ISSN:1306-3111
e-Journal of New World Sciences Academy
2012, Volume: 7, Number: 4, Article Number: 1A0337

Devrim Akgün
İbrahim Şahin
İbrahim Yücedağ
Hacer Bayıroğlu
Duzce University, Duzce-Turkey
devrimakgun@duzce.edu.tr
ibrahimsahin@duzce.edu.tr
ibrahimyucedag@duzce.edu.tr
hacerbayiroglu@duzce.edu.tr

NWSA-ENGINEERING SCIENCES
Received: August 2012
Accepted: September 2012
Series : 1A
ISSN : 1308-7231

PARALEL MATRİS ÇARPMA ALGORTİMASININ ÇOK ÇEKİRDEKLİ BİLGİSAYAR ÜZERİNDE JAVA İŞ PARÇAÇIKLARI İLE BAŞARIM ANALİZİ

ÖZET

Çok çekirdekli işlemci teknolojisinin gelişmesiyle birlikte, paralel çalışan iş parçacıklarını kullanarak sıralı algoritmaların hızını artırmak oldukça pratik hale gelmiştir. Günümüzde yaygın olarak kullanılan Java dili, iş parçacıklarının yönetilmesi için kullanılan yerleşik kütüphaneleri ile paralel uygulamaların geliştirilmesine destek sağlar. Sunulan çalışmada, altı çekirdekli bilgisayar üzerinde, matrissel işlemlerle ilgili birçok uygulamada temel oluşturan matris çarpma algoritmasının Java dili kullanılarak paralel gerçeklenmesi ile elde edilen başarım incelenmiştir. Bu amaçla, iş parçacıklarının yönetimi için uygulamada yaygın olarak başvurulan statik ve dinamik yük dengeleme yaklaşımlarının paralel işlem gücünün kullanımı açısından etkisi gösterilmiştir. Yapılan deneysel ölçümlerle, iş parçacığı adedi ve matris boyutunun paralel hesaplama ile sağlanan başarıma etkisi gösterilmiştir. Java dili ile gerçeklenmiş paralel matris çarpma algoritmasının çok çekirdekli bilgisayar üzerinde önemli seviyede hızlandırma ve paralel verimlilik sağladığı grafiksel sonuçlar ile karşılaştırmalı olarak gösterilmiştir.

Anahtar Kelimeler: Paralel Hesaplama, Çok Çekirdekli İşlemci, Java, İş Parçacığı, Matris Çarpım

PERFORMANCE ANALYSIS OF PARALLEL MATRIX MULTIPLICATION ON A MULTI-CORE COMPUTER USING JAVA THREADS

ABSTRACT

With the developing multi-core technology, it becomes quite practical to speed up sequential algorithms via parallel running threads. Java language which is widely used today supports developing parallel applications via built in libraries for managing the threads. In the presented study, experimental performance of the parallel implementation of matrix multiplication algorithm that provides a basis for most of the matrix operations is investigated on a six-core computer using Java language. For this purpose, the impact of static and dynamic load management approaches which are common in practice for managing threads on the use of computational power are shown. Thorough the experiments, the effect of number of threads and the matrix size on the parallel computation performance are measured. It has been shown by comparative graphical results that parallel matrix multiplication algorithm which is realized by Java language bring significant speed up and parallel efficiency on multi-core computer.

Keywords: Parallel Computation, Multi-Core Processor, Java, Thread, Matrix Multiplication

1. GİRİŞ (INTRODUCTION)

Hızla gelişen çok çekirdekli işlemci teknolojisi ile birlikle algoritmaların çalışma zamanlarının düşürülmesi için kullanılan paralel programlama, günümüzde başvurulan önemli hesaplama yaklaşımlarındandır. Özellikle yüksek hesaplama gücü gerektiren uygulamalarda yüksek hızlı işlemci kullanmaya alternatif olarak, algoritmayı parçalara bölüp aynı anda farklı çekirdekler üzerinde çalıştırmak ve böylece uygulamaların hızını artırmak çeşitli alanlardaki araştırmacıların üzerinde çalıştığı etkili yöntemlerdendir [1, 2, 3 ve 4]. Paralel hesaplamada hız kazancı, hesaplama yükünün küçük parçalara bölünerek her bir parçanın aynı anda işletilmesiyle sağlanır. Bu işlem genelde algoritma yapısına bağlı olarak komut bazında, fonksiyon bazında ya da döngü bazında parçalara bölmek gibi çeşitli paralelleştirme yöntemleriyle gerçekleştirilebilir. Yaygın kullanılan yöntemlerden biri olan döngü bazında paralelleştirme ile birçok bilimsel hesaplama tekniğinin temelini oluşturan matris çarpma işlemi etkili bir şekilde paralel formda gerçeklenmiştir [5]. Paralel matris çarpma işlemi determinantların belirlenmesi, doğrusal denklem takımlarının çözümü, QR faktörizasyonu ve graf teorisi uygulamaları gibi işlemlerin paralel yürütülmesinde temel oluşturur [6 ve 7].

Prosesler içerisinde oluşturulabilen iş parçacıkları paralel algoritmaların gerçekleştirilmesinde oldukça pratik yaklaşım sağlar. Çok çekirdekli işlemciye sahip bir bilgisayarda işletim sistemi birden fazla programı çekirdeklere dağıtarak aynı anda yürütebilir. Bundan dolayı, birbirinden faklı görevler yürütmek için aynı proses içinde tanımlanabilen iş parçacıkları donanımın birden fazla işlemciye sahip olduğu durumlarda paralel hesaplama için kullanılmaktadır [8-10]. Farklı işletim sistemleri ile uyumlu çalışabilen ve günümüzde çok yaygın kullanılan Java platformu, iş parçacıklarının oluşturulması için yerleşik kütüphanelere sahiptir [11 ve 12]. Java birden fazla iş parçacığının aynı program içerisinde tanımlanmasına *java.lang.Thread* sınıfı ya da *java.lang.Runnable* ara yüzü aracılığı ile izin verir. Paralel derleyici kullanmadan birçok sıralı algoritma pratik bir şekilde Java programlama yapısıyla gerçekleştirilebilir. Paralel hesaplamalar için uygun ortam sağlayan Java ile iş parçacıklarını tanımlayan nesneleri oluşturmak, yönetmek ve bunlar arasında paylaşımlı veri tipleri kullanmak oldukça pratiktir. Bu sayede çarpımı gerçekleştirilecek matrisler iş parçacıkları arasında ortak kullanılarak her bir iş parçacığı ayrılan işlem yükünü matrisin ilgili elemanlarını kullanarak gerçekleştirebilir. Paralel matris çarpma işleminde kullanılan çekirdek adedi arttıkça elde edilecek hızlandırma miktarının da aynı oranda artması beklenir. Uygulamada iş parçacıklarının başlatılması ve sonlanmasının oluşturduğu ek yükler, senkronizasyon gerektiren işlemler ya da bellek iletişimleri gibi etkenler hızlandırma oranını bir miktar düşürür. Sunulan çalışmada, çok çekirdekli bilgisayar üzerinde Java yazılımı kullanılarak hızlandırma ve paralel verimlilik, iş parçacığı adedi ve matris boyutlarına bağlı olarak incelenmiştir.

2. ÇALIŞMANIN ÖNEMİ (RESEARCH SIGNIFICANCE)

Paralel matris çarpma işlemi birçok sayısal çözümleme, veri işleme gibi matrissel işlemlere dayanan algoritmaların gerçekleştirilmesinde kullanılan temel işlemlerden biridir. Döngü bazında paralelleştirme uygulanabilen matris çarpma işleminde elde edilen başarım işlemci sayısı ile doğru orantılıdır. Sunulan çalışmada, günümüz farklı işletim sistemleri ile uyumlu çalışan Java

platformu ile çok-çekirdekli bilgisayar üzerinde matris çarpım algoritması kullanılarak elde edilen paralel hesaplama başarımı incelenmiştir. Bu amaçla paralel döngülerin kontrolünde yaygın olarak başvurulan statik ve dinamik dengeleme algoritmaları kullanılmıştır [13, 14, 15 ve 16]. Deneylerde kullanılan altı çekirdekli bilgisayar ile elde edilen hızlandırma ve paralel verimlilik sonuçları tek çekirdekli uygulamadan elde edilen sonuçlar ile kıyaslanarak elde edilen başarım tablolar ve grafikler ile gösterilmiştir. Bu çalışma, yaygın olarak kullanılan Java platformunun günümüz çok çekirdekli bilgisayar teknolojisi ile birlikte sağladığı başarımın matrissel işlemelere temel oluşturan matris çarpma işlemi ile deneysel olarak ortaya konması bakımından önem taşımaktadır.

3. JAVA İŞ PARÇACIKLARI İLE PARALEL HESAPLAMA (PARALLEL COMPUTATION WITH JAVA THREADS)

Paralel algoritmalar, çok-çekirdekli donanımların sahip olduğu hesaplama potansiyelinin sıralı algoritmaların hızını artırmada kullanımı açısından önemli rol oynar. Prosesler içinde oluşturulabilen iş parçacıkları ile bir sıralı algoritmaya ait görevler parçalara bölünebilir ve aynı anda ayrı iş parçacıkları tarafından paralel işletilerek görevin daha kısa sürede tamamlanması sağlanabilir. Bir çekirdek üzerinde aynı anda yalnızca bir iş parçacığı yürütüldüğü için iş parçacıkları ile tanımlanan işlemler işletim sistemi tarafından önceliklerine bağlı olarak zaman paylaşımlı yürütülürler. Eğer donanım iki ya da daha fazla çekirdeğe sahipse veya hyperthreading desteği varsa aynı sayıda iş parçacığı aynı anda yürütülebilir. Bu özellik kullanılarak paralel gerçeklenebilen algoritmaların yürütülmesi için geçen zaman işlemci sayısına bağlı olarak oldukça düşürülebilir. Bir çekirdek üzerinde aynı anda yalnızca bir iş parçacığı yürütüleceği için, iş parçacığı adedi çekirdek adedi ile sınırlanmalıdır.

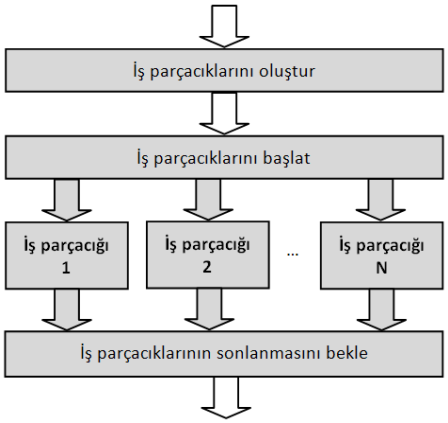
```
class IsParcasi extends Thread{
      //Sınıf ve nesne alanları
      run(){
            // Paralel işlemler
      }
}
```

```
class IsParcasi implements Runnable{
      //Sınıf ve nesne alanları
      run(){
            // Paralel işlemler
      }
}
```

Şekil 1. Standart matris çarpım algoritması
(Figure 1. Standard Matrix multiplication algorithm)

İş parçacıklarının oluşturulmasında kullanılacak sınıf, Thread sınıfı veya Runnable ara yüzü aracılığı ile Şekil 1'de görüldüğü gibi iki farklı yöntemle tanımlanabilir. Bunlardan birinde iş parçacığı sınıfı Thread sınıfını miras alarak tanımlanırken diğerinde Runnable ara yüzünü gerçekleyerek tanımlanmıştır. İş parçacıkları ile paralel hesaplama için *run()* ile tanımlanan yöntem içinde paralel işletilecek kodların yazılması gerekir. Sıralı olarak yürütülmekte olan bir programın içinde, paralel parçalar halinde işlenip tekrar sıralı hale dönülmesi genelde *fork-join* yapısı olarak adlandırılır [9]. Ana programda ise Şekil 2'de görüldüğü gibi Thread sınıfını miras alarak tanımlanan IsParcasi sınıfından nesneler üretilerek iş parçacıkları oluşturulur. Ardından üretilen her bir nesnenin *start()* yöntemi çağırılarak iş parçacıkları tanımlanan görevi gerçekleştirmesi için başlama komutu verilir. Görev tamamlanana kadar bariyer oluşturan *join()* yöntemi ile tüm iş parçacıklarının durumu kontrol edilir ve

hesaplamanın paralel kısmı sonlandırılmış olur. Bundan sonra varsa yapılacak diğer seri işlemler yürütülür. Eğer paralel işlenen veriler nesneye bağlı ise bunların birleştirilmesi işlemi gerçekleştirilebilir. Bu çalışmada, çarpılacak matrisler, sınıfa bağlı alanların tanımlamasında kullanılan *static* veri tipi ile paylaşımlı olarak tanımlanmıştır.




```
// iş parçacıklarını oluştur
IsParcasi[] o=IsParcasi[cekirdekAdedi];
for (k=0;k<cekirdekAdedi;k++){
      o[k]=new IsParcasi(parametreler);
}

//iş parçacıklarını başlat
for (k=0;k<cekirdekAdedi;k++){
      o[k].start();
}
//iş parçacıklarını sonlandır
for (k=0;k<cekirdekAdedi;k++){
      o[k].join();
}
```

Şekil 2. Fork-join (alt parçalara ayırma ve toplama) yaklaşımı
(Figure 2. Fork-join(splitting into subpieces and collecting) approach)

4. PARALEL MATRİS ÇARPIMI VE YÜK DENGELEME YAKLAŞIMLARI (PARALLEL MATRIX MULTIPLICATION AND LOAD BALANCING APPROACHES)

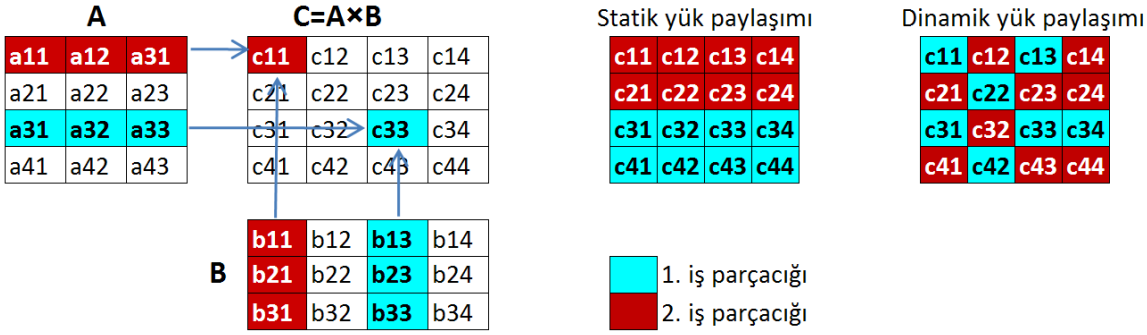
Paralel matris çarpımı birçok döngü bazında paralelleştirme (data parallel) uygulamasına temel oluşturması bakımından önemli bir matematiksel işlemdir. Matris çarpma işlemi, determinantların belirlenmesi, doğrusal denklem takımlarının çözümü ve QR faktörizasyonu gibi birçok matris işlemi ile benzer hesaplama karmaşıklığına sahiptir. $O(n^3)$ hesaplama karmaşıklığına sahip olan standart kare matris çarpma işlemi sıralı programlama ile örnek gerçekleştirilme kodları Şekil 3'te verilmiştir. Toplam üç döngüden oluşan algoritmada en iç teki döngü, dıştaki iki döngü tarafından belirtilen koordinatlara ait elemanın hesabı için ilk matristen seçilen satır ile ikinci matrisin sütununu çarpar.

```
1- for (int i=0;i<N;i++){
2-       for (int j=0;j<N;j++){
3-             C[i][j]=A[i][0]*B[0][j];
4-             for (int k=1;k<N;k++){
5-                   C[i][j]=+A[i][k]*B[k][j];
6-             }
7-       }
8- }
```

Şekil 3. Standart matris çarpım algoritması
(Figure 3. Standard matrix multiplication algorithm)

Paralel hesaplama için çarpma işleminin parçalara bölünüp her bir parçanın farklı işlemci üzerinde gerçekleştirilmesi gerekir. Bu esnada yük organizasyonuna göre ya da paralelleştirilen algoritmanın yapısından dolayı iş parçacıkları arasında veri iletişimi ve eş

zamanlı işlemler (senkronizasyon) gerekebilir. Şekil 3'de verilen matris çarpım algoritmasından görüldüğü gibi en içteki döngü hariç dıştaki iki döngü parçalara ayrılarak hesaplama sırasında iş parçacıkları arasında iletişim kurmadan işlemler gerçekleştirilebilir. Hesaplama yükünün iş parçacıkları arasındaki paylaşımı farklı şekillerde olabilir. Örneğin, Şekil 4'de iki iş parçacığı kullanılarak gerçekleştirilen paralel hesaplamada matris elemanlarının iş parçacıkları arasındaki paylaşımına örnekler verilmiştir. Sonuç matrisinin elemanları birinci matrisin ilgi satırı ile ikinci matrisin ilgi sütununun çarpımlarının toplamı ile elde edilir. Bir matris elemanının hesabı diğerinde bağımsız olduğu için bu işlemler paralel hesaplama ile aynı anda gerçekleştirilir. Daha hassas yük dağılımları için satır ve sütunların çarpımı da parçalara bölünebilir fakat bu durum bazı senkronizasyon işlemleri gerektirir.



Şekil 4. İki iş parçacığı ile paralel matris çarpımı
(Figure 4. Parallel matrix multiplication using two threads)

Statik yük paylaşımlı paralel hesaplamada, yük paylaşımı işin başında yapılır ve hesaplama boyunca değişmez. Bu yaklaşım iş başlangıcında küçük bir yük belirleme hesaplaması dışında hesaplama anında ek yük getirmediği için avantajlıdır. İş parçacıkları çalışma süresince birbiriyle haberleşmesine gerek yoktur. Bundan dolayı hızlandırma yüksek seviyelerde gerçekleşir. Buna karşın iş parçacıklarından birinin gecikmesi durumda oluşacak dengesizliklerden dolayı tüm süreç gecikir. Şekil 5'de statik yük paylaşımı ile çalışan bir iş parçacığına ait örnek kodlar verilmiştir. Gerçekleştirilen uygulamada, iç içe üç döngü içeren matris çarpma algoritmasının ilk iki döngüsüne ait yükler paylaşılmıştır. Sonuç matrisinin i ve j koordinatlarındaki bir eleman en içteki *for* döngüsünün işletilmesi ile belirlenir. Dış döngüler ise *i* ve *j* koordinatlarındaki iş parçacığına göre belirtilen değerlerden başlayıp iş parçacığına ayrılan yük tamamlanana kadar çarpma işlemini sürdürür.

```
1-  void matrisCarp(int iBasla, int jBasla){
2-  i=iBasla; j=jBasla;
3-  sayac=0;
4-  DONGU:
5-  do{
6-        do{
7-              if (sayac==toplamYuk){
8-                    break DONGU;
9-              }
10-             sayac++;
11-             toplam= A[i][0]*B[0][j];
12-             for (int k=1;k<N;k++){
13-                   toplam +=A[i][k]*B[k][j];
14-             }
15-             C[i][j]= toplam;
16-             j++;
17-       } while(j<N);
18-       j=0;
19-       i++;
20- }while(i<N);
21- }
```

Şekil 5. Statik yük paylaşımlı bir paralel matris çarpma algoritması
(Figure 5. Matrix multiplication algorithm with static load partition)

Tablo 1'de 500×500 boyutlu matrislerin üç paralel iş parçacığı ile çarpımında kullanılan başlangıç değerleri ve blok büyüklükleri örnek olarak verilmiştir. Toplam işlem adedi, iş parçacığı adedine tam bölünemediği için iş parçacıklarının yüklerinde ihmal edilebilecek seviyede farklılıklar oluşmuştur.

Tablo 1. Statik yük paylaşımı kullanılarak 500×500 boyutlu matris için belirlenen başlangıç koordinatları ve yük miktarları
(Table 1. Determined start coordinates and load amounts using static load partition for 500×500 matrix)

İş parçacığı	i-başlangıç	j-başlangıç	İş parçacığı başına yük
0	0	0	83334
1	166	334	83333
2	333	167	83333

Dinamik yük paylaşımında ise Şekil 6'da görüldüğü gibi hangi iş parçacığının sonuç matrisinin hangi elemanını hesaplayacağı çalışma anında belirlenir. Burada *artir()* yöntemi tüm iş parçacıkları arasında paylaşılan ve çağırıldığında o anki sayaç değerini döndüren ve artıran senkronize yöntemi tanımlar. Bu işlem çalışma anında hızlı iş parçacıklarının daha verimli kullanılması için gerçekleştirilir. Buna karşın kontrol işleminin sık yapılması ve eşzamanlı (senkronize) işlemler gerektirmesi performansı düşürebilir. Bundan dolayı dengeleme işlemin oluşturduğu ek yükün uygun blok büyüklüğü seçilerek minimum düzeyde tutulması gerekir.

```
1-  void matrisCarp(){
2-  i=0;j=0;
3-  sayac=0;
4-  do{
5-        do {
6-              if (sayac==blokBoyutu){
7-                    n=artir()/blokBoyutu;
8-                    i = n/N;
9-                    j = n%N;
10-                   sayac=0;
11-             }
12-             sum=A[i][0]*B[0][j];
13-             for (int k=1;k<N;k++){
14-                   sum=sum+A[i][k]*B[k][j];
15-             }
16-             C[i][j]=sum;
17-             sayac++;
18-       }while(j<N);
19-       j=0;
20-       i++;
21- } while (i<N);
```

Şekil 6. Dinamik yük yönetimi için matris çarpma algoritması (Figure 6. Matrix multiplication algorithm for dynamic load management)

Algoritmanın dengeli çalışması açısından blok büyüklüğünün toplam eleman adedini kalansız bölebilir olacak şekilde belirlenmesi gerekir. Eğer tam bölünmüyorsa iş parçacıklarından bazıları diğerlerine göre daha az yük alacağı için yük dengesizliği oluşur. Bu dengesizlik miktarı, blok boyutu toplam işlem adedinde kıyasla genelde küçük sevide olduğundan önemsiz kabul edilebilir. Dinamik yük dengeleme ile çalışma anında hesaplanan yük dağılımına örnek bazı değerler iki iş parçacığı ile gerçekleştirilen paralel hesaplama için Tablo 2'de verilmiştir. Her bir iş parçacığının i ve j başlangıç değerleri görüldüğü gibi o an alınan sayaç değerine bağlı olarak belirlenmiştir. Burada verilen iş parçacığı numaraları örnek değerler olup, çalışma anında hangi iş parçacığının hangi işlem bloğunu ele alacağı önceden bilinmez.

Tablo 2. İki iş parçacığı için örnek yük dağılımı (matris boyutu: 100×100, blok boyutu=50)
(Table 2.Load distribution for two thread (matrix size: 100×100, block size=50))

Sayaç Değeri	İş Parçacığı Numarası	İ-Başlangıç	J-Başlangıç
0	1	0	0
1	2	0	50
2	1	1	0
3	2	1	50
4	2	2	0
5	1	2	50
…	…	…	…
199	2	199	50

5. DENEYSEL SONUÇLAR (EXPERIMENTAL RESULTS)

Gerçekleştirilen deneysel analizlerde Java dili ile kodlanan paralel algoritmanın başarımı günümüzde son kullanıcı düzeyinde yaygın olarak elde edilebilen çok-çekirdekli bilgisayar üzerinde incelenmiştir. Deneysel sonuçlar, Windows 7 64 bit işletim sistemi üzerinde kurulu Java Development Kit 1.7.0_03 kullanılarak elde edilmiştir. Bu amaçla kullanılan donanım, AMD Phenom 1055T x6 model numaralı 6 çekirdekli işlemciye ve 2GB Ram belleğe sahiptir. Bu donanım üzerinde statik yük dengeleme yaklaşımı ile elde edilen örnek çalışma zamanları Tablo 3'de sunulmuştur. Elde edilen ardışık sonuçlardaki farklılık işletim sistemine ait görevler, çalışmakta olan diğer uygulamalar gibi işlemci ve bellek kullanımını etkileyen faktörlerden kaynaklanır. Dikkat edilirse iş parçacıklarının görevlerini tamamlama sürelerinde de birbirlerine göre gecikmeler oluşmuştur. Algoritmanın sonlanması tüm iş parçacıklarının görevini tamamlaması ile birlikte gerçekleştiği için iş parçacıklarından birinin gecikmesi paralel hesaplama başarımını hepsi gecikmiş gibi düşürür.

Tablo 3. 500×500 boyutlu matris çarpımı için iş parçacıklarının çalışma süreleri
(Table 3. Running durations of threads for multiplication the matrices of the size of 500×500)

Çalışma	Toplam	İş Parçacıkları					
		1	2	3	4	5	6
1	120.86	120.85	114.52	115.26	113.54	114.45	115.71
2	133.44	133.43	128.78	128.59	129.88	130.65	129.56
3	129.98	129.97	122.79	125.04	123.02	123.05	124.73
4	118.97	118.96	114.00	113.77	116.08	114.41	114.42

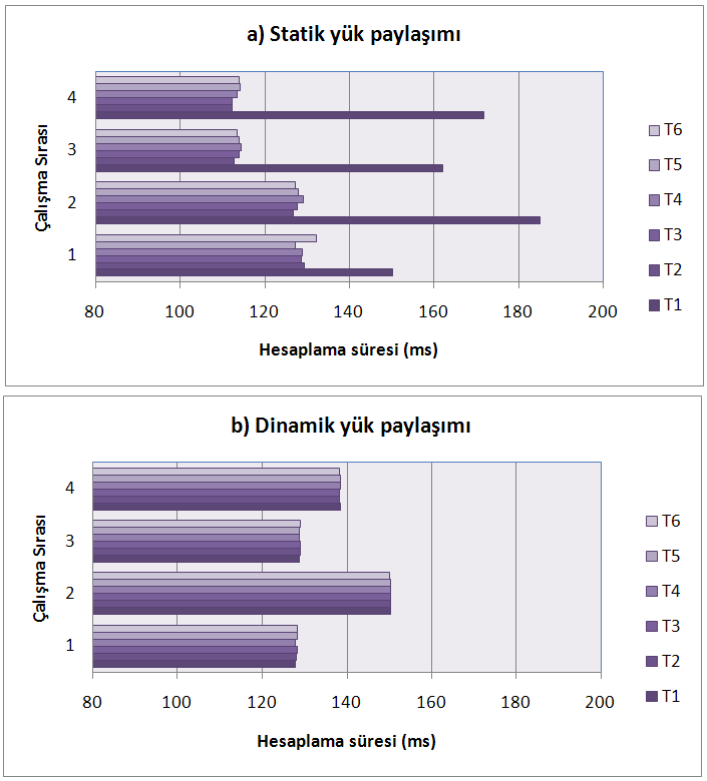
Tablo 4'de, iş parçacıklarında oluşabilecek gecikmenin tüm sürece olan etkisini azaltıp hızlı olanları daha verimli kullanmak için başvurulan pratik yöntemlerden dinamik dengeleme ile elde edilen ölçümler verilmiştir. Sonuçlar incelendiğinde iş parçacıklarının çalışma zamanları arasındaki farkın diğerlerine kıyasla oldukça az olduğu görülebilir.

Tablo 4. Dinamik yük dengeleme ile 500×500 boyutlu matris çarpımı için elde edilen sonuçlar (blok boyutu=100)
(Table 4. Results obtained by dynamic load balancing for multiplication the matrices of the size of 500×500 (block size=100))

Çalışma	Toplam	İş Parçacıkları					
		1	2	3	4	5	6
1	118.57	118.54	118.54	118.48	118.26	118.36	118.50
2	127.27	127.12	127.09	126.96	127.16	127.25	127.15
3	132.98	132.95	132.74	132.76	132.95	132.86	132.69
4	117.08	117.04	116.87	116.80	117.06	116.79	116.90

İş parçacıklarından bir tanesine çekirdeklerden birinin tamamen meşgul olması nedeniyle hesaplamaları yapamazsa veya geç kalırsa tüm süreci geciktirir. Şekil 7'de işlemcilerden biri harici bir program ile yaklaşık %50 meşgul duruma getirildiğinde iş parçacıklarının verilen görevi tamamlamada geçirdikleri zamanlara ait grafikler verilmiştir. Dikkat edilirse statik yük yaklaşımında Şekil 7a'da görüldüğü gibi iş parçacıklarından birisi diğerlerine göre oldukça

gecikmiştir. Bu durum iş parçacıklarından birinin bir süre boş çekirdek bulamamış olmasından kaynaklanır. Dinamik yük dengeleme yaklaşımını ise Şekil 7b'de görüldüğü gibi yükleri çalışma anında belirlediği için iş parçacıkları arasındaki dengesizlik oldukça düşük seviyede ortaya çıkmaktadır. İş parçacıklarının yükünü belirleyen blok boyutunun düşürülmesi iş parçacıkları arasındaki farklılıkları daha da azaltır. Bununla birlikte yük belirleme işlemi sırasında kullanılan blok sayacının ortak kullanımının getirdiği ek yük artarak çalışma sürelerinin uzamasına neden olur [13].



Şekil 7. Bir işlemcinin yaklaşık %50 meşgul olması durumunda statik ve dinamik yük dengeleme yönetimlerinden elde edilen çalışma zamanları (Figure 7. Running times for static and dynamic load scheduling in case that one of the processors busy about 50%)

Tablo 5'de yukarıda dinamik dengeleme yaklaşımı için elde edilen çalışma sürelerine ait iş parçacığı başına gerçekleştirilen hesaplama miktarları verilmiştir. İş parçacığı başına yük miktarının az olması işlemci çekirdeklerinin meşguliyetinden dolayı iş parçacığının diğerlerine göre geciktiğini göstermektedir. Yavaş iş parçacıklarının çalışmadaki etkisi düştüğü için ortaya çıkan gecikme diğer iş parçacıkları arasında paylaşılarak dengeleme sağlanmıştır.

Tablo 5. İş parçacıklarının dinamik yük dağılımı (Table 5. Dynamic load distribution of threads)

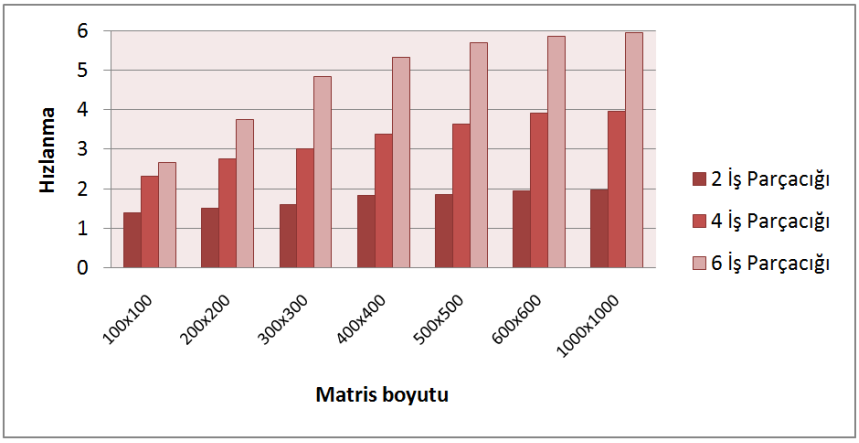
Çalışma	T1	T2	T3	T4	T5	T6
1	43900	43800	43400	43600	31800	43500
2	40500	51400	50800	39000	18200	50100
3	45500	46500	45800	45400	45800	21000
4	44500	45000	44200	43500	29700	43100

Gerçekleştirilen algoritma yapısında beklenen hız artışı doğrusal olup işlemci sayısının artmasına paralel çalışma zamanının düşmesi beklenmektedir. Paralel hesaplamanın başarımı ölçülürken genelde aynı işlemin sıralı gerçekleştirme ile elde edilen çalışma zamanı referans alınarak sağlanan hızlandırma miktarı hesaplanır. Bunun yanında paralel hızlandırmada elde edilen işlemci başına hızlandırma verimliliği de önemli bir başarım ölçütüdür. Yaygın olarak kullanılan hızlandırma ve paralel verimlilik yöntemlerine ait formüller denklem 1'de verilmiştir[17].

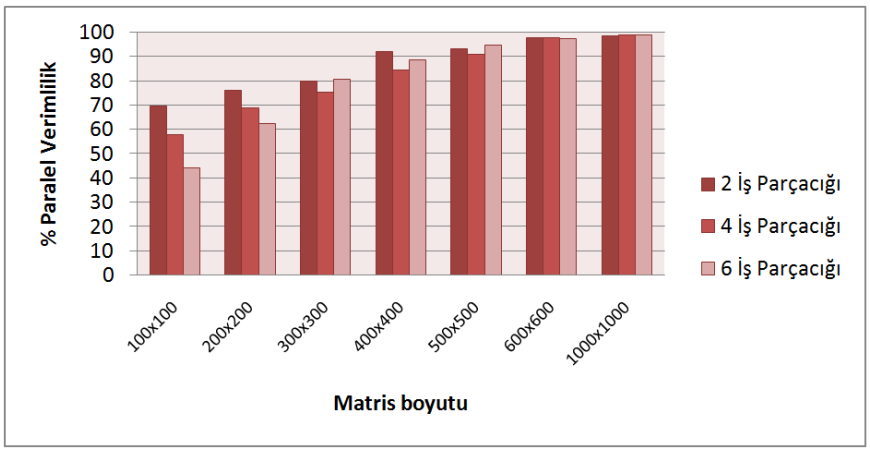
$$\text{Hızlanma} = \frac{t_{\text{sıralı}}}{t_{\text{paralel}}} \quad (1)$$

$$\%\,\text{Paralel Verim} = \frac{\text{Hızlanma}}{\text{Çekirdek adedi}} \times 100 \quad (2)$$

Elde edilen sonuçlar işletim sisteminin davranışına ve o anda yürütülmekte olan hizmetlerin durumuna göre değişeceğinden 50 kez tekrar edilerek en düşük olan değerler seçilmiştir. Ölçümler iş parçacıklarının başladığı ve sonlandığı noktalarda *System.nanotime()* ile ölçülen zaman değerlerinin farkı alınarak belirlenmiştir. Sağlanan hızlandırma miktarının matris boyutu ve iş parçacığı adedine bağlı değişimi için ölçülen sonuçlar Şekil 9'de karşılaştırmalı olarak verilmiştir. Dikkat edilirse matris boyutu arttığında paralelleştirilen işlem yoğunluğu artmakta ve hızlandırma miktarı da doğru orantılı olarak artmaktadır. Yani iş parçacığı başına düşen yük miktarını artırmak, bunların oluşturulması ve yönetilmesinden kaynaklanan ek maliyetin toplam kazancın yanında azalarak Şekil 9'da görüldüğü gibi paralel verimliliğin artmasını sağlamaktadır.



Şekil 8. Matris boyutuna bağlı paralel hızlandırma
(Figure 8. Parallel Speed-up versus matrix size)



Şekil 9. Matris boyutuna bağlı elde edilen paralel verimlilik
(Figure 9. Parallel Efficiency versus matrix size)

6. SONUÇLAR VE ÖNERİLER (CONCLUSIONS AND RECOMMENDATIONS)

Birçok matematiksel çözümleme algoritmasına temel oluşturan matris çarpma işleminin hızlandırılması, buna dayalı yürütülecek algoritmaların da performansını artırır. Sunulan çalışmada, Java dili ile yazılan paralel matris çarpma algoritmasının yerleşik kütüphanelerin sağladığı destek ile ürettiği paralel hesaplama başarımı, günümüzde son kullanıcı düzeyinde yaygın olarak elde edilebilen çok çekirdekli bilgisayar ortamında deneysel olarak analiz edilmiştir. Öncelikle, paralel hızlandırma açısından algoritmanın yükünün dağıtıldığı iş parçacıklarının aynı anda sonlandırılması problemi incelenmiştir. İş parçacıklarının dinamik yük dengeleme yaklaşımı gibi yöntemlerle kontrol edilmesi verimliliğin etkili seviyede tutulması açısından önem taşır. Gerçekleştirilen ölçümlerden, çekirdeklerden birinin meşgul olduğu ortamda iş parçacıklarından birinde oluşan gecikmenin hızlı olanlar ile kolayca telafi edilebildiği gösterilmiştir. Sıralı algoritma ve paralel eşdeğerinden elde edilen çalışma süreleri ile karşılaştırıldığında altı çekirdekli işlemciye sahip bilgisayar ile önemli seviyede hızlandırma ve paralel verimlilik elde edilebildiği grafiksel sonuçlar ile gösterilmiştir. İşlemci sayısına göre doğrusal artması beklenen hızlandırma oranı, pratikte iş parçacığı başına düşen yük miktarının artmasıyla birlikte beklenen seviyeye doğru yaklaşmıştır. Gerçekleştirilen deneysel ölçümler paylaşımlı matrissel değişkenler üzerinde gerçekleştirildiğinden, Cache bellek bakımından daha etkili şekilde çalışacak algoritmaların kullanılması performans başarımı artırabilir. Bu amaçla, gelecek çalışmalarda edilen hızlandırma değerlerini daha iyi seviyelere taşıyabilecek Cache bellek bakımından etkili yöntemlerin başarıma etkisinin analizi incelenebilir.

KAYNAKLAR (REFERENCES)

1. Nistor, A., Chin, W-N., Tan, T-S., and Tapus, N., (2009). Optimizing the parallel computation of linear recurrences using compact matrix representations, J. Parallel Distrib. Comput. 69, 373-381.
2. Olenšek, J., Tuma, T., Puhan, J., and Bůrmen, Á., (2011). A new asynchronous parallel global optimization method based on simulated annealing and differential evolution, Applied Soft Computing 11, 1481–1489.

3. Sharma, G. ve Martin, J., (2009). MATLAB®:A Language for Parallel Computing, International Journal of Parallel Programming, 37,pp:3–36.
4. Dong, W. ve Li,P., (2009). A Parallel Harmonic-Balance Approach to Steady-State and Envelope-Following Simulation of Driven and Autonomous Circuits, IEEE Transactions On Computer-Aided Design Of Integrated Circuits And Systems, Vol. 28, No. 4.
5. Hillis, W.D. and Steele, G.L., (1986). Data Parallel Algorithms Communications of the ACM.
6. Brualdi, R.A. ve Ryser, H.J., (1991). Combinatorial Matrix Theory. New York: Cambridge Univ. Press.
7. Leighton, F.T., (1992). Introduction to Parallel Algorithms and Architectures: Arrays, nTrees, Hypercubes. Morgan Kaufmann Publishers Inc. San Francisco.
8. Frost Gorder, P., (2007). Multicore Processors for Science and Engineering, IEEE Computing in Science & Engineering, 9(2), pp:3-7 .
9. Andrews, G., (2000). Foundations of Multithreaded, Parallel, and Distributed Programming, Addison-Wesley.
10. Warg, F. ve Stenstrom, P., (2008). Dual-thread Speculation: A Simple Approach to Uncover Thread-level Parallelism on a Simultaneous Multithreaded Processor, International Journal of Parallel Programming, 36, pp:166–183.
11. Sand, B., (2004). Coping with Java threads, IEEE Computer 37(4), pp:20-27.
12. Lea, D., (1997). Concurrent Programming in Java: Design Principles and Patterns, Addison-Wesley
13. Yue, K.K. ve Lilja, D.J., (1994). Parallel Loop Scheduling for High Performance Computers. Technical Report No: HPPC-94-12. University of Minessota.
14. Polychronopoulos, C.D. ve Kuck, D., (1987). Guided Self-Scheduling: a Practical Scheduling Scheme for Parallel Supercomputers, IEEE Trans. on Computers, vol. 36(12) pp:1425-1439.
15. Liu, J., Saletore, V.A. ve Lewis, T.G., (1994). Safe Self-Scheduling: A Parallel Loop Scheduling Scheme for Shared-Memory Multiprocessors, Int J. Parallel Programming, vol. 22, no. 6, pp. 589-616.
16. Yan, Y., Jin, C. ve Zhang, X., (1997). Adaptively Scheduling Parallel Loops in Distributed Shared-Memory Systems, IEEE Transactions On Parallel And Distributed Systems, 8(1), pp:70-81
17. Eager, D.L., Zahorjan, J. ve Lazowska, E.D., (1989). Speedup Versus Efficiency in Parallel Systems, IEEE Transactions on Computers, 38(3), pp:408-423.