MSU Fen Bil. Dergi., Cilt 6, Sayı 2 , s 545-552 Araştırma Makalesi/ Research Article
MSU J. of Sci., Volume 6, Issue 2, p 545-552 DOI : 10.18586/msufbd.440544

Yoğun İşlem Yüküne Sahip Matris Çarpımı Hesaplama Sürelerinin Önbellek Kullanım Optimizasyonu ve Paralel Programlama Teknikleri Kullanılarak İyileştirilmesi

Mustafa Furkan KESKENLER[1], Eyüp Fahri KESKENLER[2]
[1] Bilgisayar Mühendisliği, Mühendislik Fakültesi, Atatürk Üniversitesi, Erzurum, Türkiye
[2] Malzeme Bilimi ve Nanoteknoloji Mühendisliği, Mühendislik Fakültesi, Recep Tayyip Erdoğan Üniversitesi, Rize, Türkiye
✉: keskenler@gmail.com

Geliş (Received): 04.07.2018 Düzenleme (Revision):26.07.2018 Kabul (Accepted): 15.08.2018

ÖZ

Başta görüntü işleme/iyileştirme ve robotik olmak üzere, ekonometri, inşaat mühendisliği, kuantum fiziği gibi birçok alanda hesaplama işlemlerinde yaygın olarak kullanılan matris işlemleri üzerine cache bellek kullanım optimizasyonu yapılarak hesaplama sürelerinin değişimi incelenmiştir. Çalışmada, bilinen matris çarpma işlemi yerine kullanılan farklı algoritmalarla zamanda ve mekanda yerellik prensiplerinden yararlanılarak hesaplama performansı 14 kat daha hızlı hale getirilmiştir. Bilgisayar performansında çok önemli yeri olan ön belleğin etkisinin önemi saptanmaya çalışılmıştır. Ön bellek kullanım optimizasyon yöntemleri ile önemli ölçüde işlem süreleri kısaltılmış ayrıca ön bellek ve diğer işlem birimlerinin daha fazla kullanılmasının önüne geçilerek sistemin ömrünün uzatılması hedeflenmiştir. Cache optimizasyonunun ardından paralel programlama teknikleri kullanılarak yoğun matris işlemlerinin hesaplama sürelerinin kısaltılması amaçlanmıştır. Böylece hem ön bellek daha etkin kullanıldı hem de uygun veri paralelliği kullanılarak mümkün olabilecek en verimli hesaplama işlemlerinin gerçekleştirilmesi sağlanmaya çalışıldı. Cache optimizasyonu ardından yapılan 5 bilgisayarlı paralel programlama tekniği sayesinde hesaplama işlemi genel olarak kullanılan tekniğe göre yaklaşık 59 kat daha hızlı hale getirildi. Paralel programlama ile yapılan hesaplama işlemlerinde farklı sayıda bilgisayarlara göre Speedup değerleri hesaplandı. Ayrıca matris işlemleri için bilgisayarla yapılan hızlı hesaplama yöntemlerinin yanında hesaplama işlemlerine negatif etki gösteren algoritmalar üzerinde de durulmuştur.
Anahtar Kelimeler: Matris Çarpımı, MPI, Optimizasyon, Önbellek Optimizasyonu, Paralel Programlama

Acceleration of High Dimensional Matrix Multiplication Processes With Cache Usage Optimization and Parallel Programming Techniques

ABSTRACT

Optimization of cache on the matrix operations according to calculation times, which are commonly used in many field such as image processing, robotics, econometrics, civil engineering and quantum physics were investigated. The calculation performance is made 14 times faster than using the different algorithms instead of the known matrix multiplication, by using the temporal and spatial locality principles in the study. Effect of the cache which is very important for computer performance, has been tried to detect. Processing times have significantly shortened thanks to cache utilization optimization methods, also it has been aimed that extending the lifetime of the system by preventing use the cache and other processing units more. After cache optimization, it has been aimed to shorten the computation times of intensive matrix operations using parallel programming techniques. Thus, both the cache was used more efficiently and the most effective computations were tried to be performed using the appropriate data parallelism. Thanks to parallel programming techniques applied with 5 computers after cache optimization, the calculation process is made 59 times faster than the commonly used technique. Speedup values are calculated according to the different number of computers in the calculations made by parallel programming. In addition to computer-based fast computation methods for matrix operations, also we have focused on algorithms that have a negative effect on computation.
Keywords: Cache Optimization, Matrix Multiplication, MPI, Optimization, Parallel Programming

GİRİŞ

Matrisler birçok mühendislik ve bilim dalında kullanılan hesaplama yöntemlerinden birisidir [1]. Fizik, optik, elektromanyetizma, kuantum mekaniği, bilgisayar grafikleri, robotik gibi birçok alanda yüksek boyutlu matris işlemleri kullanılmaktadır [2-5]. Bu durumlarda yüksek performanslı hesaplama gerektiren durumlar oluşmaktadır. Paralel programlama teknikleri ile çok yüksek performans özelliklerine sahip olmayan CPU'ların birlikte çalışma gücünden faydalanılarak bu hesaplamalar hem daha ekonomik hem de daha hızlı sonuçlandırılmaktadır [6]. Çalışmada büyük boyutlu matris işlemleri, uzun zaman gerektiren hesaplamalar içerdiğinden tercih edilmiştir. Bu işlemler için cache

belleği daha etkin kullanacak ve bellekler arası transfer işlemlerini azaltacak algoritmalar üzerinde durulmuştur.
MPI (Message Passing Interface) bilgisayar iletişim protokollerinden birisidir. Dağıtık bellek modeli kullanılan bir sistemde bilgisayarlar arasında paralel program çalıştırmak için kullanılan standart bir iletişim protokoldür. MPI hızı ve taşınabilir olması gibi özelliklerinden dolayı daha önceleri kullanılan diğer kütüphanelere oranla üstündür. MPI'ın hızı, oluşturulan uygulamaların yürütüleceği donanıma göre optimizasyon edilebilmesinden kaynaklanmaktadır [7]. MPI kütüphanesi kullanılarak oluşturulan uygulamalar için genellikle Fortran, C, C++ ve Ada programlama dilleri kullanılır. MPI, birden fazla düğüm üzerinde bir programın çalıştırılması ile beraber düğümler arasında iletişim kurularak veri ve buyrukların taşınması işlemlerini kapsamaktadır. Böylece MPI veya başka kütüphaneler kullanılarak gerçekleştirilen paralel programlama ile tek bilgisayar üzerinde yürütülen uzun zaman alan işlemlerin birden fazla bilgisayara veri veya görev paralelliği mantığı ile bölünerek daha kısa sürede işlemlerin tamamlanması hedeflenmektedir.
CPU sistem belleğinden doğrudan gelen verilerin işlenmesinde verilerin transfer süreci hızına göre daha süratlidir. Bunun için CPU içerisinde transfer süresi kaybı oluşturmamak adına çok yüksek hızlı hafızalar (cache, register) yerleştirilmiştir. Ön bellekler işlenecek görevlere ait veri ve komutların geçici olarak tutulduğu düşük depolama boyutlu, çok yüksek hızlı ve çok maliyetli hafızalardır. Moore Yasası'nın [8] belirttiği üzere işlemci performansının bellek performansına oranı her yıl yaklaşık olarak yarı yarıya artmaktadır. Bu zamanla, işlemci hızının bellek hızına göre zamanla daha çok artacağı anlamına gelmektedir. İşlemci ile bellek arasındaki bu hız farkı kapatmak mümkün olmadığından bu sorunu çözmek için farklı yaklaşımlar geliştirilmiştir. Çünkü işlemci hızı gittikçe yüksek performanslar sergilemesine rağmen bu hız tam verimle kullanılamamaktadır. Yani burada bellek performansı kısıtlayan etken olmuştur. Geliştirilen çözüm olarak ön bellek, ilk olarak 1980 yıllarında [9] işlemci saat frekansının 20 MHz değerlerine ulaşmasından sonra bellek-işlemci arasında aracı görevi üstlenmesi amacıyla üretilmiş ve işlemci yonga seti üzerinde kullanılmaya başlanmıştır.
Ön belleğin görevi şu şekildedir, işlemci yeni veri ve buyruklara ihtiyaç duyar. Ardından veri ve buyruklar için işlemci önce ön belleği kontrol eder. Eğer gereksinim duyduğu yapılar burada bulunuyorsa işlemci bu veri ve buyrukları sistem belleğine erişmeden ön bellekten alır ve kullanır. Eğer işlemci aradığını ön bellekte bulamaz ise bu kez ikinci seviyede (L_2) ön belleği kontrol eder daha sonra sistem belleğini kontrol eder. Eğer veriler sistem belleğinden getirilecekse getirilen bu veriler doğrudan işlemciye aktarılmaz. Öncelikle ön bellek silinir ve yeni veriler tekrar yazılır. Bunun için yaz-sil denetim mekanizmaları kullanılır. İşlenen verinin sonuçları tekrar sistem belleğine gönderilirken bu denetim sistemi kullanılır. Yazma denetimi bu aşama için genellikle bir yazma ara belleği kullanır [10]. Böylece yürütülecek işleme ait veri ve buyruk transfer işlemleri sonuçlandırılır.

Matris Çarpımı

Matematiksel işlemlerde matris çarpımı boyutları birbiri ile uyumlu iki matris kullanılarak başka bir matrisin üretildiği işleme denir. Karmaşık sayılar, reel sayılar gibi genel olarak bütün sayılarda aritmetik hesaplamalara uygun olarak çarpım işlemi yapılabilmektedir. Matrisler aynı zamanda sayı dizileri oldukları için matris çarpımı terimini tek bir yöntemle açıklamak doğru olmaz. Matrislerin satır ve sütun sayıları, matrisin boyutu olarak isimlendirilmektedir. Ayrıca, birçok matris içeren işlemler, genellikle çarpım işlemleri içermektedir [11]. Bu nedenle, çoğu yüksek işlem gücü gerektiren algoritmalar için matris çarpımı, performans optimizasyonunun temel odağıdır [12].

A (mxn) ve B (nxp) iki matristir. AB çarpımı elemanları,

$$\text{ent}_{ij}(AB) = \text{Row}_i(A)\ \text{Col}_j(B)$$
$$= (a_{i1}, a_{i2}, \ldots, a_{in}) \begin{pmatrix} b1j \\ \cdots \\ bnj \end{pmatrix}$$
$$= a_{i1}b_{1j} + a_{i2}b_{2j} + \ldots + a_{in}b_{nj}$$

biçiminde tanımlı m x p boyutlu matris olarak tanım yapılmaktadır. Toplam sembolü ile bu bağıntı,

$$\text{ent}_{ij}(AB) = \sum_{k=1}^{n} \mathbf{a}ik\ \mathbf{b}kj = \sum_{k=1}^{n} \mathbf{ent}ik\,(A)\ \mathbf{ent}kj\,(B) \quad (1)$$

şeklinde yazılabilir [13].

MATERYAL ve YÖNTEM

Çalışmada yüksek boyutlu matris çarpım işlemleri için hesaplama süresini kısaltan iki yöntem artarda uygulanmıştır. İlk olarak işlemcinin önbelleğinin, mekansal ve zamansal yerellik prensiplerinden yararlanarak daha etkin kullanılması amaçlanmıştır. Daha sonra matris çarpım işlemleri için paralel programlama teknikleri uygulanarak hesaplama süresi oldukça kısaltılmıştır.

Önbellek Kullanım Optimizasyonu

Cache bellek kullanım optimizasyonu ile işlemcinin bu belleğe oranla daha yavaş çalışan Ram ve bilgisayar hafızası üzerinde daha fazla işlem gerçekleştirmesinin önüne geçilerek [14, 15] işlem sürelerine olan etkisi incelendi. Ayrıca, işlemcileri dinamik olarak tahsis etmeye çalışan sistemler için genellikle yüksek performanslı bir veri yolu içermesi gerektiği bulunmuştur [16, 17]. Bunun için çalışmada bu koşula dikkat edilmiş fakat çok yüksek ön bellek ve işlemci saat hızına sahip olmayan bilgisayarlar seçilerek ön bellek optimizasyonu etkisinin daha belirgin olması amaçlanmıştır. Böylece uzun süren hesaplama işlemleri

bilgisayarlarda bulunan kısıtlı ön bellekten dolayı ortaya çıkan bant genişliği erişim süresinden büyük ölçüde etkilenmektedir ve bellekten gelen iki kat veriyi almak normal olarak bir kat verinin getirilmesi işleminden iki kat daha uzun sürmektedir [18]. Çalışmada hesaplama işlemleri için bir adet 32 bitlik 1.3 GHz saat hızında L_1=32 KB ve L_2 = 512 KB ön bellekli tek çekirdek tek işlemcili 512 MB Ram ile çalışan Linux Fedora işletim sistemine sahip bilgisayarlar seçildi. Ön bellek optimizasyonunu test edecek yazılım C++ programlama dili ile yazıldı. Yazılımda iki matrisin çarpımı için iç içe 3 for döngüsü kullanıldı. Bu for döngülerinde isimleri i, j ve k olan 3 adet döngü değişkenleri mevcuttur. Matris çarpımı için 3 matris kullanıldı; Birinci matris (matrixAPtr), ikinci matris (matrixBPtr) ve sonuç matrisi (matrixCPtr)dir. Programda matris çarpımları için belirtilen şu kod parçacığı yazıldı;

```
for (int i = 0; i < byt; i++){
        for (int j = 0; j < byt; j++){for(int k=0;k<byt;k++)
          matrixCPtr[i][j]+=matrixAPtr[i][k]*matrixBPtr[k][j];} }
```

En içte bulunan for döngüsü içerisinde yer alan bu kod ile her bir adımda sonuç matrisinin bir elemanı hesaplandı. Yukarıda isimleri geçen i, j, k döngü değişkenlerini farklı kombinasyonlarda 6 (3! = 6 kombinasyonda) for döngülerinin her birisi için değiştirerek cache belleğe getirilecek veriler için işlemcinin ram ve bilgisayar hafızasına erişimi azaltıldı. Cache bellek için zamanda ve mekanda yerellik prensiplerinden yararlanılarak hesaplama işlemleri daha verimli hale getirildi. Süre ölçüm işlemlerinde epoch time kullanıldı. Hesaplama işlemi olarak 1000x1000, 2000x2000, 3000x3000, 4000x4000 ve 5000x5000 boyutlu kare matrislerin aynı boyuttaki matrislerle çarpma işlemi gerçekleştirildi. Matrisler de kullanılan değişkenler integer (4 byte) veri tipindedir.

Bir bilgisayarda program şu şekilde çalışır. Buyruklar ve veriler bellekten ön belleğe taşınır. Sonra işlemci ön bellek üzerinde işlemleri tamamlar ve sonucu tekrar RAM e gönderir. Ön belleğe kısıtlı sayıda veri getirilebilir [19]. Bu nedenle çok yüksek boyutlarda matris çarpımı yapıldığından verilerin tamamı ön belleğe sığmaz ve ön bellek ile bellek arasında sürekli işlenecek veriler getirilip götürülür. Bu da çok fazla zaman harcamak demektir. Şöyle ki, işlemcinin işlem yapacağı verileri ön belleğe getirirken, gelen verinin işi bitinceye kadar olabildiğince ön bellekte tutulduktan sonra diğer verilere erişilmelidir. Böylece veri yolu trafiğinde kaybedilen zaman minimuma indirilmiş olur.

2 matrisin çarpımını gerçekleştiren kod yazımlarında iç içe 3 for döngüsü kullanılmaktadır. Çarpma işlemi bu döngü işlemleri sonucunda tamamlanmaktadır. Bu çalışmada ön bellek kullanım optimizasyonu, bu for döngülerinin bulunduğu kod parçasında yapılan basit değişiklerle gerçekleştirilmiştir. Bu değişiklik, i, j ve k döngü değişkenlerinin tablo 1 de görüldüğü gibi iç içe 3 for döngüsünde sırasının değiştirilmesini kapsamaktadır.

Tablo 1. Matrixaptr Ve Matrixbptr İsimli Matrislerin Çarpma İşlemi İçin Kullanılan Farklı Algoritmalar

Algoritmalar	Kod

i→k→j	for (int i = 0; i < byt; i++){
	for (int k = 0; k < byt; k++){
	for(int j = 0; j < byt; j++)
	matrixCPtr[i][j]+=matrixAPtr[i][k]*matrixBPtr[k][j];}}

k→i→j	for (int k = 0; k < byt; k++){
	for (int i = 0; i < byt; i++){
	for(int j = 0; j < byt; j++)
	matrixCPtr[i][j]+=matrixAPtr[i][k]*matrixBPtr[k][j];}}

i→j→k	for (int i = 0; i < byt; i++){
	for (int j = 0; j < byt; j++){
	for(int k = 0; k < byt; k++)
	matrixCPtr[i][j]+=matrixAPtr[i][k]*matrixBPtr[k][j];}}

j→i→k	for (int j = 0; j < byt; j++){
	for (int i = 0; i < byt; i++){
	for(int k = 0; k < byt; k++)
	matrixCPtr[i][j]+=matrixAPtr[i][k]*matrixBPtr[k][j];}}

k→j→i	for (int k = 0; k < byt; k++){
	for (int j = 0; j < byt; j++){
	for(int i = 0; i < byt; i++)
	matrixCPtr[i][j]+=matrixAPtr[i][k]*matrixBPtr[k][j];}}

j→k→i	for (int j = 0; j < byt; j++){
	for (int k = 0; k < byt; k++){
	for(int i = 0; i < byt; i++)
	matrixCPtr[i][j]+=matrixAPtr[i][k]*matrixBPtr[k][j];}}

Tablo 1 da gösterilen algoritmaların 2x2 boyutunda matrislerin çarpma işleminin program çalışma adımları Tablo 2 da verilmiştir. Tablo 2 da yer alan 6 farklı algoritmalardan; i→k→j ve k→i→j, i→j→k ve j→i→k, k→j→i ve j→k→i ikili algoritmaları önbelleği birbirlerine yakın mantıkta kullandığı program çalışma

adımlarından görülmektedir. Tablo 1 da yer alan i→j→k algoritması, insanların günlük hayatta (eş. 1) matris çarpma hesaplamalarında kullandığı bilinen bir yöntemdir.

Çalışmada oluşturulan hesaplama süresine pozitif ve negatif etki gösteren algoritmaların ve genel olarak matris çarpımı işlemlerinde kullanılan algoritmanın önbelleğe verilerin taşınması işleminin basit örneği şu şekildedir,

matrixAPtr **matrixBPtr** **matrixCPtr**

$$\begin{bmatrix} x1 & x2 \\ x3 & x4 \end{bmatrix} x \begin{bmatrix} y1 & y2 \\ y3 & y4 \end{bmatrix} = \begin{bmatrix} z1 & z2 \\ z3 & z4 \end{bmatrix} \quad (2)$$

2x2 boyutlu eş. 2'deki iki matris 6 farklı döngü değişkeni sıralamasına göre Tablo 2 da çarpma işleminin nasıl yapıldığı program adımlarına göre gösterilmiştir.

Tablo 2. 2x2 Boyutunda Matrixaptr Ve Matrixbptr İsimli Matrislerin Çarpma İşlemi İçin Kullanılan Farklı Algoritmalara Göre Program Çalışma Adımları

	Algoritmalar					
Program Adımları	**i→k→j**	**k→i→j**	**i→j→k**	**j→i→k**	**k→j→i**	**j→k→i**
Adım 1	*z1=z1+x1.y1*	*z1=z1+x1.y1*	*z1=z1+x1.y1*	*z1=z1+x1.y1*	*z1=z1+x1.y1*	*z1=z1+x1.y1*
Adım 2	*z2=z2+x1.y2*	*z2=z2+x1.y2*	*z1=z1+x2.y3*	*z1=z1+x2.y3*	*z3=z3+x3.y1*	*z3=z3+x3.y1*
Adım 3	*z1=z1+x2.y3*	*z3=z3+x3.y1*	*z2=z2+x1.y2*	*z3=z3+x3.y1*	*z2=z2+x1.y2*	*z1=z1+x2.y3*
Adım 4	*z2=z2+x2.y4*	*z4=z4+x3.y2*	*z2=z2+x2.y4*	*z3=z3+x4.y3*	*z4=z4+x3.y2*	*z3=z3+x4.y3*
Adım 5	*z3=z3+x3.y1*	*z1=z1+x2.y3*	*z3=z3+x3.y1*	*z2=z2+x1.y2*	*z1=z1+x2.y3*	*z2=z2+x1.y2*
Adım 6	*z4=z4+x3.y2*	*z2=z2+x2.y4*	*z3=z3+x4.y3*	*z2=z2+x2.y4*	*z3=z3+x4.y3*	*z4=z4+x3.y2*
Adım 7	*z3=z3+x4.y3*	*z3=z3+x4.y3*	*z4=z4+x3.y2*	*z4=z4+x3.y2*	*z2=z2+x2.y4*	*z2=z2+x2.y4*
Adım 8	*z4=z4+x4.y4*	*z4=z4+x4.y4*	*z4=z4+x4.y4*	*z4=z4+x4.y4*	*z4=z4+x4.y4*	*z4=z4+x4.y4*

Tablo 3 de i, j ve k harflerinin sıralanışı sırasıyla birinci, ikinci ve üçüncü döngü değişkeni olarak kullanıldığı zamanki değerleri ifade etmektedir. i→k→j algoritması ile en yavaş çalışan k→j→i algoritması arasında 33 kat hız farkı var. Genel olarak matris çarpımı için kullanılan i→j→k algoritmasından ise i→k→j algoritması 14 kat daha hızlı çalışmaktadır. Algoritmalar arasında oluşan bu kadar büyük hız farklarının nedeni algoritmalar arasındaki önbellek kullanım farkından kaynaklanmaktadır.

Tablo 3. 6 Farklı Algoritmaya Göre 5 Farklı Boyuttaki İki Kare Matrisin Çarpma İşlemi İçin Program Çalışma Süreleri

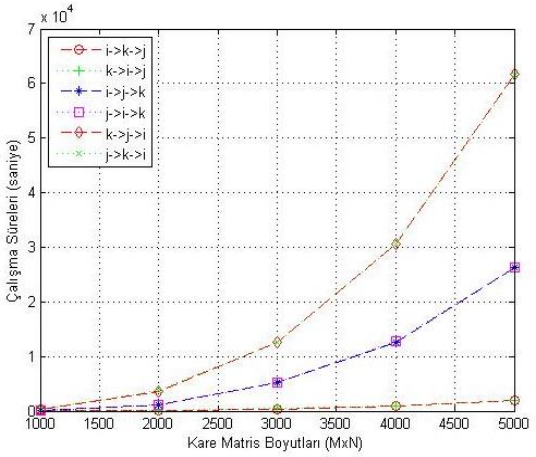
	Hesaplama Süreleri (sec)					
Matris Boyutu (MxN)	**i→k→j**	**k→i→j**	**i→j→k**	**j→i→k**	**k→j→i**	**j→k→i**
1000x1000	15.1295	15.4817	35.0544	45.796	280.038	282.288
2000x2000	119.932	123.386	1062.47	1140.81	3683.35	3691.85
3000x3000	402.785	412.826	5186.06	5203.22	12653.1	12651.8
4000x4000	960.47	990.041	12603.1	12792.2	30617.4	30627.6
5000x5000	1857.6	1916.0	26318.1	26401.8	61810.1	61561.2

Matris çarpma işlemlerinin gerçekleştirildiği iç içe 3 for döngüsü değişkenlerinin kullanım sırasına göre i,k,j ve k,i,j şeklinde uygulanan algoritmalarının çalışma mantığına bakıldığında, matrixAPtr matrisi ve matrixBPtr matrisinin sahip olduğu elemanlarının tekrar tekrar gereksizce önbelleğe getirilmediği görüldü. matrixAPtr matrisindeki bir eleman ve matrixBPtr matrisindeki bir satırın işi bitinceye kadar önbellekte kalmakta ve işi bittikten sonra önbellekten silinmektedir.

Çarpma işleminin yapıldığı iç içe 3 for döngü değişkenlerinin sırasıyla i,j,k ve j,i,k şeklinde uygulanan algoritmalarda matrixBPtr matrisinin her bir sütununun matrixAPtr matrisinin her satırı için bir verinin görevi tamamlanmadan tekrar tekrar ön belleğe getirildiği görülmektedir. En içteki for döngüsünde matrixAPtr matrisinin sütunları ile matrixBPtr matrisinin satır elemanları çarpılarak matrixCPtr matrisin bir elemanı tamamen elde edilmektedir. Buradaki i,j,k algoritması insanların standart olarak matris çarpma işlemlerinde uyguladıkları yöntemi temsil etmektedir. Bir dıştaki yani ortadaki for döngüsü hesaplama süresince matrixBPtr matrisinin ön bellekteki elemanları silinerek sıradaki sütun elemanlarını ön belleğe almaktadır. Bu hesaplama algoritmalarına göre en dıştaki for döngüsü çalıştığı sürece matrixAPtr matrisinin her satırı için matrixBPtr matrisinin her sütunu belleğe tekrar tekrar alınmaktadır. Bunun sonucunda bu iki algoritma i,k,j ve k,i,j algoritmalara göre daha yavaş çalışmaktadır. Çünkü fazladan ön belleğe taşıma ve silme işlemi gerçekleşmektedir.

Son olarak k,j,i ve j,k,i döngü değişkenlerinin sıralamasına göre oluşturulan algoritmalarda ön belleğe veri taşıma işlemi en verimsiz şekilde gerçekleştirilmiş,

iş yükü ve veri trafiği artırılmıştır. Sürekli ön belleğe taşıma ve silme işlemi gerçekleştirilmiş ve bir verinin görevi bitmeden bir başka veri ön belleğe getirilip çalıştırılmıştır. Daha sonra tekrar görevi bitmeyen veri ön belleğe getirilmiş işlemlere bu döngüde devam edilmiştir. Bu nedenle en verimsiz algoritmalar bunlardır. Şekil 1 de kullanılan 6 farklı algoritmaya ait çalışma süreleri verilmiştir. Şekil 1 de i→k→j ve k→i→j algoritması hesaplama süreleri ile k→j→i ve j→k→i algoritması çalışma süreleri arasındaki fark belirgin olarak görülmektedir.



Şekil 1. Aynı Boyutlu İki Kare Matrisin Çarpma İşlemi İçin Farklı Döngü Değişkenleri Sıralamalarına Göre Çalışma Süreleri

Paralel Programlama Tekniğinin Uygulanması

Önbellek optimizasyonu sonunda en verimli çalışan algoritma (i→k→j) tespit edilip bu algoritma paralel programlama tekniğine uyarlanarak hesaplamalar paralel hesaplanabilir hale getirildi. Çalışmada sayıları 10 adeti bulan ve yukarıda belirtilen aynı özelliklere sahip bilgisayarlarla Beowulf kümesi yöntemi kullanılarak dağıtık bellekli model yöntemine göre paralel programlama kümesi oluşturuldu. Tek program çoklu veri paralel programlamayı destekleyen C, C++ [20, 21] programlama dillerinden C++ dili kullanılan çalışmada MPI kütüphanesinden faydalanılarak kendi oluşturduğumuz algoritma ile yazılım gerçekleştirildi. MPI programlarında bir dereceye kadar yapılandırma, iç içe geçmiş paralellik için hiyerarşik grup konsepti ve paralel yazılım bileşenleri için ayrı iletişim bağlamları oluşturmayı sağlayan iletişimci kavramı tarafından sağlanmaktadır. Rauber ve Rünger, bu işlevselliğin üstünde yuvalanmış paralel çok işlemeli görevleri yönetmek için bir kütüphane sağladı [22]. Ayrıca, vektörler ve matrisler gibi spesifik dağıtılmış veri yapıları için MPI kütüphaneleri literatürde önerilmiştir [23].

Öncelikle program seri program mantığı ile tek bilgisayar üzerinde çalıştırıldıktan sonra hesaplama süreleri kayıt altına alındı. Daha sonra 5 bilgisayar ile paralel programlama yöntemine göre data paralelliği kullanılarak hesaplama yapıldı. Ardından aynı işlem 10 bilgisayar ile yapıldı. Paralel hesaplamalarda birinci bilgisayar masternode olarak seçildi. Bütün bu işlemler her defasında 3 kere tekrar edilip kayıt altına alındı. Ayrıca masternode çarpma işleminde kullanılacak matrislerin içerisindeki elemanlara rastgele ilk değer atama işlemini yapmaktadır. Bu yapılan işlem speedup'ı düşüren etmen olarak yansımaktadır [24]. Bunun nedeni matrislere rastgele ilk değer atama işleminin paralel hesaplamaya dahil edilmeyip tek bilgisayar üzerinde (masternode) yapılmasıdır.

Görev dağılımı (Taxonomy)

Algoritma 1 de görüldüğü gibi ilk önce masternode kullanıcının girdiği argümanları okuyup matrislerin boyutuna karar vermektedir. Ardından matrixAPtr ve matrixBPtr matrislerine rastgele değer atamaktadır.

Algoritma 1. Paralel Programlama İle Matris Çapımı İçin Kullanılan Algoritmanın Sahte Kodu

```
 1: input: dimension,myRank,size
 2: part = dimension / size
 3: if myRank ?= 0 then  //myRank=0 –> master node
 4:         for m = 0,dimension do
 5:                 for n = 0,dimension do
 6:                         matrixAPtr_mn = random()
 7:                 end for
 8:         end for
 9:         for m = 0,dimension do
10:                 for n = 0,dimension do
11:                         matrixAPtr_mn = random()
12:                 end for
13:         end for
14: end if
15: MPI_Bcast()
16: start = myRank * part
17: finish = start + part
18: for i = start,finish do
19:         for k = 0,dimension do
20:                 for j = 0,dimension
    do
21:
    matrixCPtr_ij+=matrixAPtr_ik*matrixBPtr_kj
22:                 end for
23:         end for
24: end for
25: MPI_Barrier() //implicit barrier
26: MPI_Gather()
27: MPI_Finalize()
28: Output: matrixCPtr
```

Daha sonra toplu haberleşme rutinlerini kullanarak matrixAPtr ve matrixBPtr matrislerini diğer bilgisayarlarla paylaşmaktadır. Her bilgisayar masternode da dahil matrixBPtr matrisinin tamamını kullanmaktadır. Kümedeki her bilgisayar sırasıyla matrixAPtr matrisinin "boyut/bilgisayar sayısı" kadar satırını kullanmaktadır. Sonuçta her bilgisayara ait işlemci matrixCPtr matrisinin kendisine düşen belli bir satırını hesaplamaktadır Paralel programlama kümesinde yer alan her bilgisayar hesapladığı bu değerleri masternoda göndermektedir. Masternode da bu satırları matrixCPtr matrisinde birleştirip istenildiği takdirde ekrana basmaktadır. Ayrıca masternode programın çalışma süresini de hesaplamaktadır. Tablo 4 de hesaplanan değerler farklı zamanlarda ve aynı koşullara göre üç kere tekrar edilip hata oranı minimuma indirilmeye çalışılmıştır.

BULGULAR ve TARTIŞMA

Tablo 4 de ölçülen 3 farklı değerin ortalamaları alınarak speedup değerlerinin hesaplanacağı değerler tablo 5 de elde edilmiştir. Tablo 5 de seri program için yer alan süreler dikkate alınırsa şekil 2 de de ayrıntılı olarak görüldüğü üzere eksponansiyel olarak artan değerler artan iş yüküne karşı tek işlemcinin yetersiz kalmasını göstermektedir.

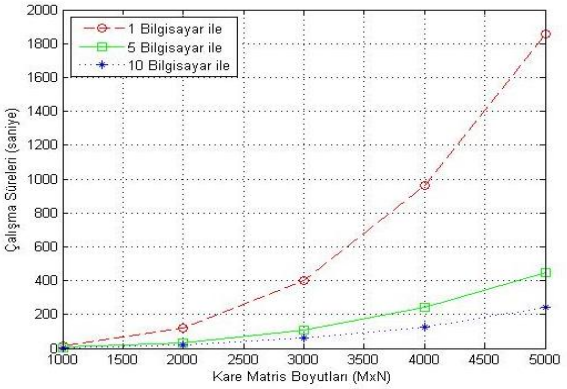
Tablo 4. Farklı Boyutlardaki Kare Matrislerin Çarpma İşlemleri İçin 5 Ve 10 Bilgisayarlı Paralel Programlar İçin Toplam Çalışma Süreleri

Boyut (MxN)	Ölçülen Süreler (sec)	
	5 Bilgisayar ile	10 Bilgisayar ile
	1. Ölçüm	
1000x1000	5.60247	3.77668
2000x2000	34.2094	19.7988
3000x3000	113.872	57.3474
4000x4000	243.798	125.454
5000x5000	454.147	253.406
	2. Ölçüm	
1000x1000	6.03794	5.0294
2000x2000	34.6254	19.8358
3000x3000	104.175	62.5309
4000x4000	248.236	125.327
5000x5000	445.455	240.598
	3. Ölçüm	
1000x1000	5.52899	3.70069
2000x2000	34.2268	19.7909
3000x3000	112.408	57.2612
4000x4000	238.748	125.471
5000x5000	448.013	241.474

Yine Tablo 5 de, 5 bilgisayar ile ve 10 bilgisayar ile çalışan programlarda 2000x2000 boyutlu iki kare matrisin çarpma işlemine kadar olan kısımlarda hesaplama sürelerinde çok fazla fark bulunmadığı görülmektedir. Bunun nedeni bu boyutlara kadar olan matris çarpım işlemlerinde işlem yükü fazla olmadığından hesaplamadan kaynaklı değil de bilgisayarlar arası iletişimden kaynaklanan süre kaybının artmasından dolayı bu hızın uygulama sonuçlarına yansımamasıdır.
Çalışmada kullanılan işlemcilerin düşük performansı göz önüne alındığında kullanılacak bilgisayarların işlem gücü arttıkça paralel programlamanın kazançları daha yüksek boyutlu matris çarpma işlemlerinden itibaren fark edilebilir seviyeye gelebilir.

Tablo 5. Farklı Matris Boyutlarına Sahip Çarpma İşlemlerinin Seri Ve Paralel Programlar İçin Ortalama Çalışma Süreleri

	Hesaplama Süreleri (sec)		
Boyut (MxN)	**Seri Program**	**5 Bilgisayar ile**	**10 Bilgisayar ile**
1000x1000	15.1295	5.72313	4.16892
2000x2000	119.932	34.3539	19.8085
3000x3000	402.785	110.152	59.0465
4000x4000	960.47	243.594	125.417
5000x5000	1857.6	449.205	245.159



Şekil 2. İki Kare Matrisin Çarpma İşlemi İçin Çalışma Süreleri

Ölçümlerin ortalamasına göre yapılan farklı boyutlardaki matris işlemleri için Speedup değeri,

$$Speedup = \frac{\text{Seri Program Süresi}}{\text{Paralel Program Süresi}} \quad (3)$$

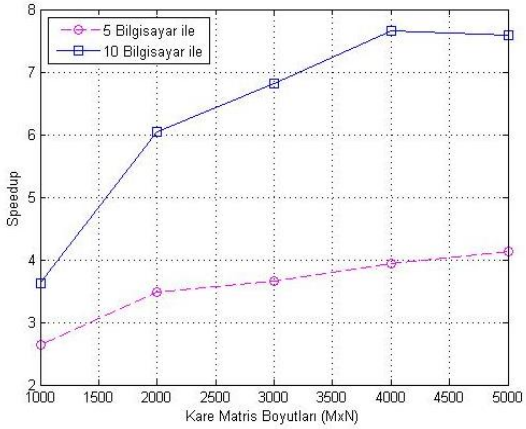
Eş. 3'te kullanılan paralel program süresi değeri, 5 veya 10 bilgisayarla yapılan ortalama program çalışma süresini ifade etmektedir. Eş. 3'te kullanılan seri program süresi ise tek bilgisayarla hesaplanan program çalışma süresidir.
Paralel programlama kullanılan hesaplamalarda çıkan değerlerin paralel çalışan bilgisayar sayısına yakın çıkması beklenir [25]. Tablo 6 da 5 bilgisayar ile yapılan paralel program çalışma süreleri göz önüne alındığında, artan matris boyutlarıyla orantılı olarak speedup değerleri artış göstermekte ve istenilen 5.0 değerine yaklaşmaktadır. Buda çok yüksek boyutlu matris çarpım hesaplamalarında paralel program tarafından elde edilen kazancın önemini göstermektedir.

5000x5000 boyutlu iki matrisin çarpım işlemi için program süreleri dikkate alındığında, önebellek kullanım optimizasyonu yapılmadan önce genel olarak kullanılan en yaygın algoritma (i→j→k) çalışma süresi 26318.1 dir.

Tablo 6. Değişen Bilgisayar Sayısı Ve Matris Boyutlarına Karşı Speedup Değerleri

Boyut (MxN)	Seri Program Çalışma Süresi (sec)	5 Bilgisayar İçin Paralel Prog. Süresi (sec)	5 Bilgisayar İçin Speedup	10 Bilgisayar İçin Paralel Prog. Süresi (sec)	10 Bilgisayar İçin Speedup
1000x1000	15.1295	5.72313	2.64	4.16892	3.63
2000x2000	119.932	34.3539	3.49	19.8085	6.05
3000x3000	402.785	110.152	3.66	59.0465	6.82
4000x4000	960.47	243.594	3.94	125.417	7.66
5000x5000	1857.6	449.205	4.14	245.159	7.58

Önbellek kullanım optimizasyonu sonrasında bu süre yaklaşık 14 kat kısalarak 1857.6 saniye olmuştur.



Şekil 3. İki Kare Matrisin Çarpma İşlemine Göre Speddup Değerleri

Daha sonra bu süre paralel programlama teknikleri ile önce 5 bilgisayar kullanılarak yapılan hesaplama, ilk hesaplama süresine oranla yaklaşık 59 kat kısalarak 449.205 saniye, sonra 10 bilgisayar kullanıldığında yaklaşık 107 kat daha hılı hale getirililerek 245.159 saniye olarak ölçülmüştür. En yavaş çalışan algoritma (k→j→i) 5000x5000 boyutlu iki matrisin çarpımında hesaplama süresi 61810.1 saniye sürmüştür. Önbellek kullanım optimizasyonu ve 10 bilgisayarla çalışan paralel programlama tekniği uygulandıktan sonra hesaplama süresi yaklaşık 252 kat kısalmıştır. Şekil 3 de görüldüğü üzere, 10 bilgisayar ile yapılan ölçüm işlemlerinde 4000x4000 boyutlu iki kare matrisin çarpımında speedup faktörü maksimum seviyeye ulaşmıştır.

SONUÇ

Tablo 1 da verilen kodlardaki en içteki for döngüsünde matrixAPtr matrisinin 1. elemanı matrixBPtr matrisinin 1. satır elemanlarıyla çarpılarak sonuç matrisinin (matrixCPtr) 1. satırının bir bölümünü hesaplanmaktadır. Ancak sonuç matrisinin ilgili kısmı tam olarak tamamlanmamaktadır. Her adımda bir kısmı tamamlanmakta ayrıca ortadaki for döngüsünde her bir değer değiştiğinde ön bellekte yalnızca bir matrisin aynı sütundaki tek elemanı önbelleğe getirilmektedir. Yani sadece bir eleman değişikliği olmaktadır. k,i,j ve i,k,j sıralamasına göre oluşturulan algoritmaların arasındaki fark ise, matrixAPtr matrisi ile matrixBPtr matrisinin yer değiştirmiş şekli olmasıdır. Yani cache belleğe taşınan verilerin birbirine yakın yöntemle taşınmasından kaynaklanmaktadır. Bu nedenle yakın sonuçlar elde edilmiştir.

Yapılan bu hesaplamalarda Tablo 2 de görüldüğü üzere cache belleğe getirilen bir verinin daha sonra tekrar tekrar Ram'den ön belleğe taşınmadan ya da olabildiğince daha az miktarda taşıma işlemi gerçekleştirilerek hesaplamaların tamamlanmasını sağlamak amaçlanmıştır. Matris işlemlerinde alışıla gelen (i→j→k) çarpma işlemi [26] yerine görünüşte insanlar için daha karmaşık olabilecek farklı mantıkta çarpma işleminin gerçekleştirilmesi ile cache bellek daha etkin kullanılmış ve işlem süresine olan olumlu ve olumsuz etkilerinin nasıl değiştiği gözlemlenmiştir. En iyi önbellek optimizasyonları daha az yürütme süresi ve daha az enerji tüketimi yapan çalışmalar olarak değerlendirilmektedir [27]. Bunun sonucunda enerji optimizasyonu transistörlerin yaşlanmasını doğrudan etkilemekte, bu da cihazların uzun vadeli güvenilirliği için bir sınırlayıcı faktör olmaktadır. Önceki çalışmalar hafıza sistemlerinde de enerji tasarrufunun yaşlanma etkilerini etkin bir şekilde kontrol edebildiğini ve önbellek ömrünü önemli ölçüde artırabileceğini göstermiştir [28, 29]. Çalışmada gerçekleştirilen etkin cache bellek optimizasyon yöntemleri ile önemli ölçüde işlem süreleri kısaltılmış ve ön bellek ve diğer işlem birimlerinin daha fazla kullanılmasının önüne geçilerek sistemin ömrünün uzatılması hedeflenmiştir. Ayrıca, bilgisayar performansında çok önemli yeri olan cache belleğin etkisinin önemi saptanmıştır.

Tablo 6 da görüldüğü üzere işlem yükü arttıkça paralel programlama hesaplama yönteminin hesaplama işlemine olan pozitif etkisi, speedup faktörünün bilgisayar sayısına yaklaşan eğilimi ile ortaya çıkmaktadır. Bu çalışmada hesaplamanın yanında haberleşmede kaybedilen vaktin öneminin fazla olduğundan 5000x5000 (M x N) boyutlu iki kare matrisin çarpımı işleminde daha düşük boyutlu çarpma işlemlerine göre speedup faktörünün azalma eğilimi

göstermeye başlamasının nedeni laboratuvarımızdaki switchlerin ve diğer ağ ekipmanlarının eski ve yavaş olmasıdır. Böylece artan iş yüküne rağmen bilgisayar sayısının artması haberleşmeden kaynaklanan gecikmeden dolayı oluşan kaybın önüne geçemediğinden speedup faktörü belirli bir iş yüküne karşı artarak devam ederken bir noktadan itibaren azalan eğilime geçmektedir. Bazı hesaplama işlemlerinde kullanılan çok daha büyük matris işlemleri, diğer hesaplama işlemleri ve analiz işlemlerinde paralel programlama işlemlerinin hesaplama performansına olan pozitif etkisi, bu çalışmada kendimizin hazırladığı algoritma ve kullandığımız dağıtık bellek modeline uygun data paralelliği doğru şekilde gerçekleşmesi ile bulunan sonuçlarda görülmüştür. Böylece yüksek iş yüküne sahip olan çok büyük boyutlu matris işlemlerinde uyguladığımız bu yöntem ile çok daha kısa sürede hesaplama işlemlerinin gerçekleştirilebileceği sonucuna varılmıştır.

ORCID

Eyüp Fahri Keskenler https://orcid.org/0000-0002-6762-856X

Mustafa Furkan Keskenler https://orcid.org/0000-0002-7604-4179

KAYNAKÇA

[1] Young C.Y. Precalculus. John Wiley Sons, 2010.
[2] Leise K.B.T. The linear algebra behind Google, SIAM Review, 48:3 569-581, 2006.
[3] Fudenberg D., Tirole J. Capital as a commitment: Strategic investment to deter mobility. Journal of Economic Theory, 31:2 227-250, 1983.
[4] Healy M. Matrices for Statistics, Oxford University Press, 1986.
[5] Godsil C.R. Gordon Algebraic Graph Theory, Springer-Verlag, 2004.
[6] Silillicorn D.B. Models for practical parallel computation, International Journal of Parallel Programming, 20:2 133-158. 1991
[7] Hillar G.C. Professional Parallel Programming with C#: Master Parallel Extensions with .NET 4, wrox, 2010.
[8] Moore,G. Cramming more components onto integrated circuits, Electronics, 38:8 1114-117, 1965.
[9] İşlemci-bellek başarımı farkı, in, acm.org, 19 May 2010.
[10] Gümüşkaya H. Mikroişlemciler ve Bilgisayarlar, Alfa Yayınları, 2011.
[11] SystemML Documents, in, systemml.apache.org/, 2016.
[12] Gu R., Tang Y., Tian C., Zhou H., Li G., Zheng X., Huang Y. Improving Execution Concurrency of Large-Scale Matrix Multiplication on Distributed Data-Parallel Platforms, IEEE Transactions On Parallel And Distributed Systems, 28 2539-2552, 2017
[13] Kart C., Matris Metodları ve Lineer Dönüşümler, Fen Fakültesi Yayınevi, Ankara, 1985.
[14] Chan K., Lam K.T., Wang C.L., Cache Affinity Optimization Techniques for Scaling Software Transactional Memory Systems on Multi-CMP Architectures, in: D. Grosu, H. Jin, G. Papadopoulos (Eds.) 14th International Symposium on Parallel and Distributed Computing, Ieee, New York, p. 56-65. 2015
[15] Goodman J.R. Cache Memory Optimization To Reduce Processor Memory Traffic, Journal of Vlsi and Computer Systems, 2 61-86, 1987
[16] Frank S.J. Tightly coupled multiprocessor system speeds memory-access times, Electronics, 57 164-169. 1984
[17] Isaak J. Squeezing the Most Out of the 68000, Mini-Micro Systems, 193-202, 1982.
[18] Goodman J. Computer Sciences Technical Report On Cache Memory Optimization to Reduce Processor/Memory Traffic, ACM Transactions on Computer Systems, 1-19, 1985.
[19] Mano M.M. Bilgisayar Sistemleri Mimarisi, Literatür Yayıncılık, 2015.
[20] Hatcher P.J., Quinn M.J. DataParallel Programming on MIMD Computers, MIT Press, 1993.
[21] Rose J., Steele G. Extended C Language for Data Parallel Programming, Technical Report PL Thinking Machines Inc. Cambridge MA. 1987.
[22] Rauber T., Rünger G. Tlib a library to support programming with hierarchical multi-processor tasks, J. Parallel and Distrib. Comput, 65 347-360, 2005.
[23] Kessler C., Keller J. Models for Parallel Computing Review and Perspectives, PARS-Mitteilungen, 24 13-29, 2007.
[24] Amdahl G.M. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities, in: In Proceedings of AFIPS spring joint computer conference, Atlantic City, NJ, USA, 483-485, 1967.
[25] Zuo W., McNeil A., Wetter M., Lee E.S. Acceleration of the matrix multiplication of Radiance three phase daylighting simulations with parallel computing on heterogeneous hardware of personal computer, Journal of Building Performance Simulation, 7 152-163, 2013
[26] Brown W.C. Matrices and vector spaces, New York, 1991.
[27] Alvarez J.D., Risco-Martin J.L., Colmenar J.M. Multi-objective optimization of energy consumption and execution time in a single level cache memory for embedded systems, Journal of Systems and Software, 111 200-212. 2016
[28] Cai Y., Schmitz M., Ejlali A., Al-Hashimi B., Reddy S.M. Cache size selection for performance, energy and reliability of time-constrained systems, in: Proceedings of the Asia and South Pacific Conference on Design Automation, p. 923-928, 2006.
[29] Mahmood H., Poncino M., Macii E. Cache aging reduction with improved performance using dynamically re-sizable cache, in: European Design and Automation Association, Proceedings of the Conference on Design, Automation Test in Europe 3001 Leuven, Belgium, p.174, 2014.