenecek farklı işlem düğümleri arasında veri dağıtımına yönelik çalışan bir yöntemdir. Bu paralellik program döngülerinde karşımıza çıkar. Her bir işlemci dağıtılmış verilerin farklı parçaları üzerinde aynı görev yaptığında paralellik sağlanmış olur[57].

2.5.1.4. Görev Paylaşımlı (Task parallelism) Paralel Sistemlerin Sınıflandırılması

Görev paylaşımlı paralel sitemlerde ise yazılıma ait her bir fonksiyon farklı işlemciler üzerinde çalışır. Bu yüzden görev paylaşımlı paralel sistemler ortak bellekli (shared memory) ve dağıtık bellekli (distributed memory) olmak üzere iki ana sınıfa ayrılmaktadır. Görev paylaşımlı paralel sistemler CPU ve GPU üzerinde denenebilme özelliğine de sahiptir.

2.5.1.5. C# İle Paralel Programlama

C#, Microsoft .NET platformu için geliştirilmiş bir programlama dilidir. C/C++ ve Java dillerinden türetilmiş, güçlü, yazım biçimi basit, oldukça esnek bir yazılım dilidir.

Programlama dillerinin seviyesi yükseldikçe bilgisayara yaptırılacak işlemler için daha alt kategoride bulunan diller ile yazılmış kütüphanelere yani hazır kod bloklarına daha

çok ihtiyaç duyulur. Diller kategorilere ayrıldığında, script dilleri en yüksek seviye programlama dilleridir. Örnek olarak (JavaScript, VBScript gibi) diller verilebilir. C# programlama dili diller arasında orta düzeyli diller kategorisinde yer almaktadır. En düşük düzeyli dil olarak Assembly yani makina dilini örnek verilebilir. C# programlama dili ile hem alt düzey hem de üst düzey programlar yazılabilir. C# 'ın söz dizimi Java ve C/C++ gibi dillerden gelmekte ayrıca bu dillerin en iyi ve en kuvvetli yönlerini de bünyesine almıştır.

Özellikle kurumsal büyük ve kapsamlı projeler geliştirirken yazılım dilinin verimli ve çok zor olmaması gereklidir. Bu sebeple Microsoft'un geliştirmiş olduğu C# bu gibi projelere iyi bir şekilde uyum sağlamaktadır. C# Nesne Yönelimli Programlamaya tam destek verir. Nesne yönelimli (object oriented) bir dildir. Bu sayede büyük ölçekli projeleri hızlı bir şekilde geliştirebilir. XML desteği sağlar. Yönetilmeyen Kod (Unmanaged Code) desteği vardır. Bu özellik sayesinde C# eski com ve dll kütüphaneleri ile etkileşime geçebilir bu kütüphanelerin .NET dll haline getirilme zorunluluğu yoktur. DLL (Dinamik Kod Kütüphaneleri) yazılabilir ayrıca C# programları için, yada .NET desteği olan diller için içinde sınıf (class) olan kod kütüphaneleri yazılabilmektedir. ASP.NET te C# ile yazılmıştır. Bu sayede Web ortamında çalışacak ASP.Net web formları C# kullanılarak yazılabilmektedir. Konsol uygulamaları geliştirilebilir. Ayrıca mobil geliştirme aracı ile, Windows işletim sistemi tabanlı cep telefonlarına, PDA, Tablet gibi mobil cihazlara, C# ve .NET uygulamaları rahatlıkla geliştirilebilmektedir.

C# ile Windows işletim sistemi için kolay bir şekilde uygulama geliştirilebilmektedir. Bunun için uygulamanın çalıştığı sistemde .NET Framework platformunun yüklü olması yeterlidir[58,59].

Günümüzde işlemci teknolojisinin hızlı gelişmesiyle artan paralel programlamanın önemi, programlama dillerini hazırlayan programcılarında dikkatini çekmiş ve uygulamaların programcıdan bağımsız şekilde paralel çalışmasını sağlayacak yapılar üretilmesi konusunda çalışmalara yönlendirmiştir. Microsoft .NET Framework yazılımcıları da bu bağlamda framework yapısına Parallel sınıfı, Task'lar, TaskFactory, eş zamanlı olarak güvenli çalışabilen koleksiyon sınıfları ve PLINQ adını verdikleri LINQ'in paralel versiyonunu eklemişlerdir.

Gerçekleştirilen bu tez çalışmasında da .Net Framework 4.5 yapısına ait Parallel sınıfı ve Task yapıları kullanılmıştır.

2.5.2 C# Parallel Sınıfı

C# Paralel sınıfında üç adet metod bulunmaktadır. Her üçü de çalışmaları sırasında çalıştırılan kodu durdurmaktadırlar ve kod ancak tüm işler tamamlandığında bir alt satırdan çalışmasına devam etmektedir.

2.5.2.1. Parallel.Invoke

Bu metod ile 'n' sayıdaki işi aynı anda çalıştırabiliriz. Tüm işler çalışmasını bitirdikten sonra bu metodu çalıştırdığımız yerin bir alt satırındaki kod çalıştırılmaktadır. Parallel.Invoke metodu bir Action delege dizisini paralel olarak çalıştırmaktadır. Bu Action'ların kesinlikle birbirleri ile ilgili olmamaları gerekmektedir. Parallel.Invoke metodu ile çok sayıda Action'ı da başarı ile çalıştırılabilir. Ancak burada actionların içindeki kodlar yazılırken iş parçacıkları açısından güvenli olmalarına özen gösterilmelidir. Zira iş parçacıklarının çakışması durumları ortaya çıkabilmektedir.

Kullanım Örneği:

```
Parallel.Invoke(
() => İşlem_fonksiyonu_2()
() => İşlem_fonksiyonu_2()
);
```

2.5.2.2. Parallel.For ve Parallel.ForEach

Bu iki metod C#'ın for ve foreach komutları ile aynı şekilde çalışmaktadır. Tek fark her iterasyonun eş zamanlı olarak çalıştırılıyor olmasıdır. Normal çalışan bir for döngüsünün, Parallel.For için kullanım örneği aşağıdaki gibidir.

```
for (int i = 0; i < myList.Count; i++) {
// program kodları
}

Parallel.For( 0, myList.Count, (i) => {
// program kodları
});
```

Normal for yapısına ait kodlar for'un ‘i’ değerine bağlı olarak sıra ile çalışmasını sağlar. Ancak parallel.for yapısı 'i' değerinin farklı değerlerini iş parçacıklarına göndererek döngünün işlemci özelliklerine bağlı olarak bir çok iş parçacığında eş zamanlı olarak çalışmasını sağlamaktadır.

Foreach ve Parallel.ForEach döngüsüde aynı yapı ile çalışmaktadır. Foreach ve Parallel.ForEach döngüsünün örnek kod yazılımı da aşağıdaki gibidir.

```
foreach( var item in myList ) {
// Program Kodları
}


Parallel.ForEach( myList, (i) => {
// Program Kodları
});
```

Bu iki yeni versiyon döngüde iterasyonların eş zamanlı çalıştırılmasıyla bilgisayarın sahip olduğu çekirdek sayısına da bağlı olarak performansı arttırılabilmekte. Ancak bazı durumlarda performans artışı yerine performans düşüşü de gösterebilmektedir. Bu nedenle paralel döngülerin kullanacağı yerler iyi belirlenmelidir.

2.5.2.3. ParallelLoopState

ParallelLoopState sınıfı, parallel.for veya forEach döngünüzün durumunu takip etme olanağı ve/veya döngüden çıkma şansı sunmak için hazırlanmıştır. ParallelLoopState yapısı ve metodları aşağıdaki gibidir.

```
public class ParallelLoopState
{
  public void Break();
  public void Stop();

  public bool IsExceptional { get; }
  public bool IsStopped { get; }
  public long? LowestBreakIteration { get; }
  public bool ShouldExitCurrentIteration { get; }
}
```

Break: Bu metot geleneksel döngülerdeki break komutu ile aynı amaçla kullanılmak üzere hazırlanmıştır. Bu metodu çağırıldığında o anki iterasyondan çıkılır ancak diğer iterasyonlar ise çalışmalarına devam ederler.

Stop: Bu metot ise geleneksel döngülerdeki exit komutu gibi çalışmakta ve döngüden

tamamen çıkmak için kullanılır. Ancak bu metod çağırdığında gerçekleşecek olan olaylar biraz farklılık göstermektedir. Bu metodu çalıştırdığında o anki iterasyondan sonraki tüm iş parçacıkları durmaya zorlanacaktır. Stop metodu kullanmak bir kritere bağlı sonlandırmalarda kullanmak faydalıdır.

LowestBreakIteration: Bu değer Break metodunun çağrıldığı en küçük iterasyon değerini içermektedir. Eğer null değeri veriyorsa herhangi bir iterasyonda henüz Break metodunun çağrılmadığı anlaşılmaktadır. Bu değer işlem bittiğinde For döngüsü tarafından durdurulamayan iş parçacıklarının gereksiz olarak işlemciyi meşgul etmemesi amacı ile kullanabilir. Bunu yapmak için kodların başlangıcında bu değer kontrol edilir eğer null değerine sahip değilse döngüden hemen çıkılabilir.

ShouldExitCurrentIteration: Bu değer Stop metodu çalıştırıldığında true değeri almaktadır. Yine kodun başlangıcında bu değerin kontrol edilmesi ile işlemci zamanını boşa harcamaktan kurtarabilir.

IsExceptional: Bir iterasyonda bir hata oluşup olmadığını bildirmek için hazırlanmıştır[58,59,60,61].

3. BULGULAR VE TARTIŞMA

3.1. DENEYSEL ARAYÜZÜN TANITIMI

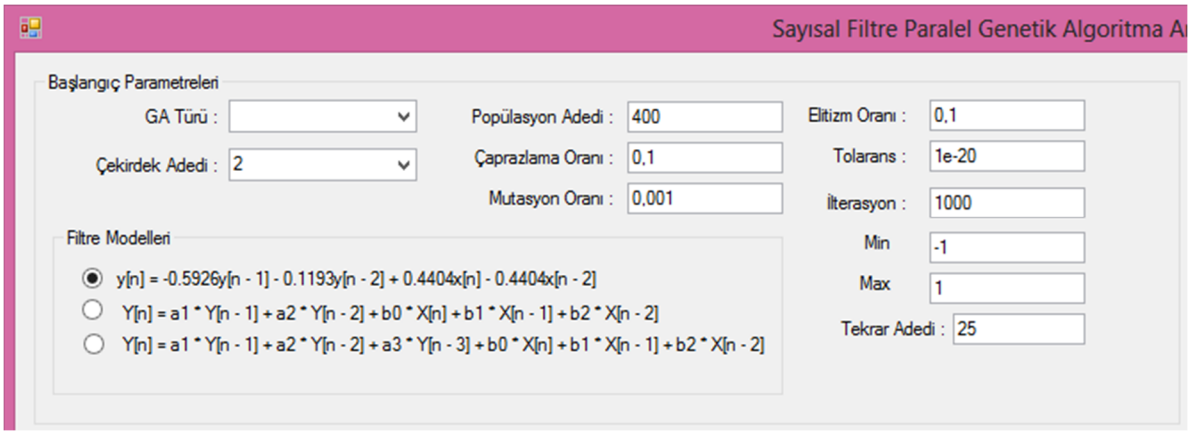
Nesne yönelimli programlama günümüzde oldukça yaygınlaşmış, yapılan çalışmalar genellikle nesne yönelimli bir programlama dili ile tasarlanmaya başlanmıştır. Günümüzde nesne yönelimli programlamada kullanabileceğimiz birden fazla program bulunmaktadır. Bunlardan en popüler olanı olarak Java, Matlab Gui, C# ve web tabanlı çalışmalarda kullanılabilecek olan Asp.Net olarak sıralayabiliriz.

Yapılan bu tez çalışmasında Microsoft firması tarafından 1999 yılı sonu itibariyle geliştirilmeye başlanmış olan Visual C# programlama dilinin, Visual Studio 2012 sürümü kullanılmıştır. Ayrıca paralelleştirme ve başarım analiz sonuçlarının elde edilmesinde .Net Framework 4.5 kullanılmıştır.

Visual studio 2012 derleyicisi kullanılarak gerçekleştirilen programın arayüz tasarımı Şekil 3.1'de görüldüğü gibidir. Arayüz de kullanılacak olan GA türünün ve GA parametrelerinin seçilebildiği başlangıç parametreleri bölümü, başarım analizinin gerçekleştirileceği filtre model seçimi, sonuçların grafiksel ve metin olarak gösterildiği bölümler mevcuttur.

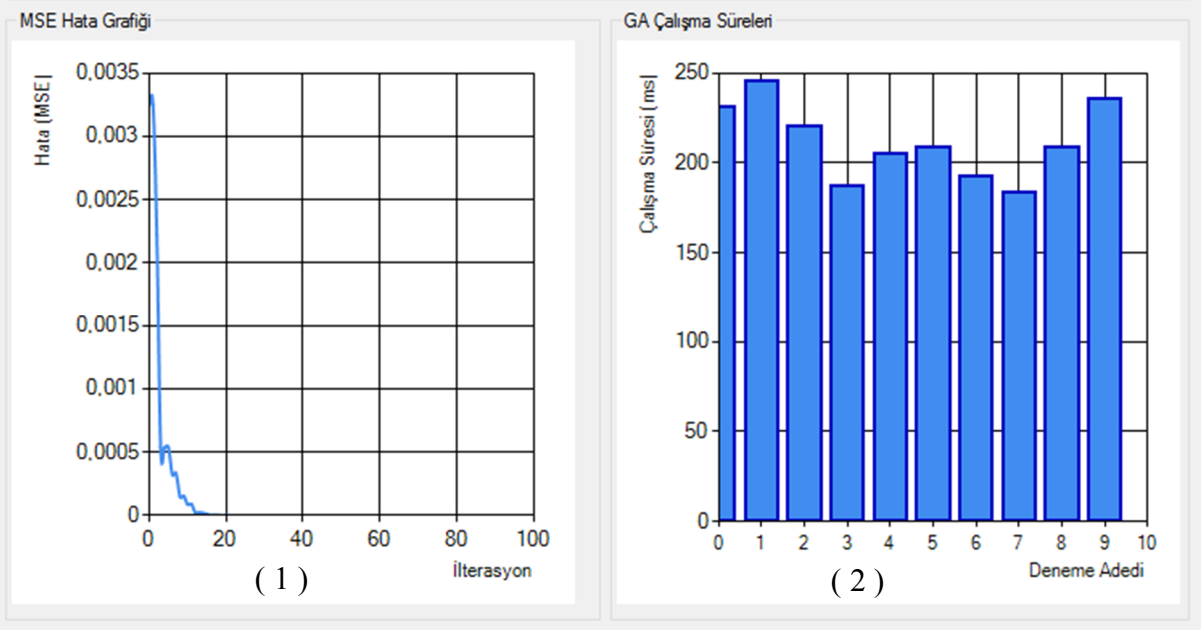
Şekil 3.1. Deneysel çalışmalar için gerçekleştirilen program arayüzü.

Şekil 3.2 de başlangıç parametrelerine ait bölüm gösterilmektedir. Tasarımda GA türüne ait seçimin yapılması için "Seri GA" , "Paralel GA" değerlerini içeren bir liste kutusu kullanılmıştır. Seçimin Seri GA seçilmesi durumunda herhangi bir paralelleştirme olamayacağından, çekirdek adedine ait liste gizlenmekte, seçimin "Paralel GA" seçilmesiyle de paralelleştirmenin gerçekleştirileceği çekirdek adedi seçimi için gerekli liste kutusu ekranda görünür olmaktadır. Arayüzün bu bölümünde GA yazılımında kullanılan parametre değerleri verilmiş ve ayrıca değiştirilmesine imkan sağlayacak şekilde tasarlanmıştır. Ayrıca arayüz üzerinde farklı başarım analizlerinin sağlanması için kodlanan farklı filtre modellerin seçilmesini sağlayacak olan radio butonlar kullanılmıştır.



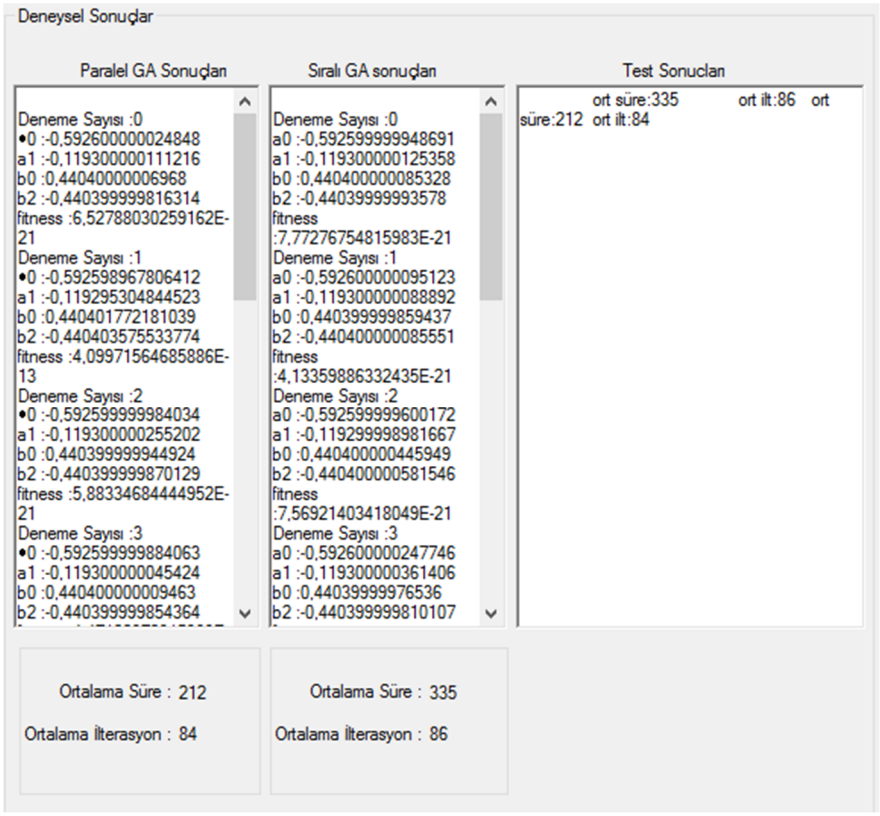
Şekil 3.2. Arayüz Başlangıç parametreleri.

Şekil 3.3'te arayüz programının çalıştırılması ile elde edilen grafiksel gösterimlerin yer aldığı bölüm görülmektedir. 1 numaralı grafik bölümünde GA nın çalışması ile her bir ilterasyon değerine ait olan MSE hata değerine dair değer grafiği gösterilmektedir. 2 numaralı grafik bölümünde ise başarım analizlerinde ki deneme adedine bağlı olarak elde edilen çalışma süreleri gösterilmiştir.



Şekil 3.3. Arayüz deneysel sonuç grafikleri bölümü.

Şekil 3.4 te GA'nın çalıştırılmasıyla hesaplanan katsayı değerleri paralel ve seri sonuçları olarak iki ayrı bölümde verilmiş, hesaplanan katsayılar, tolerans değerine ait olan değerler yazdırılmıştır. Ayrıca yapılan test sonuçlarının karşılaştırılabilmesi için, istenilen sayıdaki çalışma bilgileri kaydedilerek, yapılan teste ait genel sonuçlarının yazdırılması sağlanmıştır.

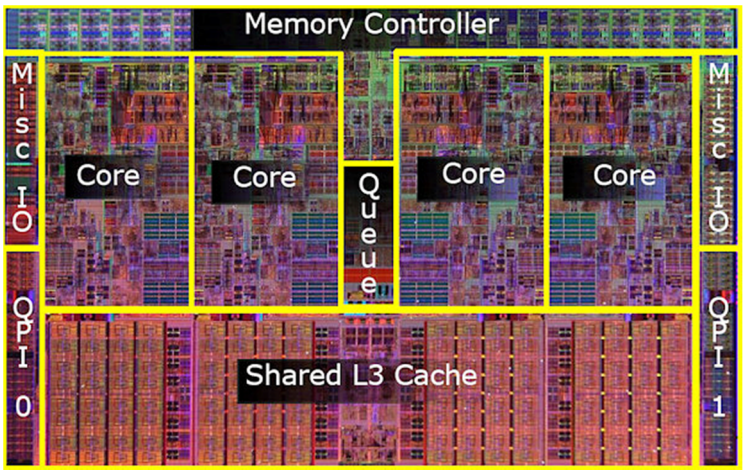


Şekil 3.4. Arayüz deneysel çalışma sonuçları bölümü.

Arayüz üzerinde bulunan "Başla" butonu ile seçilen GA türü ve parametrelerine dair bir defa çalışma sağlanmıştır. "Test" butonu ile ise belirlenen GA ve Çalışma parametrelerine dair Sıralı ve Paralel çalışma gerçekleştirilmesi sağlanmış ve sonuçların Test Sonuçları bölümünde gösterilmesi sağlanmıştır. "Sonuçları Temizle" butonuyla da geçmiş çalışmalara dair olan analiz sonuçlarının temizlenmesi sağlanmıştır.

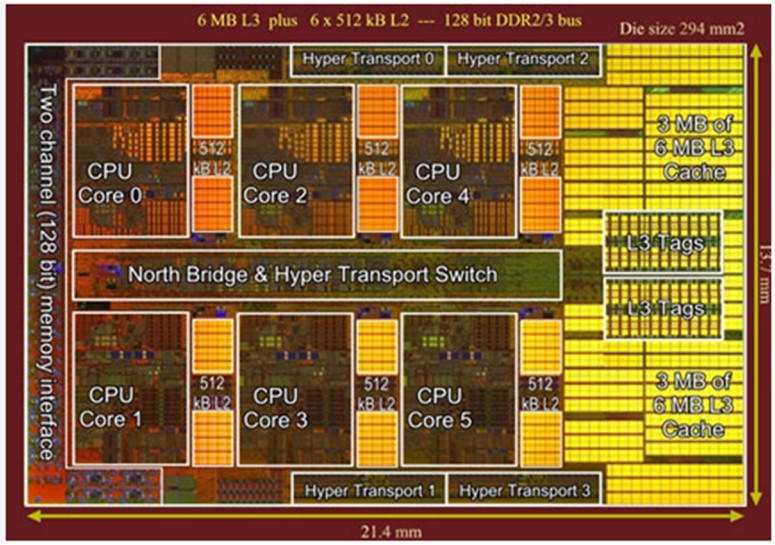
3.2. PARALLEL BAŞARIM ANALİZİ

Yapılan tez çalışmasında paralel başarım analizi çok çekirdekli bilgisayarlar üzerinde gerçekleştirilmiş, farklı özelliklere sahip Intel® Core™ i7-4950HQ ve AMD Phenom 1055T işlemcileri kullanılarak deneysel sonuçlar elde edilmiştir. Deneylerde kullanılan İntel i7 işlemcisinin iç yapısı Şekil 3.5'teki gibidir. İşlemci 6MB büyüklüğünde ön belleğe sahiptir. İşlemci dört gerçek çekirdek, dört sanal çekirdekten oluşmakta ve bu sayede Hyper Treading özelliği ile 8 iş parçacığı tanımlanabilmektedir. Deneysel çalışma da 4. Çekirdekten sonra elde edilen sonuçlar ile sanal çekirdeklerin paralelleştirmeye dair etkileri de gözlemlenmiştir.



Şekil 3.5. Deneylerde kullanılan İntel i7 işlemciye ait blok diyagramı[62].

Deneylerde kullanılan diğer bir işlemci olan AMD işlemcisinin iç yapısı Şekil 3.6'daki görülmektedir. Altı çekirdekten oluşan işlemcide çekirdek başına 128kb L1 512 KB L2 ön bellekleri ve çekirdekler arasında paylaşılan 6MB büyüklüğünde L3 ön belleğine sahiptir. Çalışma frekans 2.8GHz bellek 2 GB.



Şekil 3.6. Deneylerde kullanılan AMD işlemciye ait blok diyagram[63].

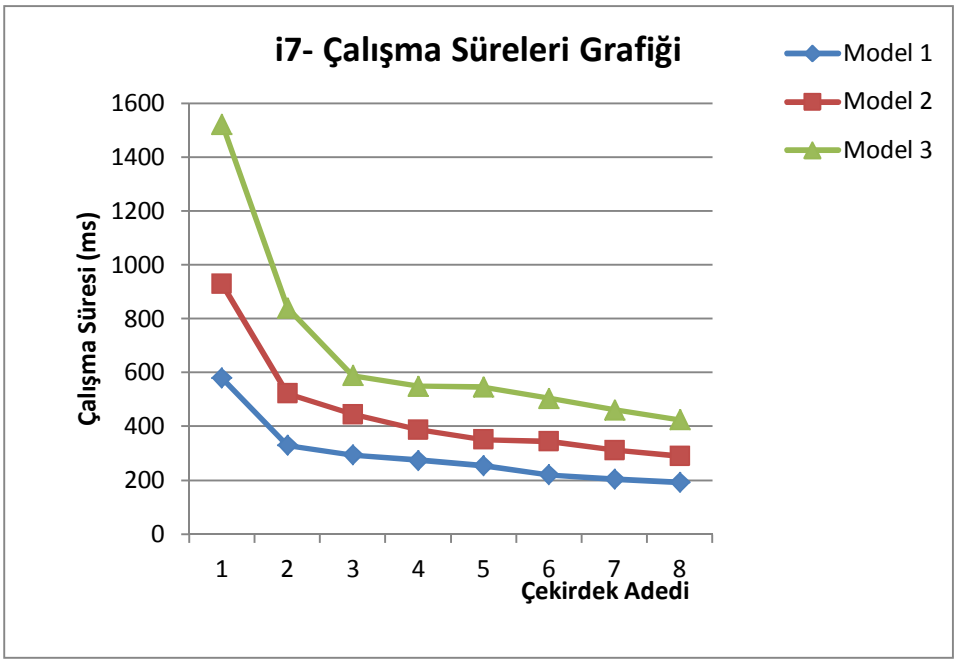
Yapılan deneysel paralelleştirme başarım analizlerinde popülasyon adedi 600 olarak seçilmiş, çaprazlama oranı (Pc) 0.3, mutasyon oranı (Pm) 0.1, elitizim oranı (Pe) 0.1, algoritmayı sonlandırma kriteri olarak hata değeri 1e-20 seçilmiştir. Çalışmalarda sonuçların elde edilmesini sağlayacak ilterasyon 5000 olarak kullanılmıştır. GA

rastlantısal bir arama metodu olduğundan sonuçlara ulaşmada farklı çalışma sürrelerinin elde edilmesi kaçınılmazdır. Bu açıdan ideal çalışma sürelerinin elde edilmesi için, her GA algoritması 500 defa çalıştırılmıştır.

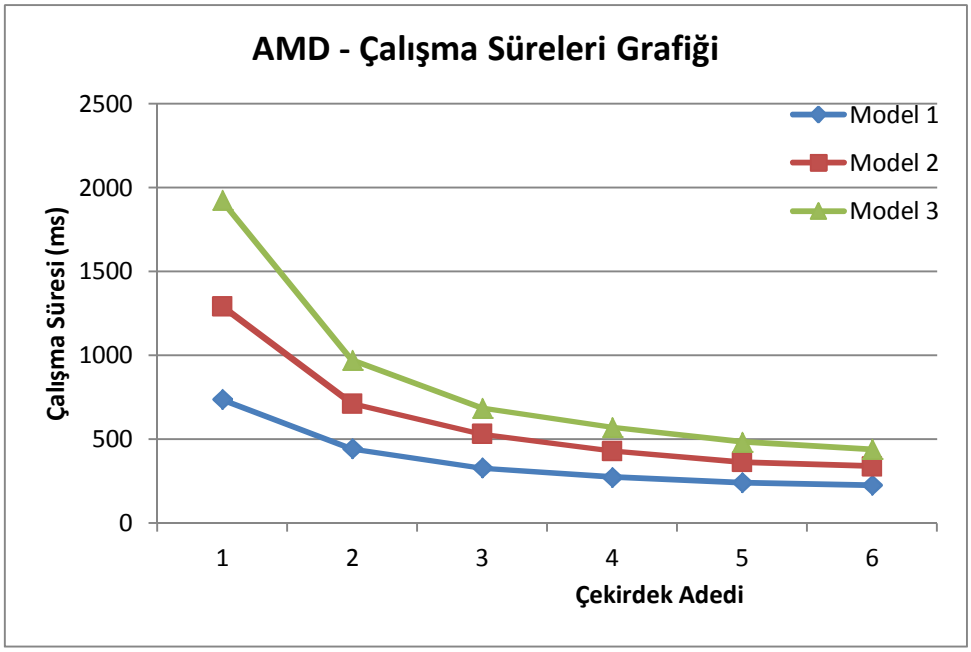
Çizelge 3.1'de i7 ve AMD işlemcisi için farklı filtre modellerine dair çalışma sürelerinin gösterildiği tablo verilmiş, Şekil 3.7 ve 3.8'de her bir işlemciye ait çalışma sürelerinden elde edilen hızlandırma süreleri grafiği verilmiştir. Elde edilen sonuçlar milisaniye (ms) cinsinden gösterilmiştir.

Çizelge 3.1. Farklı filtre modellerine ait çalışma süreleri çizelgesi (ms).

İNTEL i7 İŞLEMCİSİ								
GA Türü	**Seri**	**Paralel**						
Çekirdek Adedi	1	2	3	4	5	6	7	8
Model 1	372	196	141	138	114	123	111	115
Model 2	3699	1141	815	639	670	636	465	521
Model 3	26755	10493	5712	5150	4681	4817	3578	4038
AMD İŞLEMCİSİ								
GA Türü	**Seri**	**Paralel**						
Çekirdek Adedi	1	2	3	4	5	6	7	8
Model 1	737	442	327	272	240	224		
Model 2	1292	712	530	429	362	338		
Model 3	1924	971	684	571	482	439		



Şekil 3.7. i7 işlemcisi farklı filtre modellerine ait çalışma süreleri grafiği.

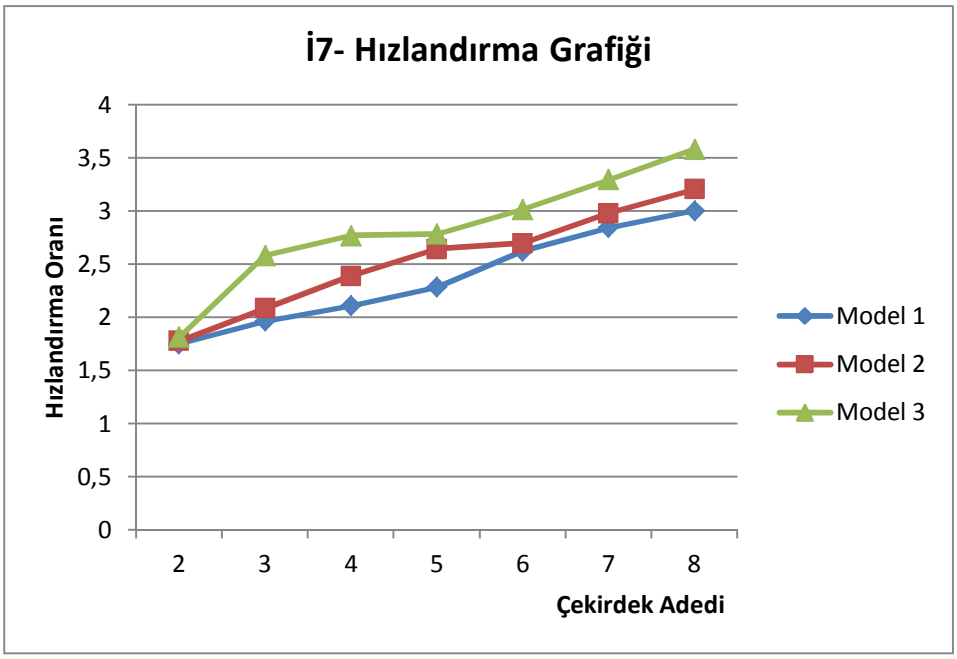


Şekil 3.8. AMD işlemcisi farklı filtre modellerine ait çalışma süreleri grafiği.

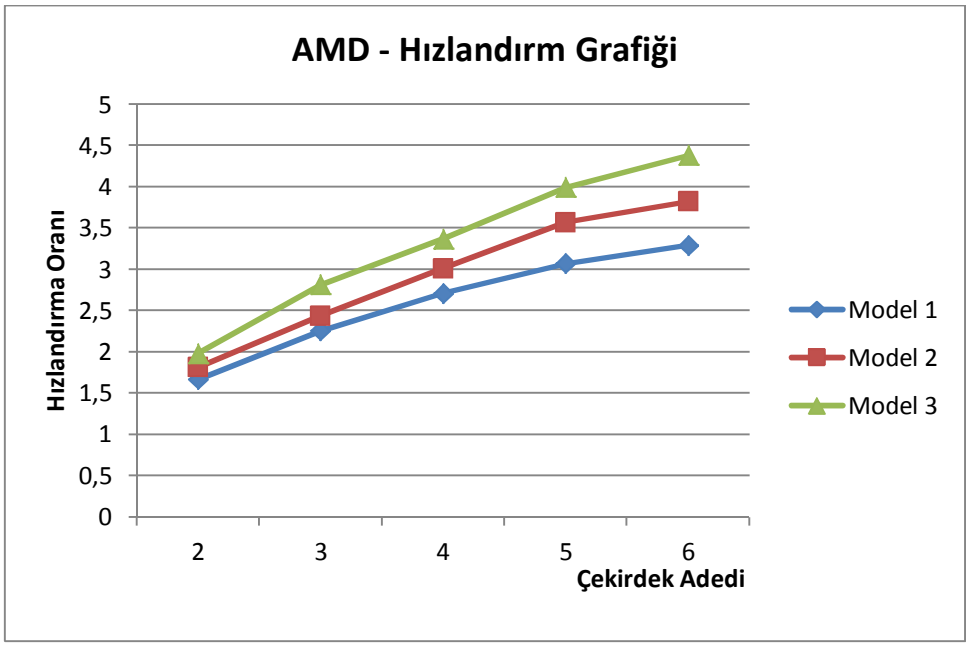
Çizelge 3.4'de de i7 ve AMD işlemcisi için farklı filtre modellerine ait paralel formda çalışan GA yapısının, sıralı çalışan GA yapısına oranla, elde edilen hızlandırma oranlarının gösterildiği tablo verilmiştir. Şekil 3.9 ve Şekil 3.10'da elde edilen hızlandırma oranlarına ait grafikler verilmiştir.

Çizelge 3.2. Hızlandırma oranları çizelgesi.

İNTEL İ7 İŞLEMCİSİ							
Çekirdek Ad.	2	3	4	5	6	7	8
Model 1	1,7523	1,966	2,109	2,283	2,624	2,843	3,005
Model 2	1,7782	2,085	2,391	2,642	2,696	2,981	3,207
Model 3	1,8131	2,581	2,769	2,784	3,016	3,297	3,584
AMD İŞLEMCİSİ							
Çekirdek Ad.	2	3	4	5	6		
Model 1	1,6674	2,254	2,71	3,071	3,29		
Model 2	1,8146	2,438	3,012	3,569	3,822		
Model 3	1,9815	2,813	3,37	3,992	4,383		



Şekil 3.9. i7 işlemcisi hızlandırma grafiği.

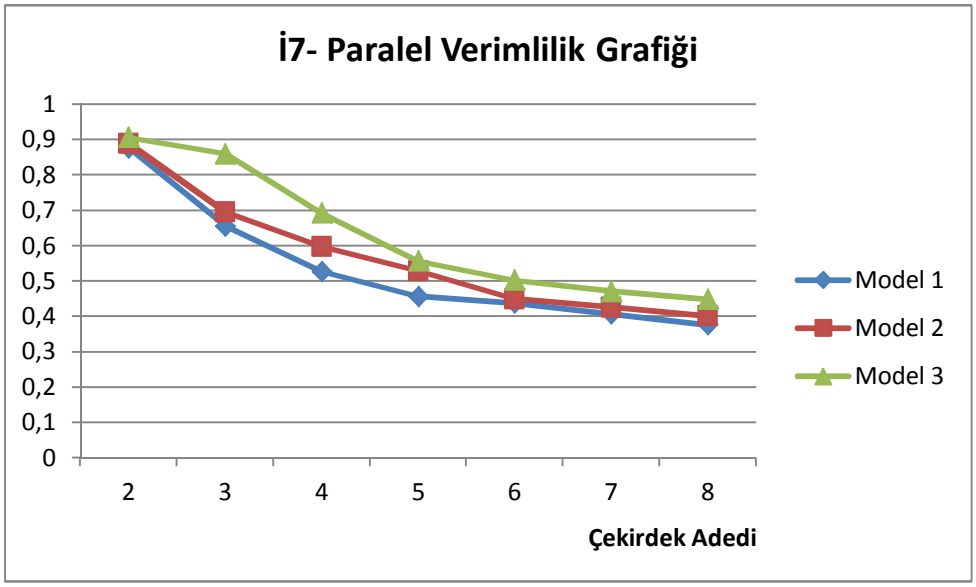


Şekil 3.10. AMD işlemcisi hızlandırma grafiği.

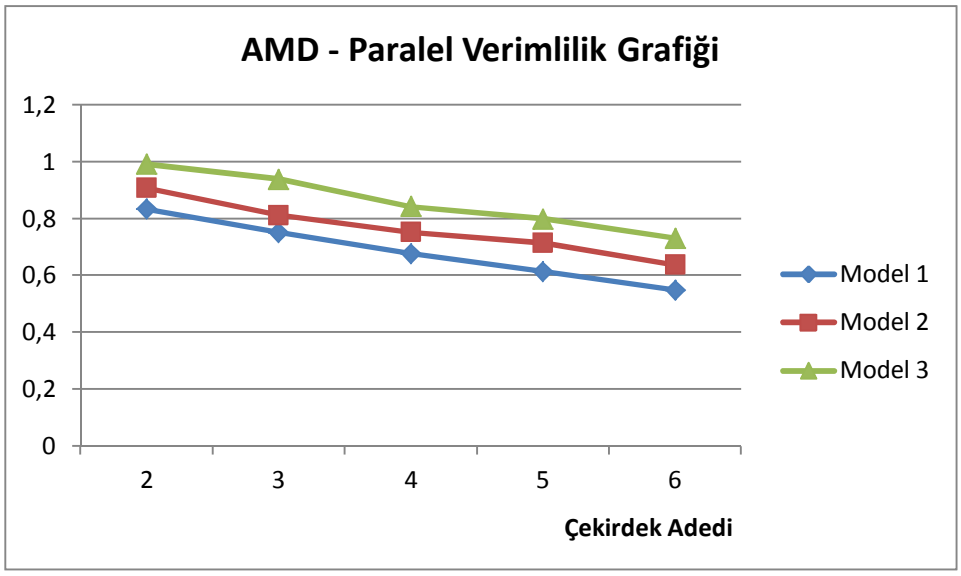
Yapılan deneysel çalışmaların sonucunda çizelge 3.4 incelendiğinde, tüm filtre modellerinden elde edilen sonuçlara göre, i7 işlemcisinde sıralı şekilde çalışan genetik algoritma yapısının 4 çekirdekli paralelleştirme sonucunda ortalama 2.7 kata kadar hızlandırma sağladığı, gözlemlenmiş, i7 işlemcisinin Hyper Treading özelliği sayesinde 3.5 kata kadar hızlandırma sağlanmıştır. Aynı paralelleştirme yöntemiyle AMD işlemcisinde elde edilen sonuçlar değerlendirildiğinde, 4,3 kata kadar hızlandırma elde edildiği görülmüştür.

Elde edilen sonuçlar filtre modelleri açısından değerlendirildiğinde 5 katsayıya sahip Model 2 ve 6 katsayıya sahip Model 3 te elde edilen hızlandırmaların daha performanslı olduğu gözlemlenmektedir. Bunun sebebi olarak paralelleştirme için harcanan sürenin toplam GA çalışma süresine ait oranına bağlı olduğu söylenebilir.

Şekil 3.11 ve Şekil 3.12'de çekirdek adedine bağlı paralel verimlilik grafikleri verilmiştir. Deneysel çalışmalar sonucu elde edilen paralel verimlilik grafikleri değerlendirildiğinde, hızlandırma grafiğinde elde edilen sonuçlarda olduğu gibi, çekirdek adedine bağlı olarak modelde kullanılan değişken adedi arttıkça elde edilen paralel verimliliğinde arttığı görülmektedir.



Şekil 3.11. i7 işlemcisi çekirdek adedine bağlı paralel verimlilik grafiği.



Şekil 3.12. AMD işlemcisi çekirdek adedine bağlı paralel verimlilik grafiği.

4. SONUÇLAR VE ÖNERİLER

Yapılan bu tez çalışmasında, sayısal filtre optimizasyonunun çok-çekirdekli bilgisayar üzerinde paralel genetik algoritmalar kullanılarak daha hızlı gerçekleştirilmesi sağlanmıştır. Deneysel sonuçların incelenmesinde algoritma parametrelerinin kontrol edilebileceği C# tabanlı bir arayüz hazırlanmıştır.

Paralel başarım analizi, 4 çekirdekli toplam 8 işparçacığına sahipIntel® Core™ i7-4950HQ ve 6 çekirdekten oluşan AMD Phenom 1055T işlemcisi kullanılarak elde edilmiştir

Yapılan deneysel çalışmalar sonucunda GA nın çekirdek adedine bağlı olarak paralelleştirilmesi sonucunda başarılı sonuçlar elde edilmiştir. Sıralı GA ile Paralel yapıda çalışsan GA arasında altı çekirdekli işlemci için yaklaşık olarak 4.3 kata kadar dört çekirdekli Hyper Threading destekli işlemci için 3.5 kat hızlandırma elde edilmiştir. Ayrıca işlemci teknolojilerinden Hyper-Treading özelliğinin de paralelleştirme işlemine önemli derecede fayda sağladığı gözlemlenmiştir.

Bundan sonraki çalışmalarda, farklı GA paralelleştirme yöntemleri ve farklı programlama dillerine ait paralelleştime yapıları kullanılarak, farklı paralelleştirme yöntemlerinin ve programlama dillerinin hız analizi yapılabilir.

5. KAYNAKLAR

[1] Kaya T., İnce M.C, , Sayısal Filtre Katsayılarının Genetik Algoritma Yardımıyla Hesaplanması, Fırat Üniv. Fen Ve Müh. Bil. Dergisi, 20(3), 459-466, 2008

[2] Kaya T., Genetik Algoritma İle Sayısal Filtre Tasarımı, Yüksek Lisans Tezi, Fırat Üniversitesi, Fen Bilimleri Enstitüsü, (**2006**).

[3] Davis L., Handbook of Genetic Algorithms, Van Nostrand, New York, (1991).

[4] Goldberg D.E., Genetic algorithms, In Search, Optimization and Machine Learning, Addison Wesley Reading, MA, (**1989**).

[5] GOLDBERG D.E., Genetic and evolutionary algorithms come of age,Communications of the ACM, vol.37,3, p.113–119,(**1994**).

[6] Wei-Der C., Coefficient Estimation of IIR Filter By A Multiple Crossover Genetic Algorithm, Computers & Mathematics with Applications, Volume 51, Issues 9–10, Pages 1437–1444 (**2006**).

[7] Kosir A., Tasic J. F., Genetic algorithm and filtering, Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA. First International Conference 343-348,(**1995**).

[8] Arslan, T., The design of analogue and digital filters using genetic algorithms, Digital and Analogue Filters and Filtering Systems, IEE 15th SARAGA Colloquium on, 2/1-2/5 (**1995**).

[9] Lee A., Ahmedi M., Jullien G.A., Miller W.C., Lashkar R.S., Digital Filtre Design Using Genetic Algorithm, IEEE, 34- 38,(**1998**).

[10] Adamıdıs P., Review of Parallel Genetic Algorithms Bibliography, Tech.rep. version1, Aristotle University of Thessaloniki, Thessaloniki, Greece, (**1994**).

[11] Lins C., Punch W., Coarse-Grain Parallel Genetic Algorithms: Categorization and New Approach. In Sixth IEEE Symposium on Parallel and Distributed Processing, IEEE Computer Society Press (Los Alamitos ,CA), (**1994**).

[12] GORDONV S., WHITLEY D., Serial and Parallel Genetic Algorithms as Function Optimizers, In FORRESTS., Ed., Proceedings of the Fifth International Conference on Genetic Algorithms, p.177–183, Morgan Kaufmann ,(**1993**).

[13] Avery Adam W., Translation Optimization And Parallelization Of Genetic

Algorithms A Discourse Of Implementation Into C++ , Senior Scholars Thesis, Computer Engineering, Texas A&M, (**2010**).

[14] Rausch T., Thomas A., Camp N. J., Cannon-Albright L. A., Facelli, J. C., A Parallel Genetic Algorithm to Discover Patterns in Genetic Markers That Indicate Predisposition to Multifactorial Disease, Computers in Biology and Medicine 38(7), 826-36, (**2008**).

[15] Kandil A., El-Rayes K., MACROS: Multiobjective Automated Construction Resource Optimization System, Journal of Management in Engineering, 22(3), 126-134, (**2006**).

[16] Wang N., Tsai C., Cha K., Optimum Design of Externally Pressurized Air Bearing Using Cluster OpenMP, Tribology International, 42, 1180-1186, 2009.

[17] Wang P., Wu X., OpenMP Programming for a Global Inverse Model, Scientific Programming, 10(3), 253-261, (**2002**).

[18] Mirghani B.Y., Mahinthakumar K.G., Tryby M.E., Ranjithan R.S., Zechman E.M., A Parallel Evolutionary Strategy Based Simulation–Optimization Approach for Solving Groundwater Source Identification Problems, Advances in Water Resources, 32(9), 1373-1385, (**2009**).

[19] Kegel P., Schellmann M., Gorlatch S, Using OpenMP Vs. Threading Building Blocks for Medical Imaging on Multi-Cores, LNCS, 5704, 654 - 665, 2009.

[20] Kuvat G., Paralel Genetik Algoritmalarda Göç Yöntemleri ve Göç Parametrelerinin Dinamik Olarak Belirlenmesi , Doktora Tezi, Eskişehir Osmangazi Üniversitesi, ElektrikElektronik Mühendisliği Anabilim Dalı, Ağustos (**2009**).

[21] Koç İ.O, Gezgin Satıcı Problemi için Çok Popülasyonlu Paralel Bir Genetik Algoritma Tasarımı, Geliştirilmesi ve Analizi, Doktora Tezi, Endüstri Mühendisliği Anabilim Dalı, Eskişehir Osmangazi Üniversitesi, Fen Bilimleri Enstitüsü Eylül (**2007**).

[22] Uşkay Selim O., Route Optimization For Solid Waste Transportation Using Parallel Hybrid Genetic Algorithms, The Middle East Technical University, December (**2010**).

[23] Aksu Ö., Yeni Bir Paralel Genetik Algoritma Modeli ve Analog Devre Tasarımına Uygulanması, Yüksek Lisans Tezi, Erciyes Üniversitesi Fen bilimleri Enstitüsü, (**2008**).

[24] Karaboğa N., Sayısal Filtre Katsayılarının Genetik Algoritma Kullanılarak

Yuvarlatılması, Doktora Tezi, Erciyes Üniversitesi, (**1994**).

[25] Kara A., Dsp Tabanlı Sayısal Filtre Tasatrımı Ve Uygulanması, Yüksek Lisans Tezi Süleyman Demirel Üniversitesi Fenbilimleri Enstitüsü, (**2008**).

[26] Batık Z., Sayısal Filtre Tasarım Yöntemleri Veperformans Analizleri, Yüksek Lisans Tezi, Sakarya Üniversitesi Fen Bilimleri Enstitüsü, (**2011**).

[27] Ertürk S., Sayısal işaret isleme, Birsen Yayınevi, İstanbul, 293s.

[28] Hsu Hwe P., Sinyaller Ve Sistemler, Nobel Yayın Dağıtım, (**2000**).

[29] PARLAK, M., Genetik Algoritmaların Hesapsal Ve Yapısal Olarak İncelenmesi, Yüksek Lisans Tezi, Ondokuz Mayıs Üniversitesi Fen Bilimleri Enstitüsü, (**2007)**.

[30] Karaboga D., Yapay Zeka Optimizasyon Algoritmaları, Atlas Yayın Dağıtım, İstanbul, (**2004**).

[31] Gürsu B., Genetik Algoritmalar ile DC-AC Çeviricilerde Harmonik Eliminasyonu, Yüksek Lisans Tezi, Fırat Üniversitesi Fen Bilimleri Enstitüsü, (**2002**).

[32] Cunkaş M.,Genetik Algoritmalar ve Uygulamaları Ders notları, Konya, (2010).

[33] Cemes R., Ait-boudaoud D., Genetic Approach to Design of Multiplierless FIR Filters, IEE, 2090-2091, (**1993**).

[34] Kaya T, Genetik Algoritma İle Sayısal Filtre Tasarımı, Fırat Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, (**2006**).

[35] Elmas Ç, Yapay Zeka uygulamaları, Seçkin Yayıncılık, 1.baskı, Ankara, (**2007**).

[36] ZEYVELi M., Genetik Algoritma ile Hız Kutusu Disli Tasarımı, Doktora Tezi Gazi Üniversitesi, Fen Bilimleri Enstitüsü, (**2005**).

[37] Kalınlı A., Mühendislikte Zeki Programlama Teknikleri ve Uygulamaları Ders Notları, Erciyes Üniversitesi, Kayseri, (**2007**).

[38] Nowostawski M., Poli R., Parallel Genetic Algorithm Taxonomy, Knowledge-Based Intelligent Information Engineering Systems, Third International Conference, Dec 1999 pp. 88 – 92, (**1999**).

[39] Cantu-Paz E., Efficient And Accurate Parallel Genetic Algorithms, Kluwer Academic Publishers, (**2000**).

[40] Bethke A. D., Comparison Of Genetic Algorithms And Gradient-Based Optimizers On Parallel Processors: Efficiency Of Use Of Processing Capacity, Tech. Rep. No. 197, Ann Arbor, MI, University of Michigan, (**1976**).

[41] Grefenstette J. J., Parallel Adaptive Algorithms For Function Optimization,

Tech. Rep. No. CS-81-19, Vanderbilt University, Computer Science Depratment, Nashville, TD, (**1981**).

[42] Fogarty T.C., Huang R., Implementing The Genetic Algorithm On Transputer Based Parallel Processing Systems, In Schwefel, H-P., Manner, R. (Eds.), Parallel Problem Solving from Nature, Springer Verlag, 145-149, (**1991**).

[43] Grefenstette, J. J., Robot Learning With Parallel Genetic Algorithms On Networked Computers, Proceedings of the 1995 Summer Computer Simulation Conference, 352-357, (**1995**).

[44] Bossert W., 1967, Mathematical Optimization: Are There Abstract Limits On Natural Selection?, Mathematical Challenges to the Neo-Darwinian Interpretation of Evolution, 35-46, (**1967**).

[45] Grosso P.B., Computer Simulations Of Genetic Adaptation: Parallel Sub-Component İnteraction İn A Multilocus Model, PhD thesis, The University of Michigan, (**1985**).

[46] Munetemo M., Takai Y., Sato Y., An Efficient Migration Scheme For Subpopulation-Based Asynchronously Parallel Genetic Algorithms, Proceedings of the Fifth International Conference on Genetic Algorithms, 649-649, (**1993**).

[47] Starkweather T., Whitley D., Mathias K., Optimization Using Distributed Genetic Algorithms, Parallel Problem Solving from Nature, Springer Verlag, 176-185, (**1991**).

[48] Rudolph G., Global Optimization By Means Of Distributed Evolution Strategies, In Schwefel, Parallel Problem Solving from Nature, Springer Verlag, 209-213, (**1991**).

[49] Whitley D., Rana S., Heckendorn, R. B., Exploiting Separability İn Search: The İsland Model Genetic Algorithm, Journal of Computing and Information Technology, 7(1), 33-47, (**1999**).

[50] Charles E. Leiserson Ilya B. Mirman, How To Survive The Multicore Software Revolution, Cilk Arts,Inc. , 1-3, (**2008**).

[51] Anonim, http://www.redhill.net.au/c/c-1.html, (Erişim Tarihi 25.02.**2013**).

[52] Ercan U., Akar H, Koçer A., Paralel Programlamada Kullanılan Temel Algoritmalar., Akademik Bilişim Konferansı, (**2009**).

[53] Yaşacan S., http://www.hermesiletisim.net/dev/paralel-programlama-nedir-2, (Erişim Tarihi 05.03.**2013**)

[54] MPI Programlamaya Giriş ve Motivasyon. http://www.uybhm.itu.edu.tr/documents/basarim09sunum/01_Giris_ve_Motivasyon_akinci_v3.pdf,(Erişim Tarihi 26.05.**2013**)

[55] Akcay M., Erdem H. A., Paralel Hesaplama ve Matlab Uygulamalari, Akademik Bilisim, Mugla Universitesi, 10-12, (**2010**).

[56] Akçay M., Şen B., Orak L.M., Çelik A., Paralel Hesaplama ve CUDA , 6. Uluslar arası İleri Teknolojiler Sempozyumu, (**2011**).

[57] Anonim, http://www.cs.cmu.edu/~fx/, (Erişim Tarihi, 05,05.**2013**).

[58] Aktaş V., Visual Studio ile Her Yönüyle C# 4.0, Kodlab Yayıncılık, 4.baskı, (**2011**).

[59] Sharp J., Adım Adım Visual C# 2010, Arkadaş Yayınları, (**2012**).

[60] Demirli N, İnan Y, Visual C# İle .Net 2008, Palme Yayıncılık, Ankara (**2011**).

[61] Algan S., C# 4.0, 16.Baskı, Pusula Yayıncılık, (**2010**).

[62] Anonim,http://www.intel.com/pressroom/archive/releases/2010/20100330comp_sm.htm, (Erişim tarihi 05.06.**2013**).

[63] Anonim,http://www.amd.com/us/products/desktop/processors/phenom-ii/Pages/phenom-ii-model-number-comparison.aspx, (Erişim tarihi 05.06.**2013**).

6. ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, adı : YILDIZ, Hüsrev

Uyruğu : T.C.

Doğum tarihi ve yeri : 10.03.1985, Düzce

Medeni hali : Bekar

Telefon : 0 544 916 55 56

E-mail : husrevyildiz@gmail.com

Eğitim

Derece	Eğitim Birimi	Mezuniyet tarihi
Lisans	Sakarya Ünv. Elektronik ve Bilgisayar Eğitimi Bölümü	2009
Önlisans	A.İ.B.Ü Düzce MYO/Endüstriyel Otomasyon	2004
Lise	Düzce Teknik Lisesi Elektronik Bölümü	2002

İş Deneyimi

Yıl	Yer	Görev
2009-2010	Düzce Merkez Ticaret Meslek Lisesi Bil.Tek. Bölümü	Öğretmenlik
2011	Düzce Üniversitesi Düzce MYO	Öğrtm. Elm
2012-2013	Özel Mantık Eğitim Kurumu	Bilgisayar Eğitmeni

Yabancı Dil

İngilizce (ÜDS/KPDS/TOEFL: ...-...)