PAMUKKALE ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ YIL : 2008
PAMUKKALE UNIVERSITY ENGINEERING FACULTY CİLT : 14
MÜHENDİSLİK BİLİMLERİ DERGİSİ SAYI : 2
JOURNAL OF ENGINEERING SCIENCES SAYFA : 175-183

KUTUP DENGELEME PROBLEMİ İÇİN YÜKSEK BAŞARIMLI BİR OPTİMİZASYON TEKNİĞİ

Bahadır KARASULU, Serkan BALLI, Serdar KORUKOĞLU, Aybars UĞUR
Ege Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, 35100, Bornova, İzmir

Geliş Tarihi : 25.07.2007
Kabul Tarihi : 14.04.2008

ÖZET

Karmaşık bilimsel problemlerin etkin olarak çözümlenmesinde yüksek başarımlı hesaplama teknikleri kullanılmaktadır. Kutup dengeleme problemi, yapay zeka alanları içerisinde önemli yer tutan robotbilim dalının en temel ölçüm araçlarından biridir. Bu çalışmada kutup dengeleme problemi, Yapay Sinir Ağı (YSA) ve yüksek başarımlı hesaplama tekniği kullanılarak çözülmüştür. Kutbu (çubuğu) dengede tutmayı sağlayan kuvvetin bulunmasında kullanılan destekleyici öğrenme yöntemini temel alan algoritma paralel ortama aktarılmıştır. Gerçekleştirimde C programlama dili ve paralel hesaplama tekniği için Mesaj Geçme Arayüzü kullanılmıştır. Bir YSA modeli olan Öz-örgütlemeli Harita Ağı'na ait yapay sinir hücre düğümleri ve ağırlıkları her biri dört çekirdekli altı adet (toplamda yirmi dört) işlemciye sahip bir sunucu bilgisayardaki işlemcilere dağıtılarak, farklı sinir hücre sayıları için performans değerleri elde edilmiştir. Yöntemin başarısı sonuçlar üzerinden tartışılmıştır.

Anahtar Kelimeler : *Yapay zeka, Kutup dengeleme problemi, Paralel hesaplama, Yapay sinir ağları, Öğrenme algoritmaları.*

A HIGH PERFORMANCE OPTIMIZATION TECHNIQUE FOR POLE BALANCING PROBLEM

ABSTRACT

High performance computing techniques can be used effectively for solution of the complex scientific problems. Pole balancing problem is a basic benchmark tool of robotic field, which is an important field of Artificial Intelligence research areas. In this study, a solution is developed for pole balancing problem using Artificial Neural Network (ANN) and high performance computation technique. Algorithm, that basis of the Reinforcement Learning method which is used to find the force of pole's balance, is transfered to parallel environment. In Implementation, C is preferred as programming language and Message Passing Interface (MPI) is used for parallel computation technique. Self–Organizing Map (SOM) ANN model's neurons (artificial neural nodes) and their weights are distributed to six processors of a server computer which equipped with each quad core processor (total 24 processors). In this way, performance values are obtained for different number of artificial neural nodes. Success of method based on results is discussed.

Key Words : *Artificial Intelligence, Pole balancing problem, Parallel computing, Artificial neural network, Learning algorithms.*

1. GİRİŞ

Geleneksel seri işlemcilerde, tek bir merkezi işlem birimi her işlemi sırasıyla gerçekleştirir. Yapay sinir ağları (YSA), geleneksel işlemcilerden farklı şekilde işlem yapmaktadırlar. YSA'lar her biri büyük bir problemin bir parçası ile ilgilenen, çok sayıda basit işlem birimlerinden oluşmaktadır. Böylece paralelleştirmeye olan yatkınlıkları ortaya

çıkmaktadır. Paralel hesaplama, aynı hesap görevinin (parçalara ayrılmış ve uyarlaması yapılmış), sonuçları seri hesaba göre daha hızlı elde etmek için çoklu işlemcilerde eş zamanlı olarak işletilmesidir. Paralel bilgisayarlar için popüler bir taksonomi ilk kez Flynn (1972) tarafından 1960'ların ortalarında tanımlanmıştır. Bu taksonomideki temel fikir, problemlerin çözüm hesabının ufak alt görev parçalarına bölünmesi ve bu parçaların eş zamanlı olarak düzenlenmesine dayanır (Grama v.d., 2003). Çok çeşitli paralel bilgisayar (işlemci) yapıları vardır. Bu çeşitler, işlemciler (işleme elemanı olarak adlandırılırlar) arasındaki veya işlemci ve hafıza arasındaki bağlantıya göre belirlenir. Genellikle, tüm işlemcilerin aynı zamanda aynı komutları işlemelerine göre (tek komut/çoklu veri-SIMD) veya her bir işlemcinin farklı komutları (çoklu komut/çoklu veri-MIMD) işlemesine göre paralel bilgisayarlar sınıflandırılmaktadır. Paralel işlemci makineleri simetrik (tüm işlemcilerin aynı seviyede olması) ve asimetrik (işlemcilerin bazı görevler için ayrılması ve önceliklerinin olması) çoklu işlemciler olarak ikiye ayrılır.

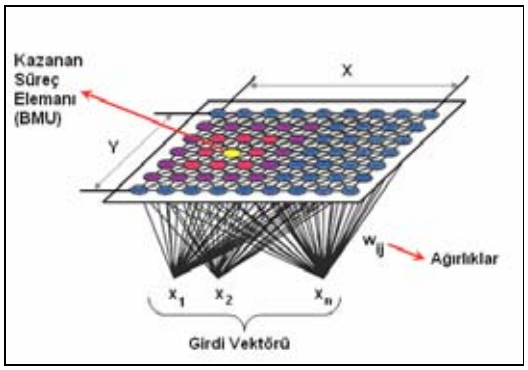
Uygulamalarda önemli bir yer tutan Çok Katmanlı Algılayıcı (Multi Layer Perceptron - MLP) sinir ağları, birçok sezim ve kestirim işlemlerini yürütmekte kullanılan ve parametrik olmayan bir yapay sinir ağı modelidir. Bu konu üzerine Grounds ve Kudenko (2006) tarafından belirtilmiş yapılan çalışmalara bakacak olursak; Destekleyici Öğrenme (Reinforcement Learning veya DÖ) yönteminin kullanımında paralelleştirmenin problemin çözümüne getirdiği iki önemli katkı vardır. Birincisi, tek-etmenli öğrenme problemlerine daha çabuk ve iyi bir çözüm bulabilmektir. İkincisi ise etmenlerin aynı problem üzerinde bilgi değiş-tokuşu yoluyla bir takım gibi ortaklaşa çalıştığı çoklu-etmen öğrenmesini mümkün kılan uygulamalar geliştirilmesinin sağlanmasıdır. Çalışmamızda etmen yaklaşımının ikinci önemli katkısında bahsedilen yapı kullanılmıştır. Ayrıca Gomez ve Miikkulainen (1998) tarafından yapılan çalışmada, kutup dengeleme problemleri (tek-kutuplu ve çok-kutuplu modeller, teleskopik kutup modelleri) ele alınmıştır. Kazanan herşeyi alır yaklaşımı (winner-takes-all approach) uyarınca, en kuvvetli aktivasyon ile ağın çıktısını (örneğin sigmoidal çıktılar için çıktının sıfıra veya bire yakın olması) kullanma yaklaşımı denenmiştir (Pardoe v.d., 2005).

Çalışmamızda, Kutup Dengeleme Problemi, kazanan herşeyi alır yaklaşımı temelli bir YSA ve yüksek başarımlı hesaplama tekniği kullanılarak geliştirilen bir uygulamayla çözümlenmiştir. Destekleyici Öğrenme yapıldığından ve Öz-örgütlemeli Harita Ağı'nın (Self-Organizing Map veya SOM) kendi yapısı gereği getirdiği bu yarışmacı yöntem iyi bir çözüm sağlamaktadır. Yazılım-tabanlı olarak Öz-örgütlemeli Harita biçimindeki yapay sinir ağı modelleri için basit bir asimetrik paralelleştirme yöntemi geliştirilerek, var olan seri biçimde tasarlanmış SOM algoritmalarını /programlarını hızlandıracak paralel uyarlama gerçekleştirilmeye çalışılmıştır. Ele alınan kutup dengeleme probleminde seri olarak kalan kısmın haricinde paralelleştirilebilecek olan kısım ağın eğitimi ve komşuluk hesaplamaları göz önüne alınarak (öz olarak ağırlıkların ve yerleşimlerin ilgili işlemciler/süreçler arasında transferleri ve bilgi paylaşımı) optimize edilmiştir. Böylece elde edilen yeni algoritma sayesinde tek işlemcili seri algoritma/programın başarım oranı arttırılmıştır. Önceki çalışmamızda (Karasulu ve Uğur, 2007) birbirine özdeş makineler (her biri 256 MB RAM 'e sahip 4 adet Intel Celeron mimarili 1.2 GHz işlemcili bilgisayar) kullanılarak test yapılmış ve en fazla 4 işlemcili bir sistemle (birbirinden bağımsız 4 bilgisayar üzerindeki bilgisayar ağı üzerinden bağlantılandırılmış işlemciler) sonuçlar alınabilmiştir. Bu çalışmamızda kullandığımız sistem, 2 GB paylaşımlı bellekli Sun© Microsystems, Inc. 'e ait UltraSparc T1 isimli altı adet dört çekirdekli işlemci'den (her biri 2.0 GHz) oluşmaktadır. Böylece toplamda 24 adet işlemciye kadar çıkılarak yeni sonuçlar elde etmemiz mümkün olmuştur. Ayrıca, yeni sistemimizin öncekinden diğer bir farklılığı da tüm işlemcilerin aynı bilgisayar üzerinde olmasıdır.

2. ÖZ-ÖRGÜTLEMELİ HARİTA AĞI

Kohonen Öz-örgütlemeli Harita Ağı (SOM) topoloji-korumalı bir haritadır. Bu harita yüksek boyutlu (üç veya daha fazla) bir haritadaki verileri, tipik bir iki boyutlu ızgara biçimindeki haritaya dönüştürür. Kohonen (1997) tarafından yapılan çalışmada, Öz-örgütlemeli haritanın ana amacının "Girdi uzayındaki komşuluk ilişkilerinin mümkün olduğunca korunması ve birimler arasındaki komşuluk ilişkilerine göre topoloji-korumalı bir haritanın yaratılması" olduğu ifadesine benzer bir biçimde açıklanmıştır. Böyle bir Öz-örgütlemeli haritanın eğitiminde başlıca zaman tüketen adımlar verilen bir örnek için kazanan süreç elemanının/düğümün (winner node) yerleştirilmesi ile ilgili alt-problem boyunca geçen adımlardır (Kohonen, 1996). Bir kazanan düğüm, her girdi vektörü için en iyi uyumlu birim (Best Matching Unit veya BMU) şeklinde ifade edilir. Bir örnekteki en yakın komşuyu bulma problemi için kullanışlı çok sayıda yöntem vardır (Vishwanathan ve Murty, 2000). En geçerli ve baskın karşılaştırma ise, şablon

vektörlerin durağan kalacağı varsayımı ile yapılmaktadır. SOM Ağı'nın bu durumunda tüm düğümlerin ağırlıkları sabit aralıklarla güncellenmektedir. Aşağıdaki Şekil 1'de BMU'nun belirtildiği bir SOM Ağı görülmektedir. Şekilden görülebileceği gibi SOM Ağı için iki boyutlu bir ızgara, buradaki girdiler ve onlara karşılık gelen ağırlıklar gösterilerek, BMU'nun "hangi düğüm" olduğunun tespit edildiği aşama gösterilmektedir.



Şekil 1. Kohonen Öz-örgütlemeli harita ağı.

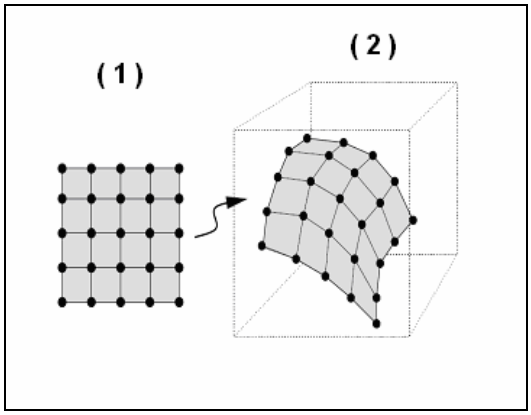
Örnek bir Öz-örgütlemeli Harita Ağı'nın formal bir biçimde tanımını yapacak olursak; böyle bir örnekte kullanılacak olan girdi vektörü, $X = [x_1, x_2, ..., x_n]^T \in R^n$ olsun. Bir *i* indisi ile düzenlenmiş birimlerin ayrık bir ızgarasını göz önüne alalım. Her düğüm ilgili ağırlık vektörü $W_i = [w_1, w_2, ..., w_n]^T \in R^n$ içermektedir. X burada, tüm ağırlık vektörleri içerisinde ağırlık vektörü onun en yakın komşusu olan birimi göstermektedir (Vishwanathan ve Murty, 2000). Buna kazanan süreç elemanı veya çoğu zaman literatürde BMU denilmektedir ve aşağıdaki Eşitlik 1 ile bulunmaktadır:

$$\|X - W_j\| = \min_i \|X - W_i\| \qquad (1)$$

Öz-örgütlemeli Ağ'ın eğitimi için her iterasyon aşağıda özetlendiği şekilde gerçekleşmektedir:

- Haritadaki düğümler arasından en yakın komşu (kazanan) her bir girdi örneği için bulunur.
- Kazananın ve tüm komşularının ağırlıkları güncellenir.

En çok zaman harcanan kısım bu komşulukları bulurken geçen süredir. Komşuluk hesapları öklid mesafesi (uzaydaki iki nokta arasındaki mesafe) uyarınca hesaplanılır. Öz-örgütlemeli harita temelde iki katmana sahiptir (Rauber v.d., 2000). Giriş katmanı tamamıyla çift boyutlu Kohonen katmanına bağlanmıştır. Çıkış katmanı ise nicemleme probleminde kullanılır ve giriş vektörünün ait olabileceği üç sınıfı temsil eder. Bu çıkış katmanı tipik olarak delta kuralını uygulayarak öğrenir. Kohonen katmanı işlem elemanlarının her biri, gelen giriş değerlerinden onların ağırlıklarının öklid mesafesini ölçmektedir. Birimin ağırlık vektörü ile girdi vektörü arasındaki öklid mesafesi bu durumda aktivasyon fonksiyonu görevi görmektedir. SOM için öklid mesafesi, öğrenme oranı'nın değişimini sağlamaktadır. Şekil 2'de görüleceği gibi fiziksel uzayda iki boyutlu bir ızgara yapısı sergileyen SOM, Ağırlık/Girdi uzayında eğimli bir yapıyı da sergileyebilmektedir.



Şekil 2. (1) Fiziksel uzayda ve (2) Ağırlık/Girdi uzayında Kohonen haritası.

Çalışmamızda elde ettiğimiz sonuçlarda da bu durumla karşılaşılmaktadır. Buna uygun olarak algoritma tasarımında ağırlık ve girdi ilişkisine dikkat edilmiştir. SOM Ağı'nda DÖ yanı sıra sıklıkla Yarışmacı Öğrenme (Competitive Learning) kullanılmaktadır. İki tip yarışmacı öğrenme yaklaşımı vardır. Bunlar hard ve soft olarak adlandırılır. Bu kavramları açıklamak adına herhangi bir SOM Ağı için, P(ξ) sinyal fonksiyonu olarak alındığını kabul edecek olursak, hard yarışmacı öğrenmeye (winner-takes-all) ait çevrimiçi-güncellemeli (online-update) yaklaşımın algoritması aşağıdaki şekilde olmaktadır;

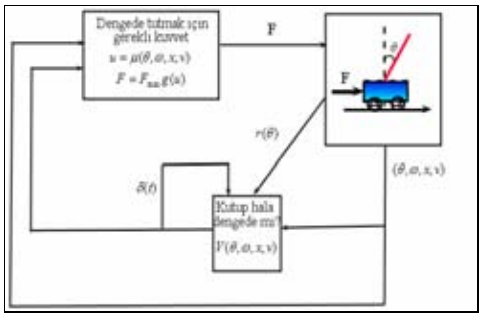
1. $P(\xi)$'ye göre rasgele seçilmiş $w_{c_i} \in \Re^n$ olan referans vektörüne sahip A kümesine N tane c_i birimini içerecek şekilde ilk değer atamasını yap, (Burada $A = \{c_1 + c_2, ..., c_N\}$ olarak verilmiştir),
2. $P(\xi)$'ye göre bir ξ giriş sinyalini rasgele olarak yarat,
3. $s(\xi) = \arg\min_{c \in A} \|\xi - w_c\|$ olacak şekilde kazananı $s = s(\xi)$ biçiminde tanımla,

4. Kazananın referans vektörünün ξ ile bağıntısını $\Delta w_s = \alpha(\xi - w_s)$ olacak biçimde uyarla,
5. Maksimum sayıda adıma (eğitim) ulaşılıncaya kadar algoritmanın 2. adımı ile 5. adımı arasında tekrar devam et.

Yukarıdaki algoritmada, α değeri öğrenme oranı olarak verilmektedir. Çalışmamızda, SOM Ağı'nın yarışmacı öğrenmeye ait çevrimiçi-güncellemeli yaklaşımındaki mantığa benzer bir algoritmik yaklaşım uygulanmıştır.

3. KUTUP DENGELEME PROBLEMİ

Kutup dengeleme veya ters çevrilmiş sarkaç problemi uzun yıllardan beri yapay zeka ve yapay sinir ağları ile ilgilenen araştırmacıların ortak bir ölçüm aracı olmuştur (Stanley ve Miikkulainen, 2002). Öz-örgütlemeli harita, öz olarak girdi ve çıktı uzaylarının arasında topoloji-korumalı formda haritalama yeteneği bulunan bir yarışmacı ağ'dır. Çalışmamızdaki yapay sinir ağı bir kutbu (burada örneğin 1 metre boyundaki bir çubuk gibi düşünebiliriz) temeline kuvvet uygulayarak dengede tutmayı öğrenmektedir. Kutbun davranışının Euler hareket yöntemine göre diferansiyel denklemlerin nümerik integrasyonu ile benzetimi yapılmıştır. Ağın görevi, kutbun durum değişkenleri ile kutbu dengede tutacak optimal kuvvet arasında bir haritalama yapılmasını sağlamaktır. Bu olay DÖ yaklaşımı ile gerçekleştirilir (Sutton ve Barto, 1998). Kutbun herhangi bir verilen durumunda, denek ağ haritalanmış kuvvetin zayıf bir değişimini dener. Eğer yeni kuvvet daha iyi bir kontrol sonucu verirse, haritada değişiklik yapılır, böylece sistem kutbun geçerli durum değişkenlerini ve yeni kuvveti bir eğitim vektörü olarak kullanır. Şekil 3'te görüleceği gibi kutup dengeleme açısı ve kutup puanı denilen kavram sayesinde karşılaştırma yapılabilmektedir (Sutton, 1992).



Şekil 3. Kutbu dengede tutmaya çalışan yapay sinir ağının çalışma şeması.

Yukarıda gösterilen Şekil 3'teki gibi bir sistem iki boyutlu bir düzlem üzerinde hareket edebilen bir plakaya bağlanmış bir çubuk (kutup) için dengeleme şartları ve Euler hareket denklemleri uyarınca hareket ettirilmektedir. Bu sistemdeki θ kutbun denge açısı, y ekseni ile kutbun yaptığı açıdır. YSA burada DÖ sırasında verilen değerler ve elde ettiği değerler ile öğrenmeyi gerçekleştirmektedir. Sisteme ait parametrelere $(\theta, \omega, \mathrm{x}, \nu)$ algoritmanın/programın başlangıcında ilk değer olarak sıfır atanmaktadır. Bunlar sırasıyla θ kutbun denge açısı (rad), ω açısal hız (rad/sn), x konum (m), ν çizgisel hız (m/sn) şeklindedir. Daha sonra rasgele olarak yaratılan ω değeri ve diğer değerler ayarlanmaktadır. Sistem çalıştırılarak öğrenmeye bırakıldıktan sonra Kutup Benzetimi sırasındaki adımlar hem gerçek çalışma süresi tutularak hem de benzetim adımı için verilen zaman adımları olan 0.1 saniye aralıklarla ölçülmüştür. Kutup Benzetimi sırasında ilgili birimin girdi vektörüne olan benzerliği bir puanlama sistemiyle kontrol edilmektedir. Buna bu çalışmada kutup puanı adı verilmektedir. Kutup dengeleme probleminde kullanılan bazı formüller ve parametreler hakkında bilgiler aşağıda yer almaktadır. Bu formül ve parametreler ilgili seri ve paralel program kaynak kodlarında kullanılmıştır. Tek bir plaka üzerinde tek bir kutbun dengelenmesi sırasındaki Euler hareket denklemleri şöyledir:

$$\ddot{x} = \frac{\left(F - M_p L\left(\dot{\omega}^2 \sin\omega - \ddot{\omega}\cos\omega\right)\right)}{M_c + M_p} \tag{2}$$

$$\ddot{\omega} = \frac{\left(G\sin\omega + \cos\omega\left(\frac{\left(-F - M_p L\left(\dot{\omega}^2 \sin\omega - \ddot{\omega}\cos\omega\right)\right)}{M_c + M_p}\right)\right)}{L\left(\frac{4}{3} - \frac{\left(M_p \cos^2\omega\right)}{M_c + M_p}\right)} \tag{3}$$

Programda kullanılan parametreler Tablo 1'de gösterilmektedir.

Benzetim sırasında kullanılan parametreler ve kullanılan değerleri Tablo 2'de görülmektedir. Önceki çalışmamızın (Karasulu ve Uğur, 2007) aksine bu çalışmamızda dengeleme süresi 300 saniye olarak alınmış, bu sayede sistemi dengeye getirecek optimum kuvvetin bulunması işlemi daha küçük boyuttaki ağ örüntülerinde de daha etkin hale getirilmiştir. Keza daha düşük dengeleme süresi ile benzetim yapıldığı durumlarda süre bitiminde sistemin hala dengeye gelemediği benzetim sonuçları da gözlemlenmiştir. Bu açıdan bakıldığında benzetimin süresi artırılmadan ve kutup denge açısı değiştirilmeden sadece dengeleme

süresinin artırılması küçük boyutlu ağlarda (25*25 düğümlük SOM ağı gibi) belirli bir performans artışı ve öğrenme süresinde düşüş yaşanmasını beraberinde getirmiştir.

Tablo 1. SOM ağı kullanan kutup dengeleme programında kullanılan parametreler.

Parametre ismi	Açıklama	Örnek
L	Kutbun (çubuğun) uzunluğu	1.0 metre
G	Yer çekimi ivmesi	9.81 m/sn^2
M_c	Plakanın (cart) kütlesi	2.0 kg
M_p	Kutbun (pole) kütlesi	1.0 kg
F	Uygulanan kuvvet	10 Newton
$\ddot{x}$	İvme	m/sn^2
$\ddot{\omega}$	Açısal ivme	rad/sn^2

Tablo 2. Benzetim sırasında kullanılan parametreler.

Açıklama	Ağ 25*25	Ağ 125* 125	Ağ 250* 250	Ağ 500* 500
SOM Satır Sayısı	25	125	250	500
SOM SütunSayısı	25	125	250	500
Eğitim adımları	100000	100000	100000	100000
Dengeleme süresi (sn)	300	300	300	300
Benzetim adım sayısı (iterasyon)	1000	1000	1000	1000
Kutbun denge açısı (y ekseni ile)	75	75	75	75
Kutup çubuğu uzunluğu (metre)	3.0	3.0	3.0	3.0
Plakanın ağırlığı (kg)	2.0	2.0	2.0	2.0
Kutbun ağırlığı (kg)	1.0	1.0	1.0	1.0
Yerçekimi (m/sn^2)	9.81	9.81	9.81	9.81
Benzetim zaman adımları	0.1	0.1	0.1	0.1

4. KULLANILAN PARALEL HESAPLAMA TEKNİĞİ VE ORTAM

Öz-örgütlemeli harita'nın seri algoritmasında, eğitim adımlar ile ilerler. Ağırlık vektörünün dizisi harita olarak alınır ve bu dizi iki kez dolaşılır. Bellek bantgenişliği sorununun öz-örgütlemeli haritanın yazılım-tabanlı uygulamalarında en büyük darboğazlardan birisi olduğu dikkate alınmalıdır. Seri algoritmada eğitimin her iterasyonunda harita üzerindeki her birime iki kez uğranılmaktadır. İlk etapta en iyi uyumlu birimi (Best Matching Unit - BMU) bulmak için dolaşılıp sonra da değişiklik için tekrar dolaşılmaktadır. Bu gerçekten verimi düşüren bir olgudur. Bu yüzden dağıtık ve dinamik bellekli ve bir çok işlemciye (sürece) destek verebilecek bir algoritmanın geliştirilmesi, verimi ve başarımı arttıracaktır. Algoritmanın geliştirilmesinden sonra, böyle bir sistemin programlanmasında mesaj geçme arayüzü (Message Passing Interface - MPI) ile yapısal ANSI C dilinin (GNU C) olanaklarının kullanılması tercih edilmiştir. Program geliştirilmesinde MPI paralel kütüphanesi olan LAM-MPI (Yerel Alan Çoklubilgisayarları – Mesaj Geçme Arayüzü) (LAM-MPI, 2007) kullanımı tercih edilerek ilgili SOM uygulaması gerçekleştirilmiş ve kutup dengeleme probleminin eğitim süreleri üzerinden başarım elde edilmiştir. LAM-MPI, MPI Standardı'nın açık kaynak uygulamasıdır. Tamamen bedava olarak dağıtılır. MPI standardı paralel uygulamalar için endüstride *de facto* standart haline gelmiştir. LAM, kütüphane çağrıları API'si olarak paralel bir uygulamaya katılan düğüm makineler arasında mesaj-geçmeyi sağlar. LAM, MPI-1 ve MPI-2 standartları elemanlarının tamamını kapsayan bir destekle gelir. Fakat bunların haricinde birkaç hata ayıklama ve izleme yazılımını da içermektedir (LAM-MPI, 2007). Seri olarak geliştirdiğimiz algoritmik yaklaşıma ek olarak bu çalışmada paralel yaklaşımla POSIX kanalları mantığına uygun bir yaklaşım ele alınmıştır. Verinin son aşamada tek noktaya toparlanarak indirgenmesinde birkaç yeni iyileştirme katkısı gerçekleştirilmiştir. Ayrıca yapılan çeşitli deneyler ve elde edilen sonuçlar bu çalışmada ayrıntıları ile ortaya konulmuştur.

5. BAŞARIM OPTİMİZASYONU İÇİN GELİŞTİRİLEN PARALEL HESAPLAMA YÖNTEMİ

Paralel programlama için birçok programlama dili ve kütüphaneleri vardır. Geçerli aktiviteleri ve programların çeşitliliğini, bu dil ve kütüphanelerin programcıya sağladığı adres uzayına bakıştaki farklılıklar sağlamaktadır. Mesaj-Geçme programlama paradigması, en eski ve en geniş paralel bilgisayar programlama yaklaşımıdır. Paralel hesaplamada verinin çeşitine göre bölümleme ve haritalama'da yapılmaktadır. Bölümleme, işlemcilere (süreçlere) dağıtılmak üzere veri yapılarını kişisel modüllere bölerek oluşturulan bir yol iken; haritalama, bu modüllerin ilgili işlemcilere (süreçlere) ilgili modül gelecek şekilde atanmasıdır (Grama v.d., 2003). Bu çalışmada kullanılan paralel hesaplama tekniği, ilgili verinin birçok süreç makinesi (hesaplama kümesine dahil edilmiş işlemciler/süreçler) arasında değiş-tokuş yöntemi ile paylaşılması ilkesine dayanmaktadır. Bu mantıkla bakılacak olursa, eğitim öncesi ve eğitim sonrası ve

BMU'nun aranması sırasında ilgili verilerin bir işlemciden diğer bir işlemciye gönderilip-alınması önündeki darboğazların aşılması gerekmektedir (LAM-MPI, 2007). Yazılımın geliştirilmesinden sonra yazılımın yeterince güçlü bir sunucu bilgisayarda denenmesi aşamasına gelinmiştir. Bu aşamada 2 GB paylaşımlı bellekli Sun© Microsystems, Inc. 'e ait UltraSparc T1 isimli altı adet dört çekirdekli işlemci'den oluşan (her biri 2.0 GHz) sunucu bilgisayarı (toplamda efektif olarak 24 adet işlemci ile çalışabilmektedir) seçilmiştir. Yazılımın bu sunucu bilgisayar üzerinde önce seri olarak, sonrasında çeşitli işlemci sayıları baz alınarak çalıştırılması sağlanmış ve böylece deney sonuçları elde edilmiştir. Sırasıyla 1, 8, 16, 24 işlemci sayısı denenmiş ve başarım optimizasyonu dikkate alınarak yazılımda uygun değişiklikler yapılmıştır. Optimizasyon sonucunda paralel hesaplama için geliştirilen en güncel yazılım deneyde kullanılan dört farklı işlemci sayısı ile dört farklı benzetim koşması için farklı ağ büyüklüklerinde test edilmiştir. Deneylerde sistemin başarım oranının işlemci sayısı artırıldıkça arttığı fakat bir noktadan sonra işlemci artımının başarıyı etkilemediği (Amdahl Kanunu'nun doğal bir sonucu olarak) görülmüştür. Tepe başarımına erişildiği noktada en verimli öğrenmeninde geliştirilen algoritma tarafından gerçekleştirildiği gözlemlenmiştir.

Deneyler sırasında çeşitli büyüklüklerde (deneyde 25*25, 125*125, 250*250, 500*500 düğüm büyüklüğünde) YSA'lar denenerek, sistemin gösterdiği tepkiler ve algoritmanın güvenirliği (Kretchmar, 2002) 'deki gibi sınanmıştır. Çalışmamızdaki algoritmaya dayanan benzetim programı gerçekleştirilmiştir. Programın hem seri hem de paralel sürümleri bulunmaktadır. Geliştirilen öğrenme ve paralel hesap yöntemine ait algoritmik akış aşağıdaki gibidir:

1. Başla,
2. İlgili dizi ve değişkenleri tanımla,
3. Gerekli dizi elemanları ve/veya değişkenlerin değerlerini sıfırla,
4. MPI'ı başlat,
5. İşlemci sayısı ve kimliklerini tespit et,
6. Euler Hareket denklemlerinde kullanılacak ilgili parametreleri dosyalardan oku,
7. Hesaplamada geçen sürenin tespiti için saat tutmaya başla,
8. Yapı bloğu kullanarak yeni bir Kohonen SOM ağı oluştur,
9. İlgili prosedürler ile rasgele sayı üreteci ile rasgele ağırlıkları oluştur,
10. Ağın eğitimini başlat,
11. Ağın eğitimi sırasında ilgili girdi ve çıktı verilerini ve ağırlıkları işlemciler üzerine dağıt,
12. Ağın eğitimi boyunca ilgili hesaplamaların işlemcilere başlangıç bulma formülü ile (eşit miktarda) dağıtılmasını (Formül = [Toplam YSA birimi (sinir hücresi düğüm sayısı) / toplam işlemci sayısı * (geçerli işlemcinin sırası +1)]) yoluyla yaptır,
13. Elde edilen yerel sonuçları işlemci sırası (rank) sıfır olan (yani fiziksel makine üzerindeki numarası sıfır olan işlemci veya yönetici işlemci olarak adlandırılan süreç düğümü) üzerinde toparla,
14. Adım 11 ile 13 arasındaki algoritma adımlarını eğitim adımları bitene kadar tekrarla,
15. Her bir süreç makinesinin veriyi doğru olarak aldığını/gönderdiğini kontrol et,
16. Her adımda elde edilmiş olan Benzetim zaman adımı değeri (0.1 saniyelik adımlar), o adımda kutup çubuğunun y ekseni ile arasındaki açı (kutbun hala dengeli olup olmadığının tespiti için) ve uygulanan optimum kuvveti içeren 3 kolonluk bilgi ilgili dosyaya yazdır (17. adımdaki detaya göre gerçekleşir),
17. Dosyaya yazdırma işleminde, sabit diske o an aktif olan işlemci erişsin ve ilgili veriyi yazdır (diğer işlemciler de aynı anda bu veriyi okuyabilsin), böylece paralel olarak dosyaya erişimi [yazma/okuma] gerçekleştir,
18. Saat tutmayı bitir,
19. MPI'ı bitir,
20. Dur.

6. ELDE EDİLEN DENEY SONUÇLARI

Bu çalışma boyunca yapılan deneyler sırasında elde edilen hızlanma oranları (başarım) ve verim yüzdelikleri aşağıdaki tablo ve grafiklerde verilmektedir. Amdahl Kanunu olarak anılan genel bir model paralel işlemdeki hızlanma oranı için Amdahl (1967) tarafından ortaya konulmuştur (El-Rewini ve Abd-El-Barr, 2005). Hızlanma deyimi (S_p) için (en basit biçimde) Amdahl Kanunu formülü,

$$S_p(n_p) = \frac{T(1)}{T(n_p)} \tag{4}$$

şeklindedir. Burada n_p , işlemci sayısıdır, *T(1)*, tek işlemci çalıştırıldığında geçen süre, $T(n_p)$ ise n_p adet işlemci çalıştırıldığında geçen süredir. Verim için (en basit biçimde) kullanılan formül ise,

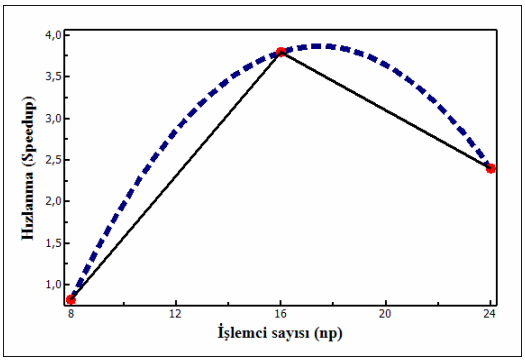
$$\varepsilon(n_p) = \frac{T(1)}{n_p T(n_p)} \tag{5}$$

şeklindedir. Tek ve Çok işlemcili benzetim programı için seri hesap (yani n_p=1) iken ve paralel hesapta (n_p=8 veya n_p=16 veya n_p=24) iken geçen süreler, bu sürelere ait hızlanma değerleri ve bu hızlanma değerlerine ait verim değerlerini Tablo 3'de gösterilmektedir.

Tablo 3. Tek ve çok-işlemcili sistemde geçen süre değerleri, hızlanma ve verim tablosu.

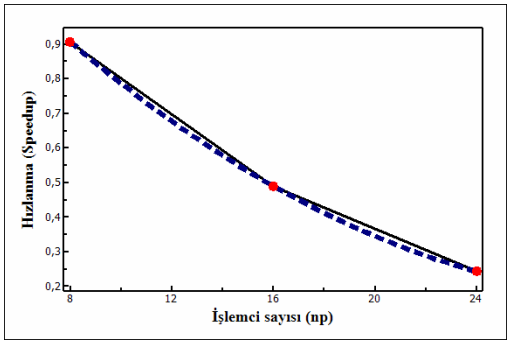
Açıklama:	İşlemci Sayısı (n_p)	Ağ (25*25)	Ağ (125*125)	Ağ (250*250)	Ağ (500*500)
Geçen Süre (sn)	1	2626	3583	12827	51392
	8	3171	3942	13827	55263
	16	691	7283	22980	84248
	24	1094	14656	42390	93806
Hızlanma Oranı	1	1	1	1	1
	8	0.828	0.908	0.927	0.929
	16	3.8	0.491	0.558	0.610
	24	2.4	0.244	0.302	0.547
Verimlilik (%) olarak	1	100	100	100	100
	8	10	11	11	11
	16	23	3	3	3
	24	10	1	1	2

Ele aldığımız sistem için 25*25 (yapay sinir hücresi) düğümlük ağ büyüklüğüne sahip Kohonen SOM Ağı'na ait hızlanma grafiği Şekil 4 'de verilmektedir.



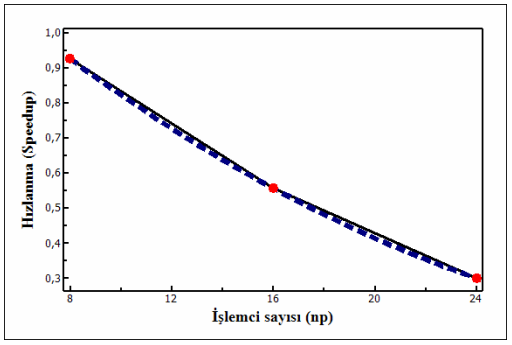
Şekil 4. 25*25'lik sinir ağı'na ait hızlanma oranı grafiği.

Şekil 4'deki düz çizgi ile gösterilen eğri gerçek hızlanma değerlerini, kesikli çizgilerle gösterilen eğri ise 'eğri uydurma' ile elde edilen hızlanma grafiğini gösterir. Şekil 4'de 24 işlemci ile çalıştırılan paralel programın, hızlanma grafiğinde tepe performansına (S_p=3.8) aynı anda çalışan 16 işlemcinin bulunduğu bir benzetim koşmasında ulaştığı görülmektedir. Bu çalışmanın en yüksek veriminin 25*25 düğümlük (SOM yapay sinir hücresi) ağ ile elde edildiği görülmüştür. Bu boyutlardaki ağın diğer 125*125, 250*250 veya 500*500 düğümlük ağlardan verimli olmasının nedeni daha az karmaşık bir düğümler-arası yapıya sahip olmasından kaynaklanmaktadır. Şekil 5'te 125*125'lik ağın işlemci sayısı arttıkça hızlanma oranındaki düşüş gözlenmektedir.

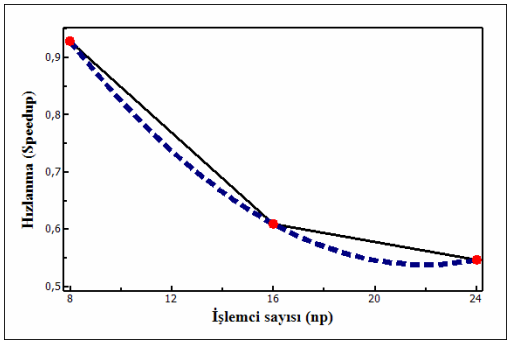


Şekil 5. 125*125'lik sinir ağı'na ait hızlanma oranı grafiği.

Şekil 5, 6 ve 7'den anlaşılacağı üzere bu sonuç, sistemin 16 işlemci ile optimum 25*25 düğümlük (sinir hücresi) ağda en yüksek hızlanmaya sahip olduğunu, fakat diğer ağ büyüklüklerinde ise hızlanmada istenilen artışın olmadığını göstermektedir.



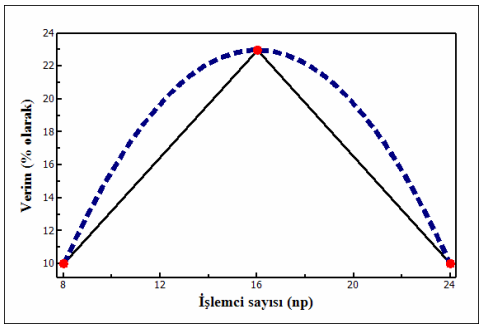
Şekil 6. 250*250'lik sinir ağı'na ait hızlanma oranı grafiği.



Şekil 7. 500*500'lük sinir ağı'na ait hızlanma oranı grafiği.

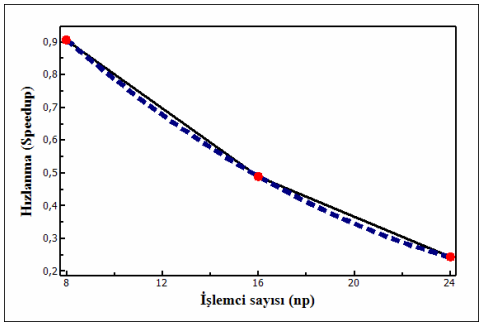
Şekil 8 'den görüleceği gibi, çalışmada 16 işlemcili benzetimdeki tepe başarım (verim) değeri % 23'dür. 125*125, 250*250, 500*500 düğümlük (sinir

hücresi) ağlar da ise 16 işlemci ve üstünde istenilen/uygun verim elde edilememiş, bu yüzden verim sadece %1 ile % 3 arasında kalmıştır.



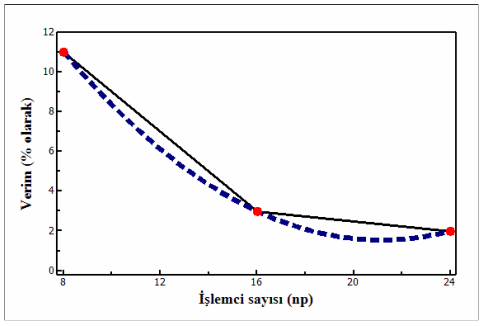
Şekil 8. 25*25'lik sinir ağı'na ait verim grafiği.

Aşağıda 125*125 ve 250*250 düğümlük (sinir hücresi) ağların verim değerleri aynı çıktığı için Şekil 9 'daki aynı grafik üzerinden gösterilmiştir. Verimin işlemci sayısı artırıldıkça düşmekte olduğu görülmektedir.



Şekil 9. 125*125'lik sinir ağı'na ve 250*250'lik sinir ağı'na ait verim grafiği.

Şekil 10'da 500*500 düğümlük (sinir hücresi) sinir ağ'ı için verimin çok düşük olduğu, işlemci sayısı artırılsa da verimin yükselmediği görülmektedir.



Şekil 10. 500*500'lük sinir ağı'na ait verim grafiği.

Elde edilen verimlerin hızlanma oranları ile tutarlı olduğu ve ağ büyüdükçe azaldığı, optimal ağ büyüklüklüğü ve optimum işlemci sayısında ancak en iyi verimin elde edildiği görülmektedir. Bu durum SOM Ağı'nın yapay sinir hücresi düğümü sayısına bağlı iletişim maliyeti artışının doğal bir sonucu olarak yorumlanabilir. Dengeleme süresinin artırılması, büyük ölçekli ağlarda hızlanmayı artırmak yerine düşürmüştür. Bunun ana nedeni SOM ağı sinir hücre düğümleri arasındaki yarışmacı öğrenmenin (BMU'ya olan uygunluğa yaklaşma çabası) sonucu sistemi dengeye getirici optimum kuvvetin tespitinde yaşanan öğrenme gecikmesiyle ilişkilendirilebilir.

5. SONUÇ VE TARTIŞMA

Bir ağ üzerinden birbirine bağlanan bilgisayarlar, LAN teknolojisi üzerinden bağlandığından çeşitli gecikmelerden dolayı aralarındaki iletişim maliyeti oldukça fazladır. Buna karşın, tek bir sunucu bilgisayar içerisindeki altı adet dört çekirdekli işlemcinin birbirleriyle iletişim maliyetleri ağ üzerinden bağlı olanlara göre oldukça düşüktür. Bu sayede çalışmamızdaki işlemciler arası veri aktarımında daha etkin bir performans elde edilebilmiştir. Geliştirilen paralel algoritmada seri algoritmada eksikliği gözlemlenen gerekli iyileştirmeler yapılmıştır. Bu iyileştirmelerin haricinde benzer bir sistemin başarım açısından daha güçlü bir sunucu sistemde (24 işlemcili) koşturulmasıyla üreteceği sonuçlar da gözlenmiştir. Uygulamada seri hesabın yanı sıra paralel hesap yapılması ile belirli bir hızlanma elde edilmiş ve sistemin farklı ağ büyüklükleri için orantılama yöntemiyle kıyaslaması/ölçümü yapılmıştır. Her ne kadar düğümler arası bağlantılar karmaşıklaştıkça (ağ büyüklüğü değiştikçe doğru orantılı olarak düğüm sayısının artıp/azalmasına bağlı) öğrenme hızlansa da, sonuca varma (optimum uygulanması gereken kuvvetin yönü ve büyüklüğünün tespiti) gecikmektedir. Verime göre sistemin uygulanan YSA katman sayısında (gizli katman) artış olmamasına rağmen düğüm sayısının artırılıp/azaltılmasının hızlanmayı doğrudan etkilediği görülmektedir.

Önceki çalışmamızda (Karasulu ve Uğur, 2007) %23 'lük verim 4 işlemcili sistemde, 500*500 nöronlu SOM sinir ağına uygulanan benzetim sonucunda elde edilmişken, yeni sistemimizde (16 işlemcili durumda) 25*25 nöronlu SOM ağında aynı başarıya ulaşılmıştır. Böylece daha az bellek gereksinimi ile aynı işin yaklaşık olarak aynı verimle yapılabileceği gösterilmiştir. Az sayıda nörondan oluşan ağlar için, yeni sistemimize benzer bir yapının daha başarılı olduğu sonucuna varılmıştır.

Tasarlanan sistem için öğrenmeyi gerçekleştirip bu yolla kutbu dengede tutabilecek optimum kuvvetin kestirimini sağlayan paralel algoritmanın/programın tutarlı ve düzgün sonuçlar elde ettiği gözlenmiştir. Fakat problemin doğası gereği seri kalan kısmın haricindeki kısım için paralelleştirme yüzdesi istenilen ölçüde olmamıştır. Tepe başarım değeri (en yüksek hızlanma değeri) 25*25 düğümlük ağ sistemi için 16 işlemcili durumda yapılan benzetim sonucunda elde edilmiştir. İleriki çalışmalar için SOM ağı yanında LVQ (Learning Vector Quantization veya Vektör Nicemlendirmeli Öğrenme) ağı gibi diğer YSA modellerinin probleme uygun parametre değerleri bulunarak test edilmesi ve daha verimli sonuçlara ulaşılması düşünülmektedir.

6. KAYNAKLAR

Amdahl, G.M. 1967. Validity of single-processor approach to achieving large-scale computing capability, proceedings of AFIPS conference, reston, VA. p. 483-485.

El-Rewini, H. and Abd-El-Barr, M. 2005. Advanced Computer Architecture and Parallel Processing, Wiley-Interscience, John Wiley and Sons Inc.

Flynn, M. 1972. Some computer organizations and their effectiveness, IEEE Trans. Comput., Vol. C-21, p. 948.

Gomez, F. and Miikkulainen, R. 1998. 2-D pole balancing with recurrent evolutionary networks. In Proceedings of the International Conference on Artificial Neural Networks (ICANN-98), Skovde, Sweden), 425–430.

Grama, A., Gupta A., Karypis G. and Kumar, V. 2003. Introduction to Parallel Computing, Second Edition, Addison Wesley Publishing, ISBN 0-201-64865-2, 856, 2003.

Grounds, M. and Kudenko, D. 2006. Parallel reinforcement learning by merging function approximations. Department of Computer Science University of York.

Karasulu, B. ve Uğur, A. 2007. Özörgütlemeli yapay sinir ağı modelinin kullanıldığı kutup dengeleme problemi için paralel hesaplama tekniği ile bir başarım eniyileştirme yöntemi, akademik bilişim 2007, Bildiri No: 25, Dumlupınar Üniversitesi, Kütahya, 31 Ocak - 2 Şubat 2007.

Kohonen, T. 1996. The Speedy Som, Technical Report a33, Helsinki University of Technology, Laboratory of Computer and Information Science, Espoo, Finland.

Kohonen, T. 1997. Self-Organizing Maps, Series in Information Sciences, Vol. 30, Second Edition Springer, Heidelberg, ISBN 3-540-62017-6, 1997.

Kretchmar, R. M. 2002. Parallel reinforcement learning. In proceedings of the 6th World Conference on Systemics, Cybernetics, and Informatics (SCI2002).

LAM-MPI Takımı websitesi. (Çevrimiçi: http://www.lam-mpi.org).

Stanley, O.K. and Miikkulainen, R. 2002. Efficient reinforcement learning through evolving neural network topologies. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002).

Pardoe, D., Ryoo, M. and Miikkulainen, R. 2005. Evolving neural network ensembles for control problems, in proceedings of the genetic and evolutionary computation conference (GECCO-2005).

Rauber, A., Tomsich, P. and Merkl, D. 2000. parSOM: A parallel implementation of the self-organizing map exploiting cacheeffects: making the som fit for interactive high performance data analysis, p. 6177, IEEE-INNS-ENNS Int. Joint Conference on Neural Networks (IJCNN'00), Vol 6.

Sutton, R. S. 1992. Reinforcement learning architectures. Proceedings ISKIT'92 International Symposium on Neural Information Processing, Fukuoka, Japan.

Sutton, R. S. and Barto, A. G. 1998. Reinforcement learning: An introduction., Cambridge, MA, MIT Press.

Vishwanathan, S. V. N. and Murty, M. N. 2000. Kohonen's SOM with cache. The Journal of Pattern Recognition Society 33, 1927-1929.