İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

İKİ BOYUTLU KARTEZYEN TOPOLOJİSİ KULLANILARAK CHEBYSHEV ÖN KOŞULLU CONJUGATE GRADIENT YÖNTEMİNİN PARALELLEŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Çağatay AKÇADOĞAN

Anabilim Dalı : Hesaplamalı Bilim ve Mühendislik

Programı : Hesaplamalı Bilim ve Mühendislik

OCAK 2010

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

İKİ BOYUTLU KARTEZYEN TOPOLOJİSİ KULLANILARAK CHEBYSHEV ÖN KOŞULLU CONJUGATE GRADIENT YÖNTEMİNİN PARALELLEŞTİRİLMESİ

YÜKSEK LİSANS TEZİ
Çağatay AKÇADOĞAN
(702011015)

Tezin Enstitüye Verildiği Tarih : 25 Aralık 2009
Tezin Savunulduğu Tarih : 29 Ocak 2010

Tez Danışmanı : Prof. Dr. Serdar ÇELEBİ (İTÜ)
Diğer Jüri Üyeleri : Prof. Dr. Hasan DAĞ (KHÜ)
Yrd. Doç. Dr. Lale ERGENE (İTÜ)

OCAK 2010

ÖNSÖZ

Bu çalışmada, Chebyshev ön koşullu conjugate gradient yönteminin paralel uygulaması incelenmiş olup, algoritmanın ölçeklenebilirliği ve performansı detaylı testlerle verilmiştir.

Çalışmam boyunca tecrübe, bilgi, yorum ve yardımlarıyla bana destek olan Prof.Dr. Serdar Çelebi'ye teşekkür ederim.

Aralık 2009 Çağatay AKÇADOĞAN

İÇİNDEKİLER

Sayfa

KISALTMALAR

PCG **:** Pre-conditioned Conjugate Gradient
CG **:** Conjugate Gradient
CPCG **:** Chebyshev Pre-conditioned Conjugate Gradient
MPI **:** Message Passing Interface
LU **:** Lower and Upper Triangle Decomposition
ILU **:** Incomplete Lower and Upper Triangle Decomposition
CPU **:** Central Procesing Unit
UYBHM **:** Ulusal Yüksek Başarımlı Hesaplama Merkezi

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

İKİ BOYUTLU KARTEZYEN TOPOLOJİSİ KULLANILARAK CHEBYSHEV ÖN KOŞULLU CONJUGATE GRADIENT YÖNTEMİNİN PARALELLEŞTİRLMESİ

ÖZET

$Ax = b$ yapısındaki lineer denklem takımlarının çözümü, sosyal bilimleri de kapsayan çalışmalar dahil olmak üzere hemen her bilim ve mühendislik probleminde gerekmektedir. Problem, lineer olmayan bir problemin yada diferansiyel denklem takımlarının çözümü aşamasında ortaya çıkabilir. Lineer denklem takımlarının çözümü için sıklıkla kullanılan yöntem LU ayrıştırmalı doğrudan çözüm yöntemleri olmuştur. Bu konu yeter seviyede olgun olup hakkında çok sayıda hazırlanmış yayın bulunmaktadır.

Çözüm için kullanılan diğer alternatif, yinelemeli yöntemlerdir. Bu kategoride, pozitif tanımlı lineer sistemlerin çözümü için yaygın olarak kullanılan yöntem CG' dır. LU ayrıştırmasına dayanan doğrudan çözüm yöntemleri yerine, yinelemeli yöntemlerin çeşitli hızlandırma teknikleri ile beraber özellikle paralel ortamlarda kullanılması büyük boyutlu lineer sistemlerin çözümü için tercih edilen bir yoldur. Hızlandırma, ön koşullama olarak adlandırılmaktadır. Çoğu zaman eldeki lineer sistemin katsayılar matrisinin tam olmayan ayrıştırması CG yöntemini hızlandırmak için kullanılır.

Bu çalışma, lineer denklem sistemlerinin katsayılar matrisinin yaklaşık tersini almakta kullanılan bir ön koşullayıcının CG yöntemi ile beraber paralel ortamda uygulanması ile elde edilen sonuçlarını sunmaktadır. Ön koşullayıcı, matris değerli Chebyshev polinomlarının lineer kombinasyonundan elde edilmektedir. Paralel hesaplama açısından doğrudan çözüm yöntemlerinin ve LU tipinde ön koşullayıcılar kullanan yinelemeli yöntemlerin paralelleştirilmesinde ciddi sınırlamalar bulunurken Chebyshev ön koşullayıcı paralel işleme oldukça yatkındır.

Bu çalışmada ilk olarak yinelemeli yöntemlere, lineer denklem sistemlerinin çözümünde kullanılan temel yaklaşımlar tanıtılarak kısa bir giriş yapılmaktadır. Önerilen yöntem teorik olarak tanıtılırken, yapılan çeşitli testlerin sonuçları yöntemin verimliliğini betimlemek için verilmektedir. Bunun ardından algoritma seviyesinden koda dönüştürülen yöntem paralel programlamada kullanılan ileri programlama teknikleri aracılığı ile optimize edilerek mevcut platformda testleri gerçekleştirilmektedir. Elde edilen sonuçlar, çizelge ve şekillerle gerekli yorum ve karşılaştırmalar da eklenerek sunulmaktadır.

PARALLELIZATION OF CHEBYSHEV PRECONDITIONED CONJUGATE GRADIENT METHOD USING TWO DIMENSIONAL CARTESIAN TOPOLOGY

SUMMARY

Solution of linear set of equations, $Ax = b$, is indispensable in almost every science and enginerring problem, including social studies. The problem may arise as part of the solution phase of a non-linear problem or solution phase of a set of differential equations. The usual practice for the solution of linear sets of equations has been LU factorization based direct methods. The subject is mature enough and there are enormous amount of publications on the subject.

The other alternative for the solution is iterative methods. In this category, for symmetric positive definite linear systems a widely used iterative method is CGM. Use of iterative methods in conjuction with some acceleration techniques, instead of LU factorization based direct methods is now a preferred way of solving large scale linear equations, especially in parallel environments. The acceleration is called preconditioning. Most of the time an incomplete LU (ILU) factorization of the coefficient matrix of the linear system at hand is used to accelerate CGM.

This work presents the results of an implementation of CGM in a parallel environment with a newly published preconditioner, an approximate inverse of coefficient matrix of linear equations to be solved. Preconditioner is obtained from a linear combination of matrix-valued Chebyshev polynomials. On the parallel computation aspect, there are serious limitations in parallelizing the direct solution methods and iterative methods that use LU type preconditioners while Chebyshev preconditioner is considerably amenable to parallel processing.

In this work, initially, a brief introduction to iterative methods is presented designating the principal approaches to the solution of linear systems. The proposed method is introduced theoretically while several test results are given in order to show the effectiveness of Chebyshev preconditioner. After that, test environment and necessary tools to implement the proposed method are introduced in detail. Advanced programming techniques in parallel environment are stated as depending on programming interface. As a conclusion, all test results are presented in corresponding tables and figures by adding comments and comparisons.

1. GİRİŞ

$Ax = b$ yapısındaki lineer denklem takımlarının çözümü, sosyal bilimleri de kapsayan çalışmalar dahil olmak üzere hemen her bilim ve mühendislik probleminde gerekmektedir. Problem, lineer olmayan bir problemin yada diferansiyel denklem takımlarının çözümü aşamasında ortaya çıkabilir. Lineer denklem takımlarının çözümü için sıklıkla kullanılan yöntem LU ayrıştırmalı doğrudan çözüm yöntemleri olmuştur. Bu konu yeter seviyede olgun olup hakkında çok sayıda hazırlanmış yayın bulunmaktadır [1].

Çözüm için kullanılan diğer alternatif, yinelemeli yöntemlerdir. Bu kategoride iki sınıflandırma vardır; Gauss-Siedel, Jacobi ve SOR vb. gibi nokta (durağan) yöntemler ve bu çalışmada ilgilenilen Krylov alt uzayı temelli azaltma yöntemleri. Krylov alt uzayı temelli yinelemeli yöntemlerde kendi içinde iki ayrı durum için analiz edilebilir; simetrik pozitif tanımlı lineer sistemler ve simetrik olmayan muhtemelen tanımsız genel lineer denklem sistemleri. İlki için en yaygın kullanıma sahip olan yöntem CG olup [2,3] ikincisi içi sıklıkla tercih edilen yöntem GMRES' dir [4].

CG yöntemini kullanmanın hesaplama süresini büyük boyutlu problemler için, LU ayrıştırmasına dayalı doğrudan çözüm yöntemlerine oranla azalttığı çeşitli çalışmalarda rapor edilmiştir [5,6]. Dolayısıyla, LU ayrıştırmasına dayanan doğrudan yöntemler yerine, yinelemeli yöntemlerin çeşitli hızlandırma teknikleri ile beraber özellikle paralel ortamlarda kullanılması büyük boyutlu lineer sistemlerin çözümü için tercih edilen bir yoldur. Hızlandırma, ön koşullama olarak adlandırılmaktadır. Çoğu zaman eldeki lineer sistemin katsayılar matrisinin tam olmayan ayrıştırması CG yöntemini hızlandırmak için kullanılmaktadır [7,8]. Ön koşullama, katsayılar matrisinin öz değer gruplarının indirgenmesi olarak kabul edilebilir. Bununla birlikte, hızlandırmanın verimliliğini artırmak için katsayılar matrisinin yeniden düzenlenmesi gerekebilir. Yapılan çeşitli çalışmalar yeniden düzenlemenin CG yönteminin yakınsama hızını etkilediğini onaylamaktadır [9,10].

Bu çalışma, çözülecek lineer denklemlerin katsayılar matrisinin yaklaşık tersini almakta kullanılan bir ön koşullayıcının, CG yöntemi ile beraber paralel ortamda uygulanması ile elde edilen sonuçları sunmaktadır. Ön koşullayıcı, matris değerli Chebyshev polinomlarının lineer kombinasyonundan elde edilmektedir [1]. Paralel hesaplama açısından doğrudan çözüm yöntemlerinin ve LU tipinde ön koşullayıcılar kullanan yinelemeli yöntemlerin paralelleştirilmesinde ciddi sınırlamalar bulunmaktadır. Paralelleştirmedeki bu kısıtlamalar ardışık şekilde gerçekleştirilen ileri ve geri yerine koymalar ve lineer denklemlerin yeniden düzenlenmesinden kaynaklanmaktadır. Dolayısıyla bu noktada iyi bir ön koşullayıcının sahip olması gereken özellikleri sıralamak gerekirse

- Oluşturulması ve uygulaması kolay olmalı.
- Katsayılar matrisinin spektrumunu, kullanılacak yöntemin yineleme sayısını azaltacak şekilde değiştirmeli.
- Paralel hesaplamaya uygun olamalı.
- Önkoşullayıcının hesaplama maliyeti mümkün olan en düşük seviyede kalamalı.

Yapılan bu tanımlamalara Chebyshev ön koşullu CG yöntemi tam olarak karşılık verebilmektedir çünkü ön koşullayıcının hesaplanmasında yerine koyma yada matris düzenlenmesi kullanılmamaktadır. Bu bağlamda yöntemin tanıtımı için, ilk olarak yinelemeli yöntemlere, lineer denklem sistemlerinin çözümünde kullanılan temel yaklaşımlar tanıtılarak kısa bir giriş yapılmaktadır. Önerilen yöntem teorik olarak tanıtılırken yapılan çeşitli testlerin sonuçları, yöntemin verimliliğini göstermek için verilmektedir. Bunun ardından, mevcut test ortamında ileri paralel programlama tekniklerine uygun olarak yapılan kodlama, sistem topolojisinin belirlenmesi ve kodun sistem üzerindeki performansını artırmaya yönelik uygulanan optimizasyonlar, farklı problem boyutları için detaylı ve karşılaştırmalı olarak verilmektedir.

2. YİNELEMELİ YÖNTEMLER

Yinelemeli Yöntem terimi, bir lineer denklem sisteminin her bir adımda daha doğru çözümünü elde etmek için kullanılan ardışık yaklaşım tekniklerinin tümüne karşılık gelmektedir. En genel kapsamı ile iki ana başlık altında toplanabilir: durağan yöntemler daha eski anlaşılması ve uygulaması daha kolay olmak ile birlikte verimliliği düşük olan yöntemler sınıfını oluşturmaktadır. Durağan olmayan yöntemler birincisine göre daha yeni olup analizleri açısından daha zor ama performansları ve elde edilen sonuçlar açısından daha tutarlıdırlar.

Bir yinelemeli yöntemin yakınsama oranı büyük ölçüde katsayılar matrisinin spekturumuna (öz değerlerinin dağılımı) bağlıdır. Bu yüzden yinelemeli yöntemler, katsayılar matrisinin spektrumunu çözüm için daha uygun bir yapıya dönüştürecek dönüşüm (transformasyon) matrisleri kullanırlar. Bu transformasyon matrislerine ön koşullayıcı adı verilir. İyi bir ön koşullayıcı yinelemeli yöntemin yakınsamasını iyileştirmeli, oluşturma ve uygulama aşamalarında yapılacak işlemler açısından yüksek maliyetli olmamalıdır. Bir ön koşullayıcı kullanmaksızın yinelemeli yöntemlerin yakınsamaması kuvvetli bir ihtimaldir [11,12,13,14].

2.1 Yöntemlere Genel Bakış

Yaygın olarak kullanılan yinelemeli yöntemler aşağıdaki şekilde verilebilir

- Durağan yöntemler,
    - **Jacobi** yöntemi her bir değişkeni yerel olarak diğer değişkenlere gore çözme mantığına dayanmaktadır. Yöntemin her bir yinelemesi her bir değişken için bir kez çözüm yapmayı gerektirmektedir. Yöntem anlaşılması ve uygulaması bakımından kolay olmakla birlikte yakınsaması yavaştır.
    - **Gauss-Seidel** yöntemi Jacobi yöntemine benzemekle birlikte tek farkı sürekli olarak güncellenmiş değerleri kullanmasıdır. Eğer Jacobi yakınsıyor ise, Gauss-Seidel yöntemi Jacobi'den hızlı yakınsayacaktır.

- **SOR** yöntemi bir dış değerleme (ekstrapolasyon) (ω) parametresi tanımlanarak Gauss-Seidel metodundan türetilebilir. Bu parametrenin optimal seçilmesi durumunda SOR Gauss-Seidel'dan daha hızlı yakınsayabilir.
- **SSOR** yönteminin SOR'a bir üstünlüğü olmamakla birlikte durağan olayan yöntemlerde önkoşullayıcı olarak kullanılabilir.

- Durağan olmayan yöntemler,
  - **Conjugate Gradient** (eşlenik gradyan) yönteminin ismi, yöntemin ardışık olarak eşlenik (yada dikey) vector dizileri üretmesinden gelmektedir. Bu vektörler yinelemelerin rezidülerini (farklarını) oluşturmaktadır. Aynı zamanda bu vektörler minimizasyonu lineer sistemin çözümüne eşit olan, ikinci derecede bir fonksiyonun gradyanlarını teşkil etmektedir. Conjugate Gradient katsayılar matrisinin pozitif tanımlı olması durumunda, yanlızca sınırlı sayıda vektörün saklanması ve kullanılmasını gerektireceğinden oldukça verimli bir yöntemdir.
  - **Minimum Residual (MINRES) ve Simetrik LQ (SYMMLQ)** yöntemleri simetrik ama tanımsız katsayılar matrisi durumu için CG metoduna alternatif olarak görülebilir. SYMMLQ, eğer katsayılar matrisi pozitif tanımlı ise CG ile aynı yineleme çözümlerini üretecektir.
  - **Normal Denklemler için Conjugate Gradient (CGNE ve CGNR)** yöntemlerinden, CGNE, $(AA^T)y = b$ sistemini y için çözer ve $x = A^T y$ için çözümü hesaplar. CGNR, $(A^T A)x = \check{b}$ sistemini çözüm vektörü x için $\check{b} = A^T b$ koşulu altında çözer. Eğer katsayılar matrisi A simetrik ve tekil değil ise normal denklem matrisleri AA^T ve $A^T A$ pozitif tanımlı olacaktır ve CG yöntemi uygulanabilecektir. Normal denklem matrislerinin spektrumu, A matrisininkinden yöntemin işlerliği açısından daha kötü olabileceğinden yakınsamada yavaş olabilir.
  - **Genelleştirilmiş Minimal Rezidü (GMRES)** yöntemi, MINRES' de olduğu gibi bir dikey vektörler dizisi hesaplar ve en küçük kareler

yardımıyla bunları bir araya getirip çözümü ve gerekli güncellemeleri gerçekleştirir. Bununla birlikte MINRES (ve CG)' den farklı olarak tüm vektör dizisinin saklanmasını gerektirir ve buda büyük miktarda depolama alanı kullanılmasını zorunlu kılar. Bu sebepten ötürü GMRES yönteminin tekrar başlatmalı versiyonu tercih edilir. Tekrar başlatmalı versiyonunda, hesaplama ve depolama giderleri üretilecek vektörlerin sayısı önceden belirlenen bir değerde sabitlenerek azaltılır. Bu yöntem özellikle genel simetrik olmayan matrisler için oldukça kullanışlıdır.

- **BiConjugate Gradient (BiCG)** yöntemi, biri orjinal katsayılar matrisi A diğeri de A^T' ye bağlı olmak üzere iki ayrı CG vektör dizisi oluşturur. Her bir vektör dizisini kendi içinde dikeyleştirmek yerine, diziler karşılıklı olarak birbirleriyle dikleştirilirler. Bu yöntem CG' da olduğu gibi sınırlı veri depolamasına ihtiyaç duymaktadır. Özellikle katsayılar matrisinin simetrik ve tekil olmadığı durumlarda bu yöntem faydalı olabilir. Bununla birlikte yakınsamanın düzensiz olması ve yöntemin beklenen sonuca ulaşmadan durma ihtimali vardır. BiCG yöntemi her bir yinelemede katsayılar matrisi ve devriği ile çarpma işlemi gerçekleştirir.
- **Quasi Minimal Rezidü (QMR)** yöntemi, BiCG rezidülerine en küçük kareler çözümü uygular ve böylece BiCG yönteminin başarısızlığa uğramasını engelleyebilir. Diğer taraftan hata ve rezidü değerlerinde herhangi bir minimizasyon yapmaz yani çoğu zaman BiCG yönteminin iterasyon sayısında bir azalmaya dolayısıyla bu yöntemin hızlı çalışmasına bir katkısı olamayacaktır.
- **Kareselleştirilmiş Conjugate Gradient (CGS)** yöntemi, A ve A^T matrisleri için kullanılan güncelleyici işlemleri aynı vektörlere uyugulayan BiCG yönteminin değiştirilmiş bir şeklidir. . İlk bakışta bu yaklaşım yakınsama oranını ikiye katlıyormuş gibi görülebilir ama pratikte yakınsama BiCG metodunda olduğundan daha az güvenilir olabilir. Bu yöntemin uygulamadaki en büyük avantajı katsayılar matsiri ile devriğinin çarpımını gerektirmemesidir.

- **Kararlılaştırılmış BiConjugate Gradient (Bi-CGSTAB)** yöntemi, CGS' de olduğu gibi BiCG yönteminin değiştirilmiş bir halidir ama CGS yönteminden daha iyi bir yakınsama elde etmek için A^T için farklı güncellemeler kullanılır [11,13,14].

2.2 Yöntemlerin Hesaplama Analizleri

Bir lineer sistemin verimli çözümü doğrudan, seçilecek yinelemeli yöntemin uygunluğuna bağlıdır. Bununla birlikte, varsayımlar ve hesaplamada kullanılacak araçlar ve bunların hesaplama yapılacak programlama yöntemlerine uygunluğu da oldukça önemlidir. Bu nokta özellikle paralel hesaplama açısından ayrı bir öneme sahiptir [11,12].

Bu özellikler dikkate alındığında yinelemeli yöntemlerin doğrudan çözüm yöntemlerinden farklı olduğu ortaya çıkar. Doğrudan yöntemlerin performansı büyük ölçüde yapılacak matris ayrıştırmasına bağlıdır. Bu işleme yinelemeli yöntemlerde ihtiyaç duyulmamakta (ön koşullayıcıların oluşturulması safhası dışında) ve buna bağlı olarak matrislerler yapılan pek çok alt işleme gerek kalmamaktadır. Bu işlemler günümüz bilgisayar platformlarında yüksek performanslar ile gerçekleştirildiğinden, doğrudan yöntemlerden daha düşük flop oranlarını yinelemeli yöntemler için beklemek yanlış olacaktır (Dongarra ve Van der Vost [15] bu konuyla ilgili pek çok deneysel sonuç vermektedir).

Bunlara ilave olarak, yinelemeli yöntemler uygulama açısından doğrudan yöntemlerden daha kolay oldukları için ve depolanması zorunlu olan tam bir matris ayrıştırması gerektirmediklerinden doğrudan yöntemlerde çalışılan sistemlerden daha büyükleri ile çalışabilirler. Seçilecek yinelemeli yöntem için göz önünde tutulması gereken önemli hususlar aşağıdaki şekilde iki ana başlık altında toplanabilir;

- Matris özellikleri: Tanımlanan yöntemlerin hepsi bütün problem tipleri için çalışmazlar dolayısıyla matrisin özellikleri yinelemeli yöntem seçimi için ana kriter olmalıdır.
- Hesaplama teknikleri: Farklı yöntemler probleme bağlı farklı hesaplama teknikleri ve üzerinde çalışılacak mimariye uygun tercihler gerektirir.

Çizelge 2.1: *i*. yineleme için yapılan işlemlerin özeti**.**

Yöntem	İç Çarpım	SAXPY	Matris Vektör Çarpımı	Önkoşullayıcı Çözümü
JACOBI			1[a]	
GS		1	1[a]	
SOR		1	1[a]	
CGM	2	3	1	1
GMRES	i+1	i+1	1	1
BiCG	2	5	1/1	1/1
QMR	2	8+4[b]	1/1	1/1
CGS	2	6	2	2
Bi-CGSTAB	4	6	2	2

Çizelge 2.1 tanımlaması yapılan yöntemlerin her bir yineleme adımında gerek duyduğu hesaplamaların özetini vermektedir. *"x/y"* ifadesinde *"x"* matris ile yapılan çarpımı *"y"* matrisin kendi transpozu ile yapılan çarpımı ifade etmektedir. Çizelge'de kullanılan *"a"* o yöntemin gerçek bir matris-vektör çarpımı gerçekleştirmediği yada önkoşullayıcı çözümü yapmadığı ama yapılan işlem sayısının bir matris-vektör çarpımına eşit olduğunu ifade eder. *"b"* ilgili yöntemin gerçek SAXPY işlemleri ile birlikte vektör ölçeklemeleri gerçekleştirdiğini belirtmektedir.

Yapılan bu tespitler ışığında her bir yöntemin hesaplama karakteristiğini aşağıdaki şekilde ifade edebiliriz;

1. Jacobi Yöntemi
    a. Kullanılması çok kolay olmak ile birlikte eğer matris köşegeni baskın değil ise bu yöntemin yinelemeli yöntemlere bir giriş yada durağan olmayan yöntemler için bir ön koşullayıcı olarak kabul etmek yerinde olacaktır.
    b. Paralelleşitirilebilmesi önemsizdir.
2. Gauss-Seidel Yöntemi
    a. Jacobi yönteminden hızlı durağan olmayan yöntemlere göre yavaştır.
    b. Köşegen baskın yada pozitif tanımlı sistemlere uygulanabilir.
    c. Paralleştirme özellikleri katsayılar matrisinin yapısına bağlıdır. Bilinmeyenlerin farklı şekillerde düzenlenmesi paralelleştirmenin

verimini artırabilir. Çok gruplu düzenlemeler tam bir paralelleştirme sağlayabilir.

d. SOR yönteminde ekstrapolasyon (ω) değeri 1 seçilerek elde edilebilen özel bir durumdur.

3. SOR

a. Gauss-Seidel'ın yakınsamasını hızlandırır ($\omega > 1$) ve yine Gauss-Seidel'ın başarısızlığa uğradığı durumlarda yakınsayabilir ($0 < \omega < 1$).

b. Yakınsama hızı ω' ya bağlıdır. ω için seçilecek optimum değer belli koşullar altında Jacobi iterasyon matrisinin spektral yarı çapından tahmin edilebilir.

c. Paralelleştirme özellikleri Gauss-Seidel ile aynıdır.

4. Conjugate Gradient (CG)

a. Simetrik pozitif tanımlı sistemlere uygulanabilir.

b. Yakınsama hızı matrisin koşul sayısına bağlıdır; eğer baskın öz değerler birbirlerinden yeterince iyi ayrılmışlarsa süper doğrusal yakınsama elde edilebilir.

c. Paralel ortamda iç çarpımlar sekronizasyon noktaları olarak görülebilir.

d. Paralel özellikleri katsayılar matrisinden bağımsızdır ve ön koşullayıcının yapısına bağlıdır.

5. GMRES

a. Simetrik olmayan matrislere uygulanabilir.

b. GMRES sabit sayıdaki yineleme adımı sayısı için en küçük rezidüyü elde edebilir ama bu adımlar hesaplama açısından oldukça pahalıdır.

c. Her bir yineleme için artan depolama ihtiyacı ve hesaplama işlerini azaltmak için tekrar başlatma zorunludur. Bunun yapmak *A* matrisi ve sağ yan vektörüne bağlıdır ve yapılacak düzenlemeler yetenek ile deneyimi zorunlu kılar.

d. GMRES yanlızca katsayılar matrisi ile vektör çarpımını gerektirir.

e. İç çarpılmarın sayısı yeniden başlatma noktasına kadar yineleme sayısıyla doğrusal olarak artar.

6. BiConjugate Gradient (BiCG)

   a. Simetrik olmayan matrislere uygulanabilir.

   b. Katsayılar matrisi ve transpozu ile vektör çarpımlarını gerektirir. Bu, yöntemin en büyük handikaplarından biridir çünkü matrisin bir operatör olarak verildiği durumlarda uygun bir transpoz operatörü bulmak mümkün olmayabilir.

   c. Paralelleştirme özellikleri CG yönteminkine benzerdir; iki matris-vektör çarpımı (ve ön koşullama adımları) bağımsızdır ve kolaylıkla paralelleştirilebilir.

7. QMR

   a. Simetrik olmayan matrislere uygulanabilir.

   b. BiCG yönteminin düzensiz yakınsama ve bozulma davranışlarını engellemek maksadı ile dizayn edilmiştir.

   c. Eğer BiCG bir adımda belirgin işlem yapıyor ise QMR' da aynı sonuçları aynı adımda elde eder. Ama BiCG geçici olarak durduğu yada ıraksadığı zaman QMR her ne kadar yavaş olmakla birlikte, rezidüleri azaltmayı sürdürebilir.

   d. Her bir adımdaki hesaplama maliyeti BiCG ile benzer olmakla beraber az da olsa farklılık göstermektedir, matrisin transpozu ile vektör çarpımını gerektirir.

   e. Paralelleştirme özellikleri BiCG ile aynıdır.

8. Kareselleştirilmiş Conjugate Gradient (CGS)

   a. Simetrik olmayan matrislere uygulanabilir.

   b. Yakınsama (yada ıraksama) BiCG' den yaklaşık iki kat daha hızlıdır.

   c. Yakınsama davranışı çok defa, güncellenmiş rezidülerde doğruluk kaybına yol açabilecek derecede düzensiz olabilir.

d. Her bir yinelemedeki hesaplama maliyetleri BiCG ile benzer olmakla birlikte yöntem matris transpozu gerektirmez.

e. BiCG'den farklı olarak, iki matris-vektör çarpımı bağmsız değildir dolayısıyla parallel ortamdaki sekronizasyon noktası sayısı daha büyüktür.

9. Kararlaştırılmış BiCG (Bi-CGSTAB)

   a. Simetrik olmayan matrislere uygulanabilir.

   b. Her bir yinelemedeki hesaplama maliyeti BiCG ve CGS ile benzer olmakla beraber yöntem matris transpozu gerektirmez.

   c. CGS' nin düzensiz yakınsama yapısından kaçınırken yakınsama hızını koruyan alternatif bir yöntemdir. Bununla birlikte çoğu defa güncellenmiş rezidülerde doğruluk kaybıyla karşılaşmak mümkündür.

Çizelge 2.2 : *i.* yineleme için depolama gereksinimleri.

Yöntem	Depolama Gereksinimi
JACOBI	matris+3n
SOR	matris+2n
CGM	matris+6n
GMRES	matris+(i+5)n
BiCG	matris+10n
CGS	matris+11n
Bi-CGSTAB	matris+10n
QMR	matris+16n

Çizelge 2.2 her bir yöntem için ihtiyaç duyulan depolama miktarını vermektedir (ön koşullayıcı olmaksızın). Bu çizelgeye orjinal sistem $Ax = b$' nin skalerlerin depolanması eklenmiştir. *n* matrisin boyutuna karşılık gelmektedir [11].

2.3 Conjugate Gradient Yöntemi (CG)

Conjugate Gradient yöntemi en eski ve en iyi bilinen, pozitif tanımlı doğrusal simetrik sistemlerde verimli olabilen durağan olmayan yinelemeli bir yöntemdir. Yöntem, yinelemelerin vektör dizilerini (çözüme ardışık yaklaşım), yinelemelerdeki rezidüleri ve yinelemeleri güncellemekte kullanılan arama doğrultularını üreterek ilerler. Bu dizilerin uzunluklarının büyük olmasına rağmen az sayıda vektörün bellekte tutulması yeterli olacaktır [2,3,11,12].

2.3.1 Krylov Alt Uzayı

Basit yinelemeli yöntemler her bir yineleme adımında aşağıdaki öz yineleme formunu kullanırlar;

$$x_{i+1} = x_i + M^{-1}(b - Ax_i) = x_i + M^{-1}r_i \quad \textbf{(2.1)}$$

İlk adımdan başlayarak

x_0

$x_1 = x_0 + (M^{-1}r_0)$

$x_2 = x_1 + (M^{-1}r_1) = x_0 + M^{-1}r^0 + M^{-1}(b - Ax_0 - AM^{-1}r_0)$

$= x_0 + 2M^{-1}r_0 - M^{-1}AM^{-1}r_0$

$\vdots$

Bu ifade

$$\boldsymbol{x_1} \in \boldsymbol{x_0} + span\{M^{-1}r_0, M^{-1}A(M^{-1}r_0) \dots (M^{-1}A)^{i-1}(M^{-1}r_0)\} \quad \textbf{(2.2)}$$

olarak tanımlanabilir. $K^i(A; r_0) \coloneqq span\{r_0, Ar_0 \dots A^{i-1}r_0\}$ alt uzayı A matrisi ve başlangıç rezidüsü r_0' a karşılık gelen i boyutlu Krylov uzayı olarak adlandırılır. Her hangi bir yinelemeli yöntem ile hesaplanan x_i, $x_0 + K^i(M^{-1}A; M^{-1}r_0)$' nin elemanıdır [11,12].

2.3.2 CG Yöntemi Kuramı

Eğer A matrisi $A = A^T$ ve $x > 0$ için $x^TAx > 0$ şartlarını sağlıyor ise simetrik pozitif tanımlı matris olarak adlandırılır. Bu şart CG yönteminin türetilmesi ve çalışması açısından en önemli özelliktir.

İlk yaklaşım olarak $\|x - x_i\|_2$ minimize edecek $x_i \in K^i(A, r_0)$ vektörü seçilsin. İlk yinelemedeki x_1 vektörü, $\|x - x_1\|_2$ minimum yapmak kaydıyla seçilecek bir α_0 sabiti ile $x_1 = \alpha_0 r_0$ yapısında yazılabilir. Bu da

$$\|x - x_1\|_2^2 = (x - \alpha_0 r_0)^T (x - \alpha_0 r_0) = x^T x - 2\alpha_0 r_0^T x + \alpha_0^2 r_0^T r_0 \quad \textbf{(2.3)}$$

olarak tanımlanabilir. Eğer $\alpha_0 = r_0^T x / r_0^T r_0$ olursa, (2.3)' de verilen norm minimize edilecektir. Ama x bilinmediği için bu seçim belirlenemeyecek dolayısıyla bu yaklaşım faydalı bir yöntem olmaktan uzak kalacaktır. Bununla birlikte $Ax = b$ bilindiğinden dolayı A' ya sahip bir iç çarpım kullanmak α_0 hesaplamaya yardımcı olacaktır.

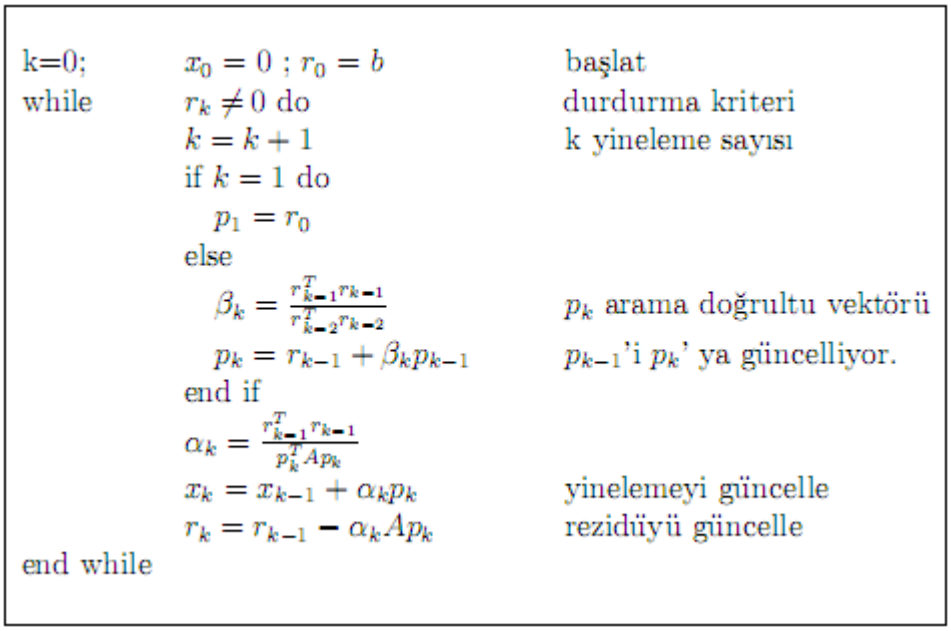
A iç çarpım $(y, z)_A = y^T A z$ ve A norm da $\|y\|_A = \sqrt{(y, y)_A} = \sqrt{y^T A y}$ şeklinde tanımlansın. Yukarıda belirtildiği üzere A matrisi pozitif simetrik tanımlı olmanın şartlarını sağlıyor ise $(\ldots)_A$ ve $\|.\|_A$ iç çarpım ve norm ile ilgili tüm özellikleri sağlayacaktır. $\|x - x_1\|_2$ minimum yapacak x_1 elde etmek için

$$\|x - x_1\|_A^2 = x^T A x - 2\alpha_0 r_0^T A x + \alpha_0^2 r_0^T A r_0 \quad \textbf{(2.4)}$$

$\alpha_0 = r_0^T A x / r_0^T A r_0 = r_0^T b / r_0^T A r_0$ seçilmelidir. Buradan görüleceği gibi yeniden tanımlanan iç çarpım kolaylıkla çözülebilecek bir minimizasyon problem ortaya koymuştur. Sonraki iterasyonlarda hesaplanacak x_i

$$\|x - x_i\|_A = \min_{y \subset K^i(A; r_0)} \|x - y\|_A \quad \textbf{(2.5)}$$

formundan yararlanılarak elde edilir. Bu minimizasyon probleminin çözümü Conjugate Gradient yönteminin kendisini teşkil eder. Şekil 2.1' de görüldüğü gibi A matrisi, x_k ve r_k vektörlerine ek olarak p_k vektörünün bellekte saklanması yeterli olacaktır. Buna ilave olarak bir önceki yinelemede kullanılan vektörlerin üzerine ondan sonra gelen yinelemede yazılabilir. Bu algoritmada r_k değeri r_{k-1} den $r_k = r_{k-1} - \alpha_k A p_k$ denklemi kullanılarak hesaplanıyor. Bunun sebebi $r_k = b - A x_k$ hesaplayarak fazladan bir matris vektör çarpımına engel olmak. Bazı uygulamalarda CG algoritmasından elde edilen güncellenmiş rezidü gerçek değeri olan $b - A x_k$ ifadesinden yuvarlama hatalarından dolayı bir hayli sapabilmektedir. Bundan dolayı güncellenmiş rezidü için durma kriteri sağlandıktan sonra gerçek ve güncellenmiş rezidü normlarının karşılaştırılmasında fayda vardır. Bu durumda eğer gerçek rezidü, durma kriterini sağlamaz ise CG yönteminde bulunan x_k başlangıç vektörü olarak seçilerek yeniden başlatılmalıdır.



Şekil 2.1: Conjugate Gradient yönteminin algoritması.

Teoride CG yöntemi sonlu bir yöntemdir. n. yinelemeden sonra Krylov alt uzayı $\Re^n$ eşit olacaktır. $\|x - y\|_A$ ifadesi $K^n(A; r_0) = \Re^n$ üzerinde minimize edildiğinden norm sıfıra eşit $x_0 = x$ olacaktır. Bununla birlikte pratikte bu özellik iki sebepten dolayı kullanılmaz; birincisi pek çok uygulamada n çok büyük olduğundan n yineleme yapmak uygun değildir, ikinci olarak n küçük bile olsa yuvarlama hataları vektörlerin hatalı hesaplanamasına neden olabilir.

(2.5)' den ve $K^k(A; r_0) \subset K^{k+1}(A; r^0)$ ifadesinden anlaşılabileceği gibi $\|x - x_k\|_A$ dizisi azalan bir dizidir ve $\|x - x_{k+1}\|_A \leq \|x - x_k\|_A$ biçiminde düşünmek yanlış olmayacaktır. Pratikte, x bilinmediği için $\|x - x_k\|_A$ hesaplamak kolay değildir. Rezidü normu $\|r_k\|_2 = \|x - x_k\|_{A^T A}$ olarak verilir ve bu dizinin azalan olması gerekmez. Uygulamada $\|r_{k+1}\|_2$ değeri $\|r_k\|_2$' den daha büyük olabilir.

Bu CG yönteminin ıraksadığı anlamına gelmez. $\|r_k\|_2 = \|Ax_k - b\|_2 \leq \sqrt{\|A\|_2}\|x - x_k\|_A$ eşitsizliği $\|r_k\|_2$ ' nin $\sqrt{\|A\|_2}\|x - x_k\|_A$ azalan dizisinden daha küçük olduğunu gösterir dolayısıyla belli bir yineleme sayısından sonra rezidü normu tekrar azalacaktır.

Şekil 2.1' e bakıldığında β_k ve α_k' yı hesaplayan iki oran görülmektedir. Eğer bunlardan birisinin paydası sıfır'a eşit ise CG yöntemi duracaktır. β_k paydasının sıfır olması durumu için $r_{k-2}^T r_{k-2} = 0$ dolayısıyla $r_{k-2} = 0$ ve $x_{k-2} = x$ olacaktır bu da

lineer sistemin çözüldüğü anlamına gelir. α_k paydası $p_k^T A p_k = 0$ dolayısıyla $p_k = 0$ durumunda sıfır olur. $span\{p_1, \dots, p_k\} = span\{r_0, \dots, r_{k-1}\} = K^k(A; r_0)$ olduğu göz önüne alınırsa $r_{k-1} = 0$ olacak dolayısıyla sistem tekrar çözüme ulaşmış olacaktır [2,3,11,12].

2.3.3 CG Yönteminin Yakınsama Analizi

Yinelemeli yöntemlerin yakınsamaları ile ilgili doğru tahminler yapabilmek için çok zor olmakla birlikte, kullanışlı sınır değerler elde etmek mümkündür. CG yöntemi için, hata değerinin sınırları A matrisinin spektral koşul sayısı κ_2 kullanılarak belirlenebilir. Eğer λ_{max} ve λ_{min} simetrik pozitif tanımlı A matrisinin sırasıyla en büyük ve en küçük öz değerleri ise A' nın spektral koşul sayısı $\kappa_2(A) = \lambda_{max}(A)/\lambda_{min}(A)$ olacaktır. Eğer $Ax = b$ doğrusal sisteminin tam çözümü x ise $\alpha = (\sqrt{\kappa_2} - 1)/(\sqrt{\kappa_2} + 1)$ olmak üzere;

$$\|x_i - x\|_A \leq 2\alpha^i \|x_0 - x\|_A \qquad \textbf{(2.6)}$$

şeklinde ifade edilebilir. **(2.6)**' dan görülebileceği gibi ε hata değerini azaltmak için yapılacak yineleme sayısı $\sqrt{\kappa_2}$ ile orantılıdır. Diğer taraftan eğer A matrisinin baskın öz değerleri birbirlerinden yeteri derecede ayrık ise super lineer yakınsama elde edilebilir, bunun anlamı yakınsama hızının her yinelemede bir öncekine göre artması demektir. Bu olgu CG yönteminin öncelikle baskın öz değerlere ait öz vektörlerin doğrultusundaki hata bileşenlerini yok etmeye eğilimli olmasıyla açıklanabilir. Bu bileşenler elendikten sonra yöntem bu öz değerler sistemde mevcut değilmişcesine devam eder. Dolayısıyla bu yakınsama oranının daha küçük bir koşul sayısına sahip bir indirgenmiş sisteme bağlı olduğu düşünülerek çözüm değerlendirilebilir [2,3,11,12].

3. ÖN KOŞULLAYICILAR

Ön koşullayıcı yöntemlerin yakınsama hızı katsayılar matrisinin spektral özelliklerine bağlıdır. Bu yüzden lineer sistemi aynı çözüme sahip ama hesaplama verimliliği açısından daha uygun spektral özelliklere sahip diğer bir sisteme dönüştürmek tercih edilen bir yoldur. *Ön koşullayıcı* bu tip dönüşümleri gerçekleştirmekte kullanılan matrislere verilen isimdir [11,12].

Örneğin, katsayılar matrisi A' ya yaklaşık eşit bir M matrisi lineer sistemi $M^{-1}Ax = M^{-1}b$ şeklinde dönüştürdüğünde gerçek sistem $Ax = b$ ile aynı çözüme sahip olmasına rağmen kendi katsayılar matrisi $M^{-1}A$' nın spektral özellikleri çözüm açısından daha uygun olacaktır.

Ön koşullayıcı seçimi aşamasında iki yöntem izlenebilir; birincisi A' ya yaklaşık ve sistemi A ile olduğundan daha kolay çözecek bir M matrisi seçmek yada ikinci olarak sadece M ile çarpım gerektirecek A^{-1}' e yaklaşık olarak eşit bir M matrisi seçmek. Ön koşullayıcı yöntemlerin büyük çoğunluğu birinci kategoriye girmek ile birlikte bu çalışmada ikinci yöntem ile çalışılacak bir ön koşullayıcı seçilmektedir [11,12,13,14].

3.1 Ön koşullayıcıların Hesaplama Analizleri

Yinelemeli yöntemlerde kullanılan ön koşullayıcılar oluşturulma ve her bir yinelemedeki uygulanmaları aşamasında fazladan hesaplama gerektirdiklerinden bir seçim yapmadan önce neden olacakları hesapalama maliyetleri ile yakınsamada kazandıracakları hız arasında verimlilik açısından bir analiz ve tercih yapılmalıdır. Bazı ön koşullayıcılar az yada hiç kurulum aşaması gerektirmezken (örneğin SSOR ön koşullayıcılar) bazıları da örneğin tam olmayan ayrıştırmalarda olduğu gibi hatırı sayılır sayıda işlem yapmayı zorunlu kılmaktadır. Sayısal terimlerde yapılan iş tek bir yineleme ile karşılaştırılabilir olsa da, ön koşullayıcının uygulaması paralelleştirilebilir olmakla birlikte oluşturulması olmayabilir. Bu gibi durumlarda başlangıç maliyeti yinelemeler üzerinden yada aynı ön koşullayıcının birden çok

lineer sistemde kullanılmasıyla karşılanmalıdır.Pek çok ön koşullayıcı kullanıldıkları uygulamalarda değişkenlerin sayısı ile orantılı iş yaparlar.

Bu, ön koşullayıcıların her bir yinelemede yapılan işi sabit bir çarpanla çarptıkları anlamına gelir. Değiştirilmiş tam olmayan ayrıştırmalar ve multigrid teknikler bu sakıncaları aşmakta faydalı olabilir [11,12].

Bütün bunlarla birlikte parallel makinelerde elde edilecek performanslar halen kullanılmakta olan pek çok ön koşullayıcıda oldukça yetersiz kalmaktadır. Bunun nedeni kullanılan bu geleneksel ön koşullayıcıların paralleştirilmesinin verimli olmayan geniş seri parçalar içermesidir. Özet olarak söylemek gerekirse seçilecek bir ön koşullayıcının sahip olması gereken özellikler

- Oluşturulması ve uygulaması kolay olmalı.
- Katsayılar matrisinin spektrumunu, kullanılacak yöntemin yineleme sayısını azaltacak şekilde değiştirmeli.
- Paralel hesaplamaya uygun olmalı.
- Ön koşullayıcının hesaplama maliyeti mümkün olan en düşük seviyede kalmalı.

şeklinde sıralanabilir. İşte yapılan bu çalışmada yukarıda bahsedilen kriterlere karşılık verebilecek yeni bir ön koşullayıcı'nın parallel uygulaması üzerinde çalışılmıştır.

3.2 Chebyshev Ön Koşullayıcı

3.2.1 Chebyshev Polinomları ve Sayıların Yaklaşık Tersi

Sayısal değerli ortogonal Chebyshev polinomları

$$T_0(z) = 1, T_1(z) = z$$

$$T_k(z) = 2zT_{k-1}(z) - T_{k-2}(z), \qquad k = 2,3, \ldots \tag{3.1}$$

öz yinelemeli formu kullanılarak hesaplanabilir. Şekil 3.1' de farklı derecelerde ki Chebyshev polinomlarını görmek mümkün. Polinomlar $|\boldsymbol{T_i(z)}| \leq \mathbf{1}$ özelliğine sahip olup aslında $\boldsymbol{z} \in [-\mathbf{1}, \mathbf{1}]$ bölgesinde -1 ve 1 değerleri arasında salınım yapmaktadır.

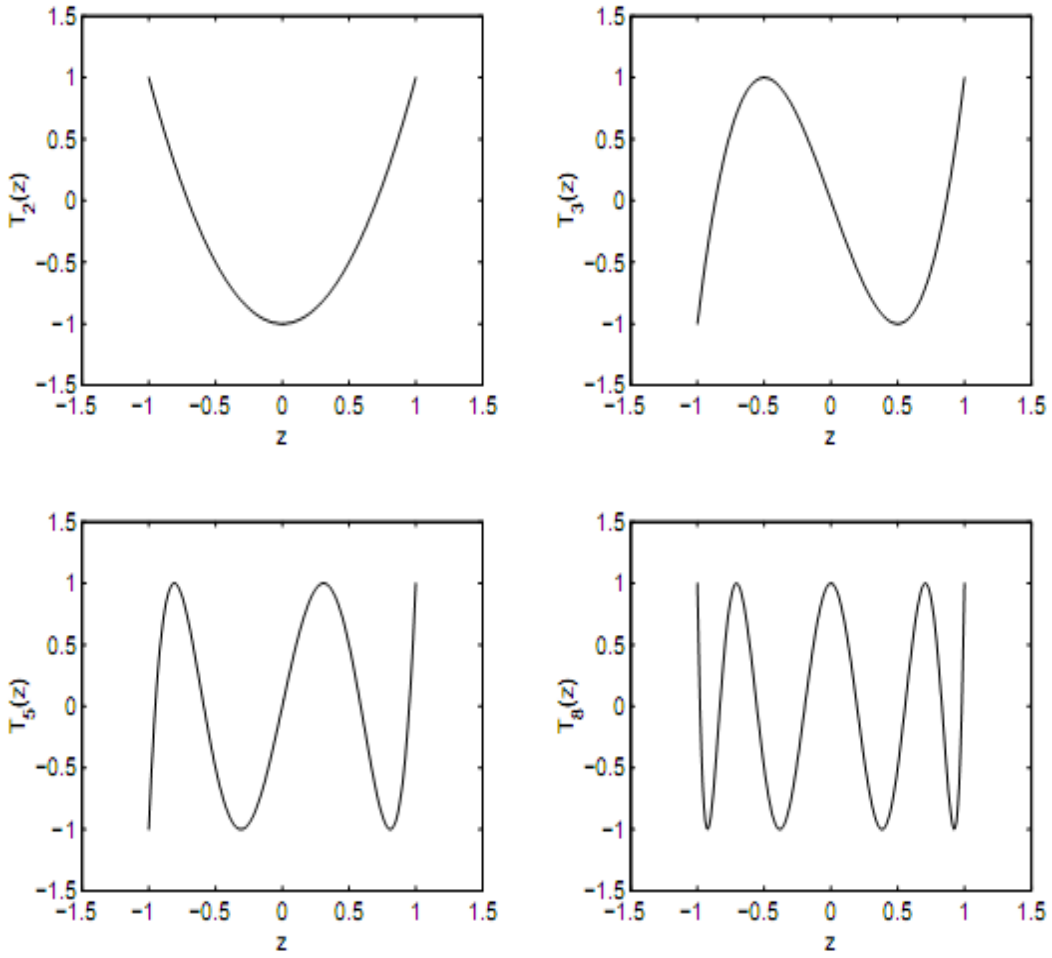
$\mathbf{0 < \alpha < y < \beta}$ aralığındaki y sayılarının tersi, polinomların lineer bir kombinasyonu

$$y^{-1} = \frac{c_0}{2} + \sum_{k=1}^{k=\infty} c_k T_k(z) \qquad \textbf{(3.2)}$$

yakınsaklık serisi kullanılarak hesaplanabilir. Burada $\boldsymbol{z = \frac{2}{\beta-\alpha}\left[y - \frac{\beta+\alpha}{2}\right]}$ olup y değerlerini $[\mathbf{-1, 1}]$ aralığında değiştirir. $[\boldsymbol{\alpha, \beta}]$ aralığı için **(3.2)**' de verilen $\boldsymbol{c_k}$ katsayıları

$$c_k = \frac{1}{\sqrt{\alpha\beta}}(-q)^k, \qquad q = \frac{1-\sqrt{\alpha/\beta}}{1+\sqrt{\alpha/\beta}} \qquad \textbf{(3.3)}$$

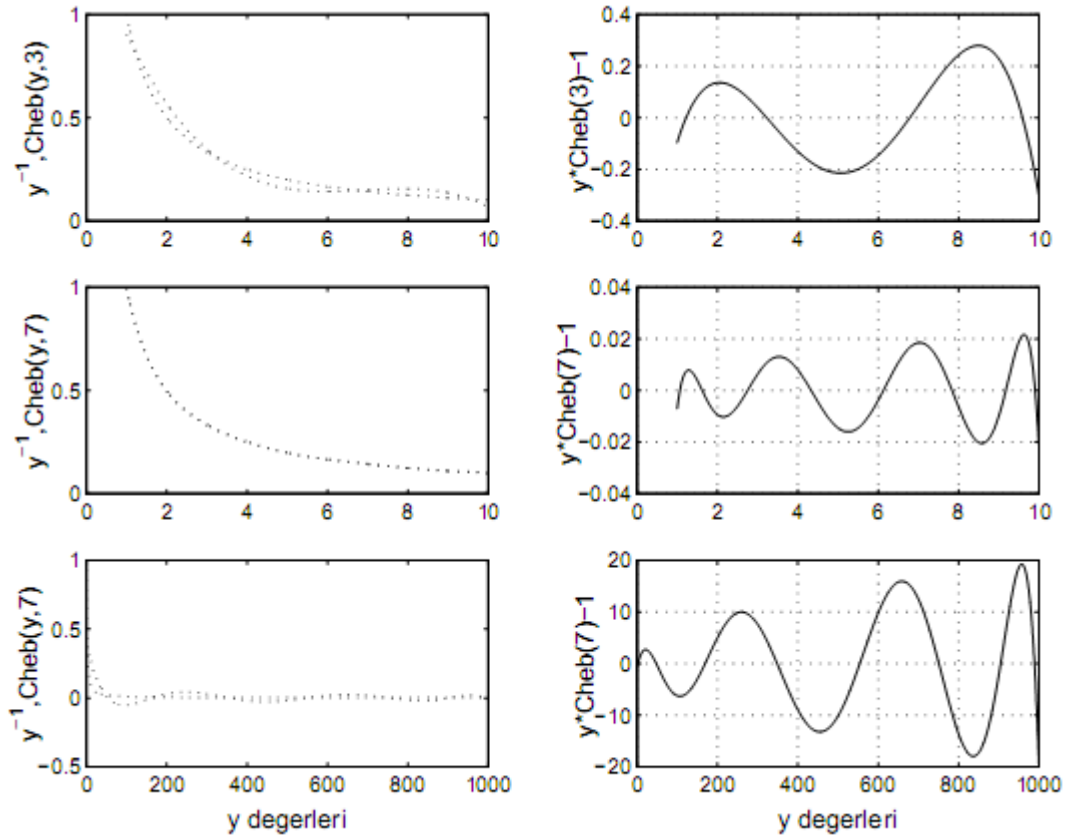
ifadesinden hesaplanabilir.



Şekil 3.1 : Sekizinci dereceye kadar Chebyshev polinomları $\boldsymbol{T_i(z)}$.

(3.2) y sayılarının tersinin tam olarak hesaplar. Bununla birlikte yukarıdaki aralıkta verilen sayıların yaklaşık tersleri Chebyshev polinomlarının dereceleri küçültülerek elde edilebilir. Sayıların yaklaşık terslerini elde etmek için kullanılan farklı derecelerdeki Chebyshev polinomlarının kullanımi Şekil 3.2' de gösterilmektedir.

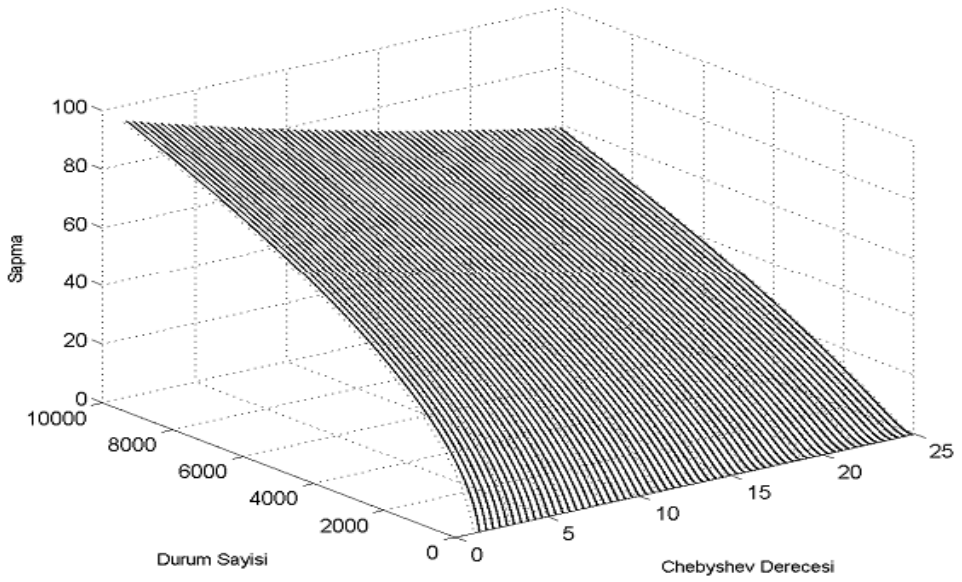
Şekil 3.2' deki $Cheb(y,r)$'nin anlamı y sayısının tersinin $r.$ dereceye kadar olan Chebyshev Polinomları kullanılarak elde edildiğidir. Şekilden görülebildiği gibi küçük aralıktaki sayıların (sol üst çizim) yaklaşık tersleri polinom derecesi 3'e kadar iyi sonuç vermektedir. Sayıların ve yaklaşık terslerinin çarpımının birim değerden sapma miktarı %25' den daha azdır (sağ üst çizim). Eğer polinom derecesi artırılır ise beklenildiği gibi yaklaşım iyileşecek ve sapma %2' den daha az olacaktır (ortadaki çizimler). Bununla birlikte sayı aralığı genişletildiğinde, polinom derecesinin 7 olmasına rağmen yaklaşık ters değerleri iyi olmamakta ve sapma %2000' nin üzerine çıkabilmektedir (alt çizimler). Şu andan itibaren sayı aralığını oluşturan en büyük sayının en küçüğe oranına koşul sayısı denilecek ve β/α ile ifade edilecektir [1].



Şekil 3.2 : Chebyshev polinomları kullanarak bir sayının tersi ve hata değerleri.

Şekil 3.3' de büyük sayı aralıkları için (büyük koşul sayısı) 25. dereceye kadar Chebyshev polinomları ile hesaplanmış yaklaşık ters değerlerinin sapmaları görülebilir. Sapmalar düzgün bir yüzey oluşturmaktadır. Beklenildiği gibi polinom derecesi artırıldığında sapma değerleri düşmektedir. Bununla birlikte 10000 ve üzeri gibi çok büyük koşul sayılarında polinom derecesi 25 dahi olsa sapma değerleri çok büyük olmaktadır. Amaç spektrumu $[\alpha, \beta]$ aralığında bulunan A matrisinin yaklaşık tersini hesaplamak. Bu yüzden α ve β değerleri matrisin en büyük ve en küçük öz değerleridir. Sayıların yaklaşık tersleri heasplanırken oluşan hata değerleri A matrisinin yaklaşık tersinin hesaplanmasında oluşacak hata değerini tahmin etmede kullanılabilir. Bir operatör olarak kullanılan herhangi bir matrisin öz değerleri ile tamamıyla karakterize edilebilmesi başlangıç noktası olarak sayıların tersi üzerinde yoğunlaşılmasının nedenini teşkil etmektedir.

Şekil 3.2 ve 3.3' den görüleceği gibi aralığın en sağ tarafında sapma en kötü değerini almaktadır. Bu, sayı aralığının en büyük elemanı olan β' ya yakın değerlerdeki sapmanın en küçük eleman olan α' ya göre daha büyük olduğunu ifade etmektedir. Burada önemli olan husus sapmanın büyük olduğu noktalardaki değerlerini azaltabilecek bir alternatif geliştirmek. **(3.3)**' de belirtildiği şekliyle Gerçek α değerini kullanmak yerine yine $[\alpha, \beta]$ aralığında kalacak daha büyük β/n değeri kullanılsın. Bu durumda elde edilecek $n = \beta/\alpha$ değeri *yapay koşul sayısı* olarak tanımlanacak. Örneğin, $\alpha = \beta/25$ durumunda yapay koşul sayısı 25 olacaktır.

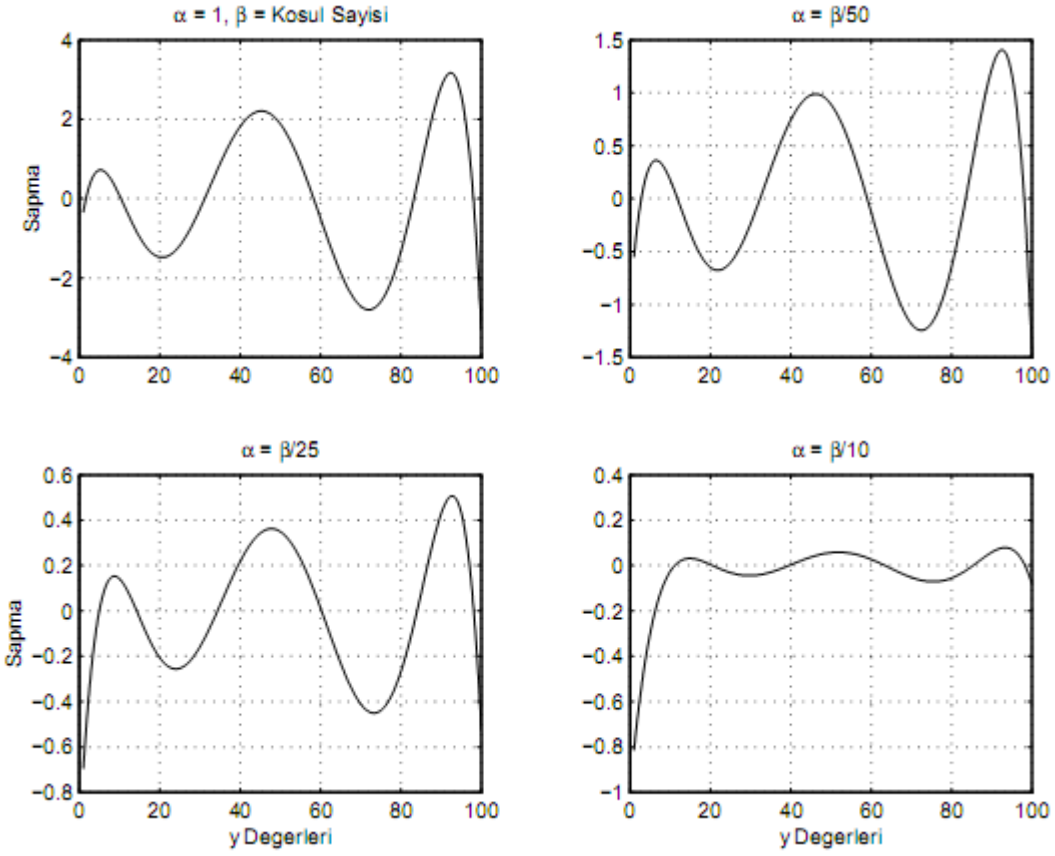


Şekil 3.3 : 25. dereceye kadar Chebyshev polinomları kullanılarak elde edilen yaklaşık sayı terslerinin sapma değerleri.

Şekil 3.4, 1 ile 100 arasındaki sayıların yapay koşullama sayısına karşı olan davranışını göstermektedir. Şekilden görülebildiği gibi yapay koşul sayısı küçülürken sayı aralığının sağ yanındaki sapma değerleri de küçülmekte ve sol yandaki sapma değerleri de hiç bir şekilde başlangıç $(\alpha = 1)$ değerinden kötü olmamaktadır. Dolayısıyla buradan çıkarılacak sonuç, $[\alpha, \beta]$ aralığındaki sayıların yaklaşık tersi bulunurken **(3.3)**' de kullanılan α değeri aralıktaki ilk sayıya eşit alınmalıdır. Burada önemli nokta en küçük sapmayı verecek en iyi yapay koşul sayısını seçebilmektir.

Bu hususla ilgili [1]' de farklı dağılımlara sahip sayı aralıklarında yapılmış olan istatiksel çalışmaların sonuçlarını görmek mümkün. Lineer ve logaritmik dağılımlı aralıklarda elde edilen deneysel sonuçlar aşağıdaki şekilde özetlenebilir;

- Logaritmik dağılımlı sayılar için en iyi yapay koşul sayısı kullanılan en büyük polinom derecesinin 5 katıdır.
- Lineer dağılımlı sayılar için en iyi yapay koşul sayısı $\lfloor . \rfloor$ en küçük tamsayıyı göstermek üzere $\lfloor (En\ büyük\ polinom\ derecesi) \rfloor \cdot 5$ ile ifade edilebilir.



Şekil 3.4 : Sayıların sapma değerlerinin yapay koşul sayısı ile değişimi.

Örneğin, logaritmik dağılımlı sayılarla 3. dereceye kadar olan polinomlar kullanıldığında en iyi yapay koşul sayısı 15 olacak ve böylece α değeri $\beta/15$ olarak tanımlanabilecek. Yine yukarıda belirtildiği gibi lineer dağılımlı sayılarda aynı polinom derecesiyle $\alpha = \beta/5$ olacak. Buraya kadar anlatılanlardan çıkarılabilecek sonuç A matrisinin yaklaşık tersi alınırken olacak. Buraya kadar anlatılanlardan çıkarılabilecek sonuç A matrisinin yaklaşık tersi alınırken α değeri en iyi yapay koşul sayısına eşit alınırken β değeri A matrisinin en büyük özdeğerine eşit olarak seçilebilir [1].

3.2.2 Chebyshev Polinomları ile Matrislerin Terslerinin Hesaplanması

Denklem **(3.2)** spektrumu $[\alpha, \beta]$ aralığında olan A matrisi için

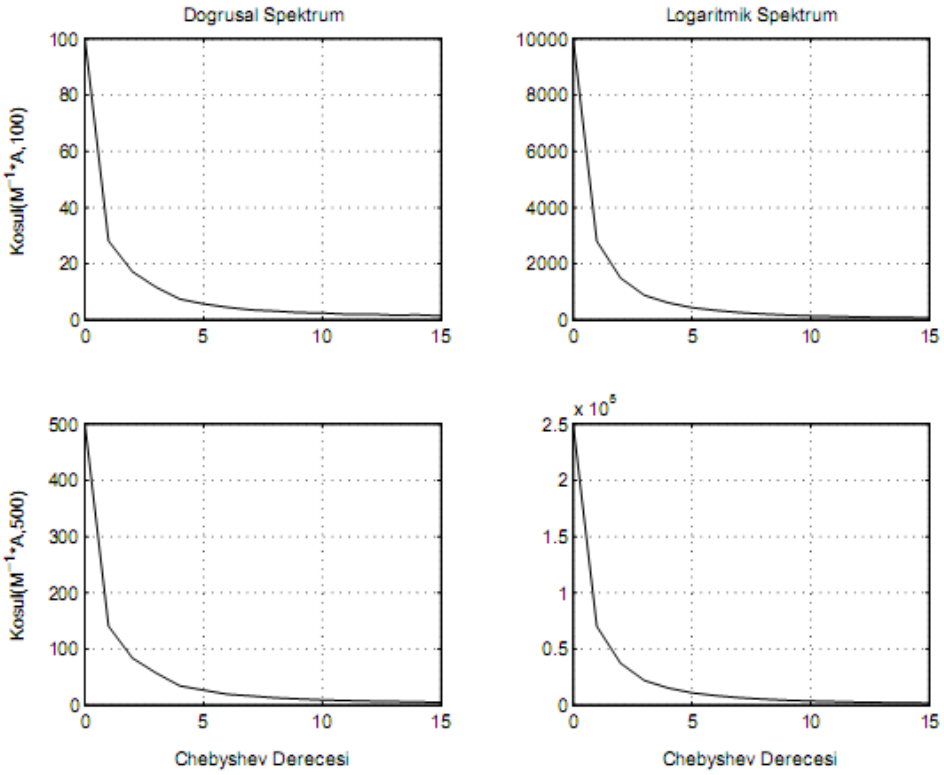
$$A^{-1} = \frac{c_0}{2} I + \sum_{k=1}^{k=\infty} c_k T_k(Z) \qquad \textbf{(3.4)}$$

şeklinde yazılabilir. Burada

$$Z = \frac{2}{\beta - \alpha}\left[A - \frac{\beta + \alpha}{2} I\right] \qquad \textbf{(3.5)}$$

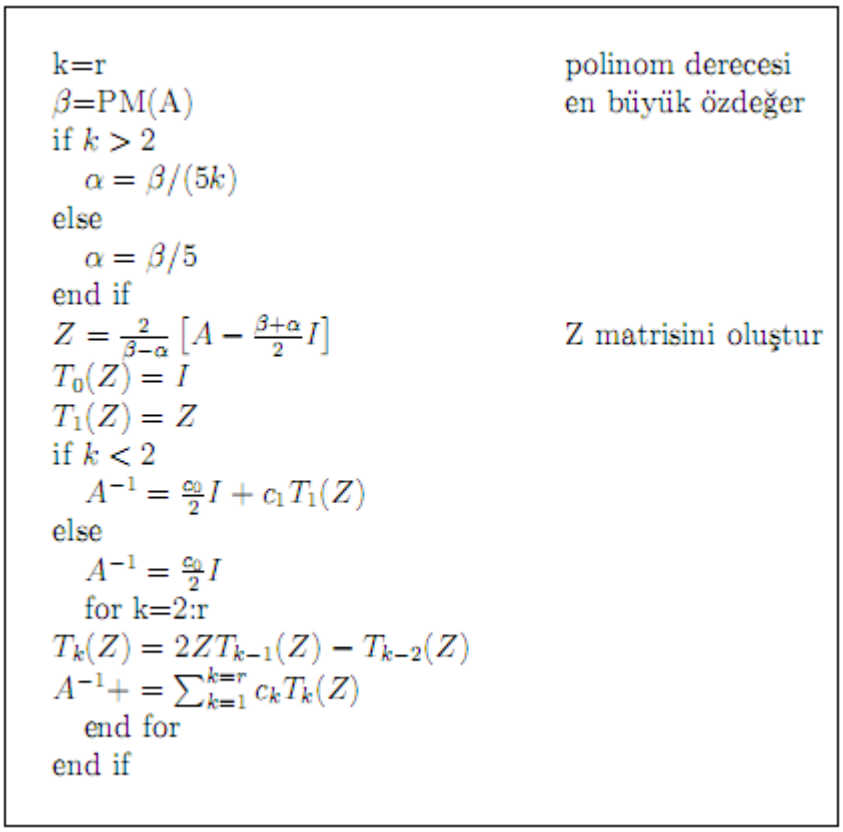
ve I birim matristir. α yapay koşul sayısı olup β değeri matrisin yaklaşık en büyük öz değerine karşılık gelir ve kuvvet yöntemi (power method) ile hesaplanabilir. Z matrisi A' nın spektrumunu $[-1,1]$ aralığına yerleştirir. $T_0(Z) = I$ ve $T_1(Z) = Z$ olmak üzere $T_i(Z)$ matris değerli Chebyshev polinomudur ve daha büyük dereceli polinomlar **(3.1)** kullanılarak hesaplanabilir. Buradaki amaç A matrisinin yaklaşık tersini bulmak olduğundan **(3.4)**' deki k çok küçük değerlerde durdurulabilir. Şekil 3.6' da bu yöntem ile ilgili algoritma görülebilir.

Şekil 3.5' de $n = 100$ ve $n = 500$ için öz değerleri lineer (sol çizimler) ve logaritmik (sağ çizimler) dağılımlı matrislerin yaklaşık tersleri ile kendilerinin çarpımının koşul sayısının Chebyshev polinom derecesi ile değişimi görülmektedir. Tüm çizimlerden de anlaşılabileceği gibi Chebyshev polinom derecesi artırıldıkça koşul sayısı beklenen değer olan 1' e yaklaşmakta. M^{-1} yaklaşık matris tersi CG yöntemi için ön koşullayıcı olarak kullanılacağından yapılan bu tespit ön koşullu CG yönteminin analizi ve yakınsaması açısından çok önemlidir.



Şekil 3.5 : Matrisin ve yaklaşık tersinin çarpımının, koşul sayısı ile Chebyshev polinom derecesinin değişimi.

Şekil 3.5' de $n = 100$ ve $n = 500$ için öz değerleri lineer (sol çizimler) ve logaritmik (sağ çizimler) dağılımlı matrislerin yaklaşık tersleri ile kendilerinin çarpımının koşul sayısının Chebyshev polinom derecesi ile değişimi görülmektedir. Tüm çizimlerden de anlaşılabileceği gibi Chebyshev polinom derecesi artırıldıkça koşul sayısı beklenen değer olan 1' e yaklaşmakta. M^{-1} yaklaşık matris tersi CG yöntemi için ön koşullayıcı olarak kullanılacağından yapılan bu tespit ön koşullu CG yönteminin analizi ve yakınsaması açısından çok önemlidir. Eğer polinom derecesi çok büyük seçilip $M^{-1} \cdot A = I$ olarak elde edilirse ön koşullu CG yöntemi tek yinelemede sonuca ulaşacaktır. Bununla birlikte dikkat edilirse Chebyshev polinom derecesinin her bir artırılışı fazladan bir matris-matris çarpımını gerektirecek ve dolayısıyla çok büyük polinom dereceleri için ön koşullayıcı matrisinin oluşturma maliyeti CG metodunun kendisinden daha büyük olacaktır. Buradan çıkarılması gereken sonuç Chebyshev polinom derecesini özellikle çok büyük boyutlu sistemlerde mümkün oldukça küçük tutmanın yöntemi hesaplama maliyeti açısından oldukça verimli kılacağıdır.



Şekil 3.6 : Chebyshev yöntemi ile matrisin tersinin alınması ile ilgili algoritma.

3.3 Ön Koşullu CG Yöntemi

Bölüm 2.3.3' de belirtildiği gibi CG yönteminin yakınsama hızı A matrisinin öz değerlerine bağlıdır. Spektral koşul sayısı $\kappa_2(A) = \lambda_{max}(A)/\lambda_{min}(A)$ hatanın derecesini belirleyecektir. Dolayısıyla burada üzerinde yoğunlaşılacak nokta $Ax = b$ sistemini öz değer dağılımı yakınsamayı daha da hızlandıracak bir yapıya kavuşturmak. Bu yaklaşım şekline önceki bölümlerde açıklandığı gibi lineer sistemin ön koşullanması denmektedir. Ön koşullu CG yönteminin temelini oluşturan fikir CG yöntemini dönüştürülmüş $\tilde{A}x = \tilde{b}$ sistemine uygulamaktır. Burada $\check{A} = M^{-1}A$ ve $\check{b} = M^{-1}b$' dir. Sonuçta elde edilecek algoritma Şekil 3.7' deki gibi olacaktır.

Hesaplama yükü açısından ön koşullu CG yöntemi, CG yönteminden farklı olarak her bir adımda fazladan bir matris vektör çarpımını zorunlu kılacaktır [3,11,12,13,14].

k=0;	$x_0 = 0$; $r_0 = b$	başlat
while	$r_k \neq 0$ do	durdurma kriteri
	$z_k = M^{-1} r_k$	önkoşullama
	$k = k + 1$	k yineleme sayısı
	if $k = 1$ do	
	$p_1 = z_0$	
	else	
	$\beta_k = \frac{r_{k-1}^T z_{k-1}}{r_{k-2}^T z_{k-2}}$	p_k arama doğrultu vektörü
	$p_k = z_{k-1} + \beta_k p_{k-1}$	x_{k-1}'i x_k' ya güncelliyor.
	end if	
	$\alpha_k = \frac{r_{k-1}^T z_{k-1}}{p_k^T A p_k}$	
	$x_k = x_{k-1} + \alpha_k p_k$	yinelemeyi güncelle
	$r_k = r_{k-1} - \alpha_k A p_k$	rezidüyü güncelle
end while		

Şekil 3.7 : Ön koşullu Conjugate Gradient yönteminin algoritması.

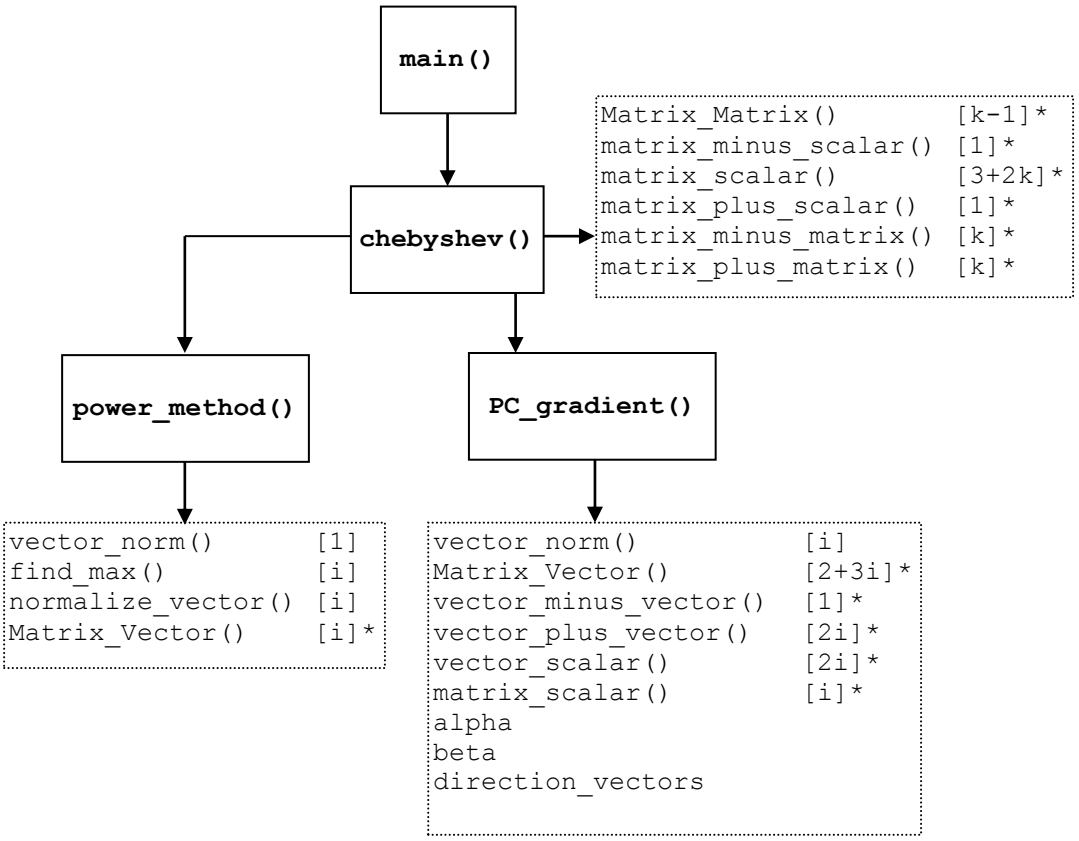
4. ÖN KOŞULLU CG YÖNTEMİNİN PARALEL PROGRAMLANMASI

Sayısal modelleme, bilimsel simülasyon ve mühendislik problemleri çoğu kez hesapalama açısından çok büyük hızlar gerektirirler çünkü bu tip çalışma alanları büyük miktardaki veri üzerinde geçerli sonucu elde edebilmek için pek çok işlemin defalarca yinelenmesini zorunlu kılar. Dolayısıyla hesaplamaların kabul edilebilir bir zaman aralığında tamamlanması verimlilik adına büyük öneme sahiptir. Bu bağlamda hesaplama hız ve verimliliğini artırma yollarının başında aynı problem üzerinde çalışan birden çok işlemcili sistemler kullanılması gelmektedir. Bu gibi bir durumda problem parçalara ayrılarak her birinin çözümü farklı bir işlemci tarafından gerçekleştirilir. Bu tip yapılar için gerçekleştirilen program yazma işlemine parallel programlama denmektedir.

Mevcut yöntemin paralel programlanması için, *UYBHM* dahilinde bulunan *Anadolu* isimli sunucu üzerinde her biri 2 core' dan oluşan 256 işlemci kullanılmıştır. Yöntemin paralel algoritması ve kodlaması, tüm işlemcilerin 2 boyutlu bir mesh teşkil edeceği şekilde tasarlanıp gerçekleştirilmiştir. Kod *C* dilinde ve platforma özel olarak optimize edilmiş *MPI* kütüphaneleri kullanılarak yazılmıştır. Kodlama ve performans analizleri değerlendirilerek, programlama dili ve platforma özel çeşitli optimizasyon teknikleri kullanılarak hatırı sayılır iyileştirmeler elde edilmiştir.

4.1 Fonksiyonlar ve Hesaplama Çekirdekleri

Yapılan bu çalışma kapsamında, CPCG (Chebyshev Ön Koşullu Conjıgate Gradient) algoritması ve kod uygulaması Şekil 4.1' de görüleceği üzere 3 ana fonksiyondan oluşmaktadır. *main()* içersinde problem verilerinin dışarıdan okunmasını ardından *chebyshev()* fonksiyonu çağrılır. *chebyshev()* öncelikle *power_method()* çağırarak katsayılar matrisinin en büyük öz değerinin yaklaşık bir değerini hesaplar. Bunun ardından *chebyshev()*, ön koşullayıcı olarak kullanılacak katsayılar matrsinin yaklaşık tersini hesaplar ve bunu *PC_gradient()* fonksiyonuna geçirerek PCG yönteminin çözümü gerçekleştirmesini sağlar.



Şekil 4.1 : CPCG yönteminin fonksiyon ve hesaplama çekirdekleri.

Şekil 4.1' de her bir fonksiyonun çağırdığı hesaplama çekirdekleri görülebilir. Bu çekirdek fonksiyonlarının karşısında ifade edilen [ ] içersindeki değerler, o fonksiyonun kodun koşması süresince kaç kez çağrıldığını göstermektedir. *i* değeri *power_method()* yada *PC_gradient()* yineleme sayılarına karşılık gelirken *k* değeri *chebyshev()* fonksiyonu tarafından kullanılan Chebyshev polinom derecesini göstermektedir. Yanında * ile ifade edilen çekirdek fonksiyonlar, o fonksiyonun mevcut kod içersinde paralelleştirildiği anlamına gelmektedir.

4.2 2 Boyutlu Mesh Topolojisi

Problemin mevcut sistem üzerindeki uygulaması, mevcut işlemcilerin MPI arayüzü kullanılarak iki boyutta kartezyen yapısında gruplandırılmaktadır. 2 boyutlu yapının kullanılması, üzerinde çalışılacak olan *A* katsayılar matrisi ve *b* sağ yan vektörünün bu yapıya uygun olarak blok'lar halinde ayrıştırılmasını zorunlu kılar.

4.2.1 Tek Boyutlu ve İki Boyutlu Ayrıştırma

Tek boyutlu ve iki boyutlu topolojilerin iletişim maliyetlerini karşılaştırmak için üzerinde çalıştığımız NxN boyutlu problemi göz önüne alalım. Tek boyutlu yapıda, problemi *n* adet işlemciye dağıtmak için bir boyutu *N* diğer boyutu N/n olan bir ayrıştırma gerçekleştirilir. İki boyutlu yapıda ise problem her bir boyutu N/n olan parçalara ayrıştırılır.

m byte boyutlu bir veriyi yollayabilmek için gerekli olan zamanı en basit şekliyle $T_{comm} = s + rm$ olarak tanımlayabiliriz. Burada s gecikme zamanı yani; iletişimi başlatmak için geçen zaman, *r* ise 1 byte veri yollamak için ihtiyaç duyulan zaman olarak tanımlanabilir. Büyük boyutlu mesajlarda veriyi yollamak için harcanak zaman gecikme zamanına kıyasla çok büyük olacağı için $T_{comm} \approx rm$ şeklinde verilebilir.

Kendi komşusuyla veri alış verişi yapan bir işlemciyi göz önüne alacak olursak iletişim maliyeti tek boyutlu topoloji için $T_{comm} \approx 4(s + rN)$ olarak verilirken iki boyutlu topolojiler için $T_{comm} \approx 4\left(s + rN/\sqrt{n}\right)$ olacaktır. Buradan görüleceği üzere iki boyutlu topolojilerde işlemci sayısı artırıldıkça iletişim maliyeti azalacaktır, dolayısıyla iki boyutta ayrıştırma tek boyutu nazaran daha verimli olacaktır.

4.3 Hesaplama Çekirdeklerinin Paralel Programlanması

Şekil 4.1' de görülen hesaplama çekirdeklerinden problemin verimliliği ve karakteristiği açısından en baskın olan 2 tanesi *Matrix_Matrix()* ve *Matrix_Vector()* fonksiyonlarıdır. Bu fonksiyonlar sırasıyla matris-matris ve matris-vektör çarpımlarını gerçekleştirir ve tüm kodun çalışma zamanının yaklaşık %90' nını harcamaktadır.

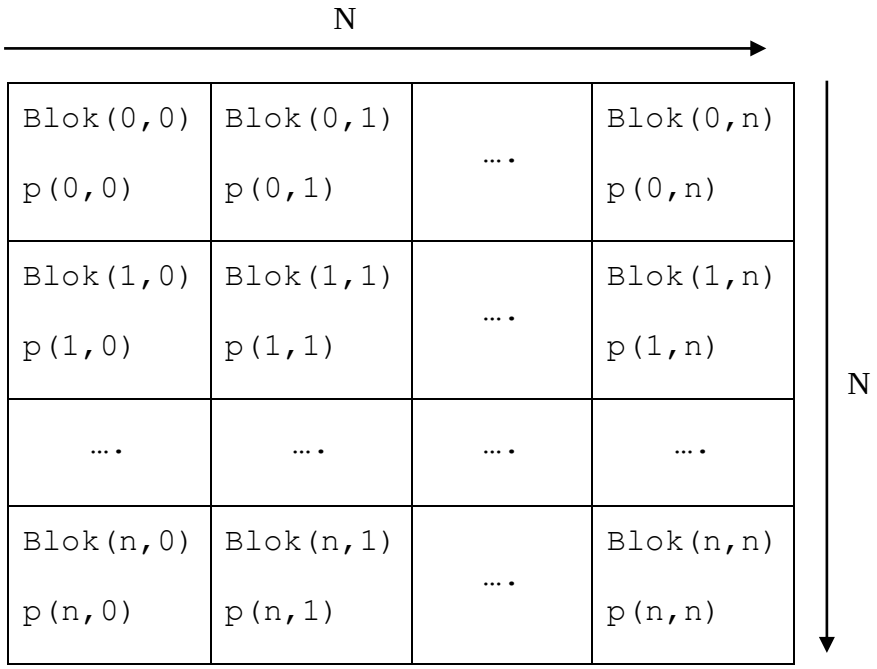
Şekil 4.2' de *nxn* büyüklükte 2 boyutlu mesh toplojisine göre dağıtılmış işlemcilerin koordinatları görülmektedir. Programlama tekniği açısından her bir işlemci koordinatlari ile tanimlanip birbirleri ile iletişime bu koordinatlar vasıtası ile geçmektedirler.

Kodun işlemciler üzerinde çalışmaya başlaması ile birlikte *A* katsayılar matrisi Şekil 4.2' deki işlemci gruplarına, matris üzerinde karşılık gelen bloklar dağıtılacak şekilde bir ayrıştırma gerçekleştirilir.

p(0,0)	p(0,1)	….	p(0,n)
p(1,0)	p(1,1)	….	p(1,n)
….	….	….	….
p(n,0)	p(n,1)	….	p(n,n)

Şekil 4.2 : 2 boyutlu mesh topolojisine göre dağıtılmış *n* adet işlemci.

A matrsinin işlemcilere bloklar halinde dağıtılmasından itibaren, tüm hesaplama tamamlanana kadar bu bloklar Şekil 4.1' de görülen paralel hesaplama çekirdekleri tarafından kullanılır.



Şekil 4.3 : *NxN* boyutlu matrsin 2 boyutlu mesh topolojisine göre ayrışması.

Şekil 4.3' de katsayılar matrisinin 2 boyutlu mesh topolojisine göre nasıl ayrıştırıldığı görülebilir. Her bir işlemci ilgili matris blok verisini tüm işlemler için kullanacaktır. Her bir blok, boyutu $(N/n)x(N/n)$ olan matris parçalarını ifade etmektedir. CPCG algoritmasının çalışması boyunca işlemciler üzerinde dağıtılan katsayılar matrisinin blokları değişmeden local olarak saklanacaktır .

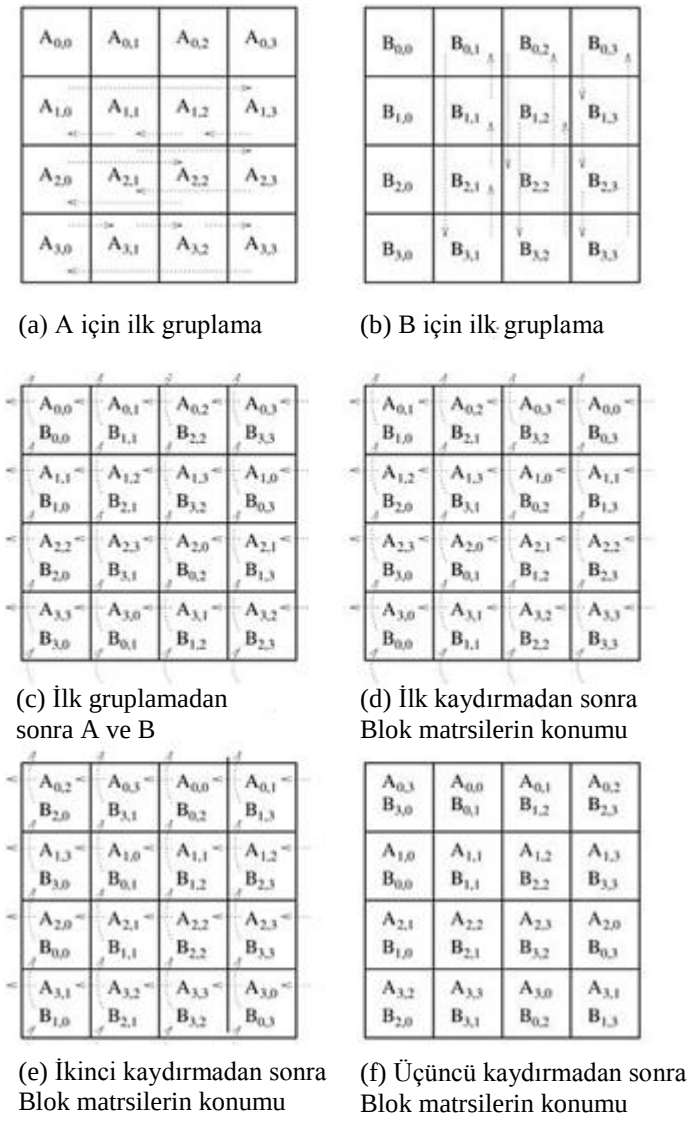
4.3.1 MPI İle İki Boyutlu Mesh Topolojisinin Oluşturulması

Yapılan çalışmada kullanılan ana hesaplama çekirdeği olan Cannon matris çarpımının karakteristiği gereği, mevcut işlemciler kullanılarak iki boyutlu kare mesh oluşturulmaktadır. Bu maksatla *Init_Mesh(MESH_DATA *grid)* isimli bir ilklendirme fonksiyonu kullanılmaktadır. Bu fonksiyon MPI kütüphaneleri içersinde tanımlı olan *MPI_Cart_create* ve *MPI_Cart_coords* fonksiyonlarını çağırmaktadır. Bu fonksiyonlara sırasıyla *periods* (tekrar sayısı) ve *dimensions* (boyut) boyut değişkenleri gerçirilmekte ve algoritma tarafından kullanılması için *coordinates* (koordinat) değişkeni döndürülmektedir.

Bu fonksiyonunu tüm işlemciler tarafından ilklendirilmesi ile birlikte her bir işlemci Şekil 4.2'de görülen yapı içersinde 2 boyutlu topoloji üzerinde kendi koordinatlarını elde eder. Bölüm 4.3' de tanımlandığı üzere ikili koordinatlara sahip işlemciler Şekil 4.3' de gösterildiği gibi kendi koordinat değerlerine karşılık gelen blok veri yapılarını *MPI_double* veri tipi ile yerellerinde saklarlar. Bu saklanan blok veri değerleri algoritmanın sonraki aşamalarında, satır ve sütun grup işlemciler tarafından kendi aralarında, koordinat olarak gelen bir sonraki işlemciye kaydırılarak blok veri transferini gerçekleştirirler.

4.3.2 2 Boyutlu Mesh Topolojisi Üzerinde Cannon Matris Çarpımı

Cannon algoritması, Şekil 4.3' de olduğu gibi 2 boyutlu mesh topolojisine göre dağıtılmış işlemcileri matris çarpımına girecek *A, B* ve sonuç matrisi olan *C*'yi bloklar halinde tutmak için kullanır.



Şekil 4.4 : 3x3'luk mesh topoloji için Paralel Cannon algoritması.

Eğer matris boyutları NxN ve işlemci sayısı p ise ayrıştırılan matrislerin blok boyutları $(n/\sqrt{p})x(n/\sqrt{p})$ olacaktır. Mesh üzerindeki Pi,j işlemcisine sırası ile Ai,j, Bi,j ve Ci,j blok matrisleri yerleştirilecektir. Şekil 4.4' de görülen ilk gruplamadan sonra algoritma $\sqrt{p}$ adım çalışacaktır. Her bir adımda işlemciler, Şekil 4.4' de olduğu gibi çarpıma giren matrislerin yerel bloklarını, *A* matrisine ait olan sol *B* matrisine sahip olanlar üst taraftaki komşu işlemcilere yollanacak şekilde ilerlerler [15].

Yapılan bu işleme kaydırma adı verilmektedir. İşlemciler her bir adımda kaydırma işlemini gerçekleştirmeden önce, yerellerinde bulunan sonuc matrislerini, matris bloklarına seri matris-matris çarpımı uygulayarak hesaplarlar. Elde edilen sonuç local Ci,j matrisi üzerinde tututulur.

Her bir adım sonunda elde edilen sonuç Ci,j yerel matrisi üzerine kümülatif olarak eklenir. Tüm adımlar hesaplandıktan sonra işlemciler üzerinde sonuç matris *C*'nin blokları bir sonraki algoritma adımında kullanılmak üzere saklanır.

Yapılan çalışmada kullanılan Cannon matris çarpımını gerçekleştiren fonksiyon;

*void Matrix_Matrix(double *a, double *b, double *c, int n, MESH_DATA *grid)*

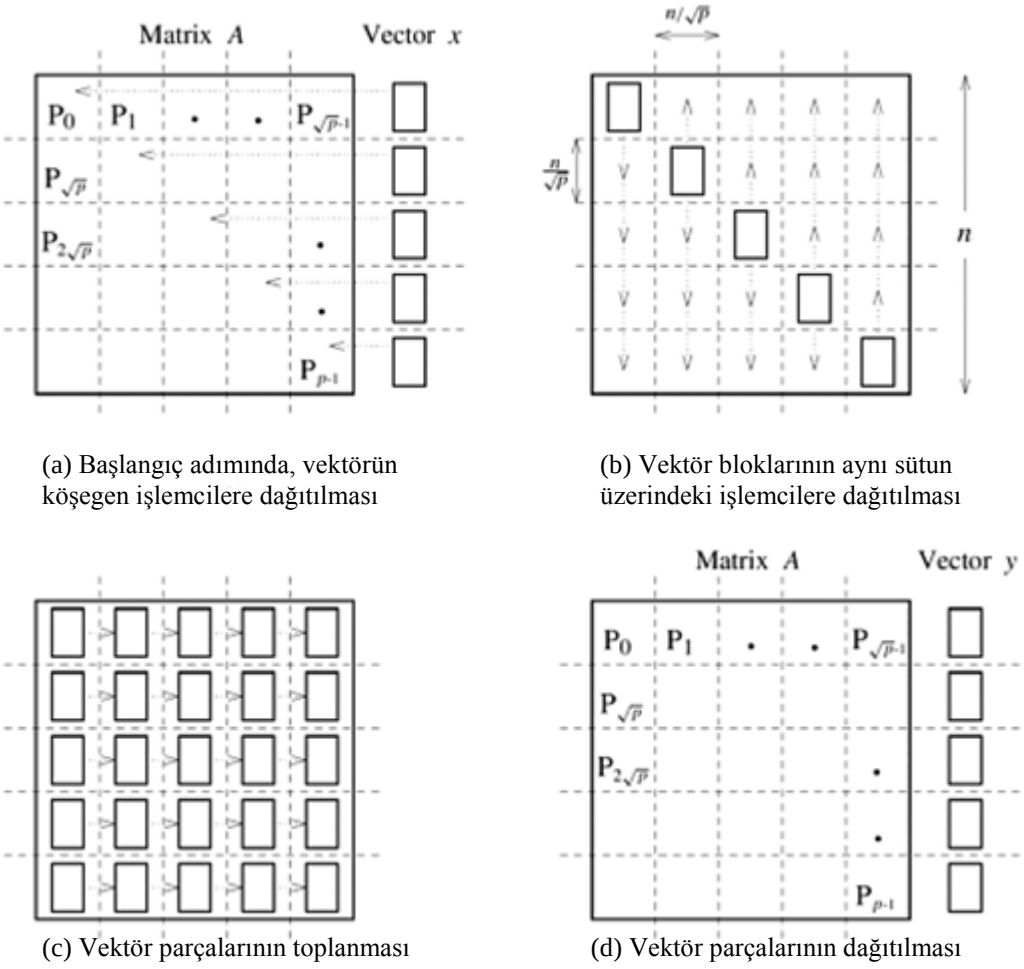
olarak kullanılmaktadır. Burada *a* ve *b* çarpıma giren matris blokları yani Ai,j ve Bi,j' dir. *c* sonuç matris Ci,j olarak sonraki işlemlerde kullanılmak üzere saklanmaktadır. *MESH_DATA *grid* Şekil 4.2' de tanımlanan topolojiyi kullanılmak üzere çekirdek fonksiyona geçirmek için kullanılır. En son adımdan sonra blok matrsiler orjinal yerlerine tekrar gelmiş olurlar. Bu, diğer parallel çekirdek fonksiyonların doğru matris bloklarını kullanabilmesi için önemlidir.

4.3.3 2 Boyutlu Mesh Topolojisi Üzerinde Matris-Vektör Çarpımı

2 boyutlu matris-vektör çarpımı, Şekil 4.3' deki mesh topolojiyi kullanır. Şekil 4.5' de görüldüğü üzere 2 boyutlu matris-vektör çarpımında, *n* matris ve vektör boyutunu *p* ise işlemci sayısını ifade etmek üzere öncelikle sağ yan vektörü *b*, 2 boyutlu grid üzerine yerleştirilmiş işlemcilere, köşegen üzerinde bulunanlara yollanacak şekilde parçalanarak dağıtılmaktadır. Köşegen işlemcileri $P_{i,i}\ \ i = 1,2,3,\dots,\sqrt{p}$ olarak tanımlanabilir [15].

Köşegen üzerinde bulunan işlemciler kendilerine gelen vektör bloklarını yerel olarak sakladıkları matris blokları ile çarpıp elde edilen sonuçları yerellerinde sakladıkları sonuç vektörüne yazarlar. Köşegen üzerinde bulunan işlemciler kendilerine gelmiş olan vektör bloklarını bir sonraki adımda *MPI_Bcast()* fonksiyonunu kullanarak kendi sütünlarında bulunan bütün işlemcilere dağıtırlar. İşlemciler aldıkları bu vektör bloklarını yerel matris blokları ile çarpıp elde ettikleri sonuçları yerel vektör bloklarına yazarlar.

Tüm işlemciler hesaplamalarını tamamladıktan sonra yerel sonuç vektörlerini toplanmak üzere kendi satırlarında bulunan ve $P_{i,0}\ \ i = 1,2,3,\dots,\sqrt{p}$ olarak ifade edilebilen birinci sütun işlemcilerine yollarlar. Böylece sonuç vektörü yerel olarak birinci sütun işlemcilerinde sonraki adımda bir araya getirilmek üzere saklanır.



(a) Başlangıç adımında, vektörün köşegen işlemcilere dağıtılması

(b) Vektör bloklarının aynı sütun üzerindeki işlemcilere dağıtılması

(c) Vektör parçalarının toplanması

(d) Vektör parçalarının dağıtılması

Şekil 4.5 : 2 boyutlu paralel matrsi-vektör çarpımı.

Yapılan çalışmada kullanılan, matris-vektör çarpımını gerçekleştiren fonksiyon;

*void Matrix_Vector(double *a, double *b, double *x1_block, int n, MESH_DATA *grid)*

Fonksiyonda kullanılan değişkenlerden; *a* çarpıma giren matris blok'u *b* çarpıma giren vektör ve *x1_block* sonuç vektör bloku olarak tanımlanmaktadır. *n* çarpımı gerçekleştirilen sistemin boyutu ve *MESH_DATA *grid* Şekil 4.2' de tanımlanan topolojiyi, kullanılmak üzere çekirdek fonksiyona geçirmek için kullanılır.

4.3.4 CPCG Uygulamasında Kullanılan Diğer Hesaplama Çekirdekleri

Şekil 4.1' de görülen fonksiyonlar tarafından kullanılan çekirdek fonksiyonların detaylı listesi Çizelge 4.1' de görülebilir. Çizlege 4.1' de görülen ve paralel uygulaması olmayan fonksiyonların kullandığı değişkenler ayrıştırılmamaış vektörler olup, tüm işlemciler tarafından çağrılıp kullanılırlar. Paralel uygulaması olan fonksiyonların hepsi, oluşturulmuş olan 2 boyutlu mesh topolojisini kullanarak

çalışırlar ve kendilerine geçirilen matris ve vektörlerin hepsi, Şekil 4.3' de görülen yapıya göre ayrıştırılmış blok formatındaki verilerdir.

Çizelge 4.1' de ifade edilen hesaplama çekirdek fonksiyonları daha önceden belirtildiği ve uygulamalı olarak gösterileceği gibi tüm uygulama işlem zamanının sadece %10' nun teşkil etmektedirler.

Çizelge 4.1 : CPCG uygulamasında kullanılan tüm hesaplama çekirdekleri.

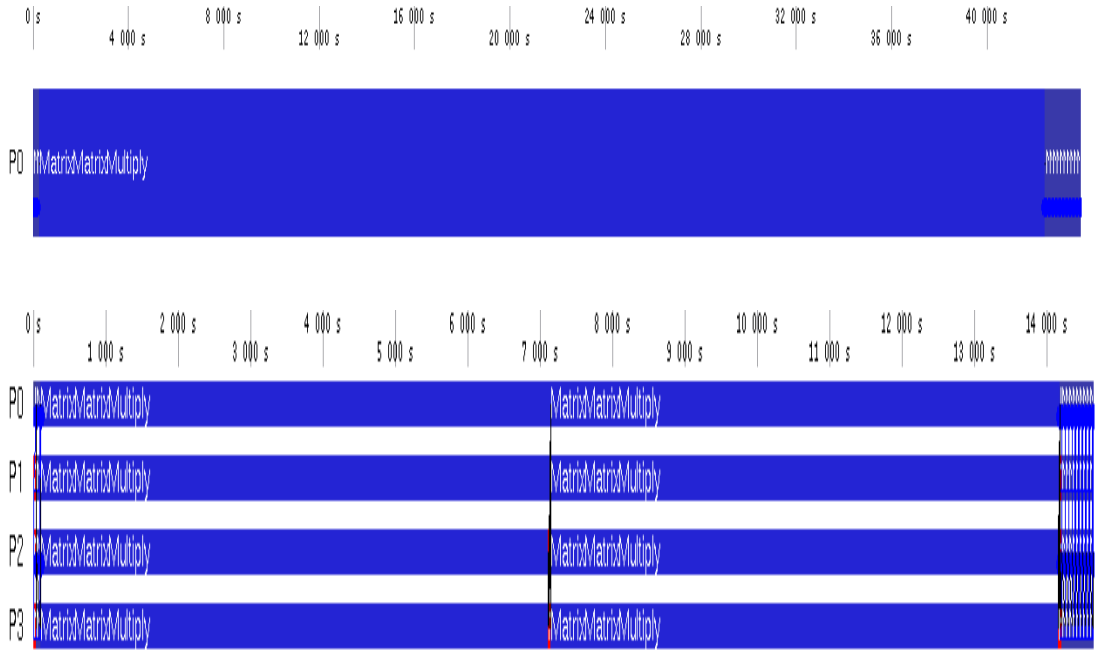
Çekirdek Fonksiyon Adı	Paralel Kullanım	Fonksiyon Değişkenleri
`find_max()`	Yok	`double *vector, double *result, int size`
`vector_norm()`	Yok	`double *vector, double *result, int size`
`normalize_vector()`	Yok	`double *vector, double value, int size`
`vector_minus_vector()`	Var	`double *vector_a, double *vector_b, double *result, int size`
`vector_plus_vector()`	Var	`double *vector_a, double *vector_b, double *result, int size`
`vector_scalar()`	Var	`double *vector, double *result, double scalar, int size`
`matrix_scalar()`	Var	`double *matrix, double *result, double scalar, int size`
`matrix_plus_scalar()`	Var	`double *matrix, double scalar, int size, int Row, int Col, int rank`
`matrix_minus_scalar()`	Var	`double *matrix, double scalar, int size, int Row, int Col, int rank`
`matrix_minus_matrix()`	Var	`double *matrix_a, double *matrix_b, double *result, int size`
`matrix_plus_matrix()`	Var	`double *matrix_a, double *matrix_b, double *result, int size`

5. CPCG YÖNTEMİNİN PARALEL UYGULAMASI

Bu bölümde, üzerinde çalışılan yöntemin uygulamasını gerçekleştirmek üzere 4 farklı boyutta deneysel problem tipi seçilmiştir; *2048, 4096, 8192 ve 12800.* Bu problemler Chebyshev derecesi $k = 2$ olacak ve iterasyon hata değerleri $\varepsilon = 0.01$ ile limitlenecek şekilde uygulama kodu çalıştırılmıştır.

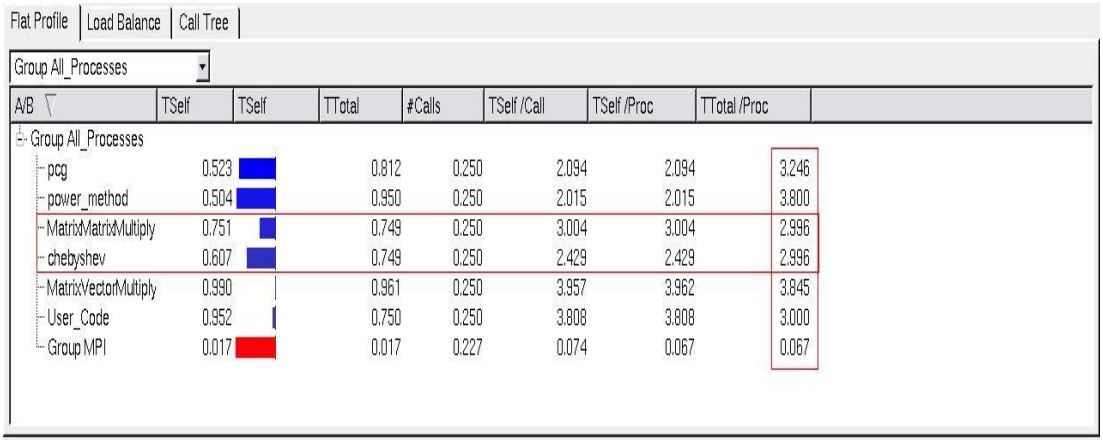
5.1 Paralel Uygulamanın Performans Değerlendirmesi

Şekil 5.1' de 8192 boyutlu problemin 1 ve 4 işlemci ile yapılan uygulaması görülmektedir. Baskın hesapalama çekirdeği olarak matris-matris çarpımı tüm uygulamanın CPU zamanının yaklaşık %90' nını domine etmektedir.



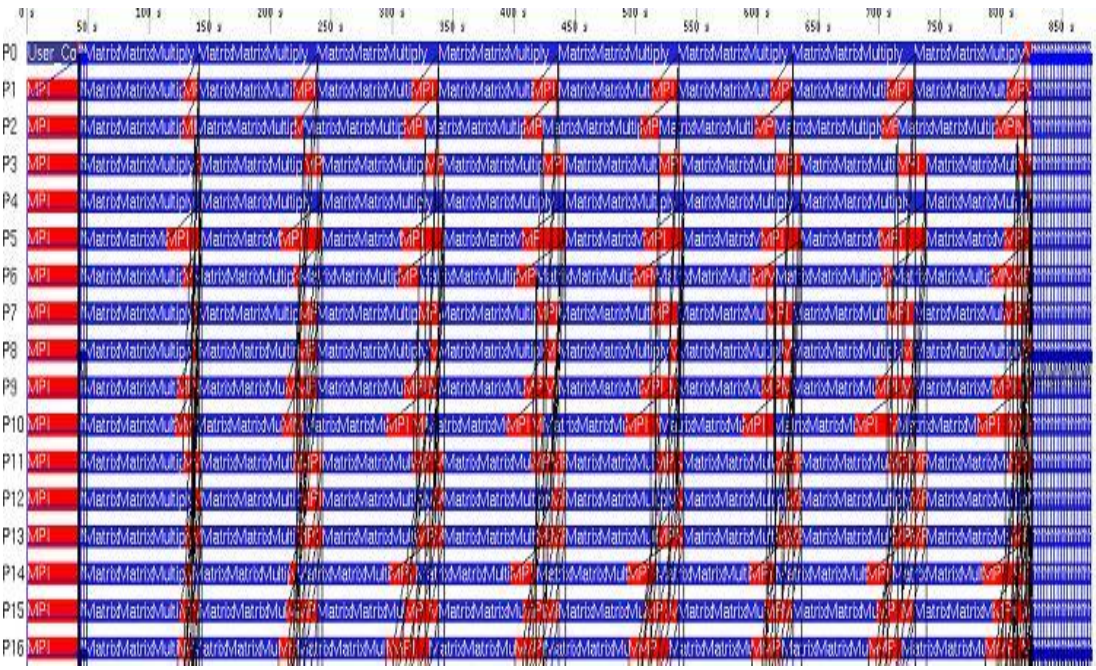
Şekil 5.1 : 8192 boyutlu problemin 1 ve 4 işlemcili CPU zamanları.

Şekil 5.2' de performans karşılaştıması yapabilmek maksadı ile kullanılacak olan hızlanma değerleri görülebilmektedir. Hızlanma değerini $T_s = T_1/T_n$ olacak şekilde ifade edebiliriz. Burada T_1 kodun tek bir işlemci üzerinde koştuğunda harcanan CPU zamanı olmakla birlikte, T_n işlemci sayısı n olduğu zaman ölçülen CPU zamanını göstermektedir.



Şekil 5.2 : 8192 boyutlu problemin 4 işlemcili hızlanma değerleri.

Şekil 5.2' de görülebildiği gibi, CPCG yönteminin baskın hesaplama çekirdeği olan matris-matris çarpımının hızlanma değerinin 4 işlemci için yaklaşık olarak 3' dür. Şekil 5.3' de işlemci sayısı 64' e çıkararak yapılmış bir test sonucu görülmektedir. Test sonucu detaylı olarak incelendiğinde anlaşılacağı gibi işlemci sayısının artırılmasıyla birlikte şekilin baş tarafında görülen, dosya okuma ve verilerin dağıtılması için harcanan iletişim zamanın artmasına ek olarak özellikle matris-matris çarpımı ve iterasyon döngüleri içindeki işlemciler arası iletişim zamanlarınında arttığı fark edilebilir (Kırmızı yada koyu renkler MPI iletişim fonksiyonlarının harcadığı CPU zamanlarını göstermekte ve Mavi yada açık renkler uygulamalar tarafından harcanan CPU zamanlarıdır).



Şekil 5.3 : 8192 boyutlu problemin 64 işlemci ile elde edilen CPU zamanlar.

5.2 CPCG Yönteminin Paralel Kodunun Optimize Edilmesi

Bölüm 5.1' de yapılan ilk test sonuçlarından görüldüğü üzere, problemin çözümünü gerçekleştiren parallel kodun bir takım optimizasyonlara ihtiyacı bulunmaktadır. Özellikle MPI iletişim fonksiyonları tarafından kodun belli bölgelerinin bloklandığını ve iletişim maliyetinin çok yüksek olduğunu fark etmekteyiz. Bunlara ek olarak baskın hesaplama çekirdeği olan matris-matris çarpımının harcadığı CPU zamanı optimize edip iyileştirmek mümkün olmaktadır.

5.2.1 Veri Dosyasının Okunma ve Dağıtılmasının Optimize Edilmesi

Problemin çözümünde kullanılacak katsayılar matrisi ve sağ yan vektörü koddan bağımsız olarak bir veri dosyasında tutulmaktadır. Yapılan ilk testlerde kullanılan kod içersinde, bu veri dosyası *root* olarak adlandırılan işlemci tarafından okunup, ayrıştırılıp grid üzerinde bulunan tüm işlemcilere dağıtılmaktadır. İlgili kod parçası,

```
if (grid->myrank==0){
  /* Root işlemci kendi blokunu alır */
  for(i=0;i<Block_size;i++)
    for(j=0;j<Block_size;j++)
      A_Block[i*Block_size + j] = matrix[stride*i+j];
  /* Root işlemci yerel blokları diğer işlemciler dağıtır*/
   for(i=1;i<grid->npes;i++)
     MPI_Send(&matrix[displs[i]], 1, blockmatrix, i ....);
  /* İşlemciler kendi blokllarını alır */
}else{
   MPI_Recv(A_Block, Block_size*Block_size, MPI_DOUBLE....);
} /* EndIf */
```

olarak verilebilir. Bu kod parçasından anlaşılacağı gibi problem boyutu arttıkça ayrıştırılacak veri boyutu artacak buda iletişim zamanlarını artıracaktır. Bütün bunlara ek olarak veri transferi için kullanılan *MPI_Send* ve *MPI_Recv* fonksiyonları, *root* işlemci ayrıştırma ve dağıtma işlemini bitirene kadar diğer

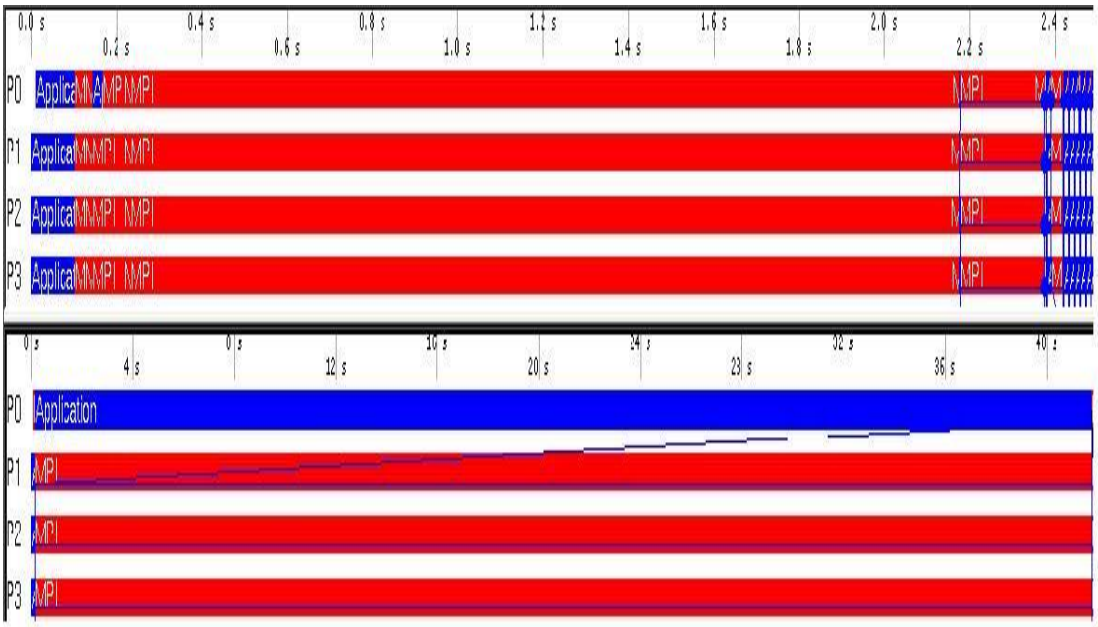
işlemcileri bloklayacaktır. Şekil 5.4' de (alt resim) başlangıç sürecinde 0 numaralı root işlemci dışındaki diğer işlemcilerin bloklandığı görülebilinir.

Verimliliği artırmak adına bu problemi aşmak için işlemcilerin kendi veri bloklarını eş zamanlı olarak okumasını sağlamak hatırı sayılır bir iyileştirmeye neden olacaktır. Bunun için MPI kütüphanesi içinde bulunan FILE I/O fonksiyonlarını kullanarak vektörel veri tanımlaması yapılacaktır. Aşağıda bu optimizasyonu içeren kod parçası görülebilir,

```
/* Dosya aç */
mode = MPI_MODE_RDONLY;
MPI_File_open( MPI_COMM_WORLD, file_name, mode, MPI_INFO_NULL,
&fh_A );
/* A blok oku */
offsetA = MPI_OFFSET_ZERO;
MPI_File_read_at_all( fh_A, offsetA, A_Block, B_Matrix_size,  MPI_DOUBLE,
&status );
/* Dosya kapat */
MPI_File_close( &fh_A );
```

Bu optimizasyon ardından tüm işlemciler eş zamanlı olarak veri dosyasına ulaşıp kendi matris blok verilerini dosya içersinden alacaklardır.

Şekil 5.4' de optimizasyondan önce (alt resim) ve optimizasyondan sonra (üst resim) test edilmiş veri okuma CPU zamanları görülebilir. 8192 boyutlu problem için 64 işlemci kullanıldığında yaklaşık 20 kat daha hızlı veri okuma gerçekleştirilmiştir.



Şekil 5.4 : Optimizayon öncesi ve sonrası veri dosyası okuma hızları.

5.2.2 Cannon' un Bloksuz Uygulaması İle Optimizasyon Yapılması

Yapılan ilk test çalışmalarında Cannon algoritmasının uygulamasında Bölüm 4.3.1' de anlatılan işlemciler arası matris bloklarının kaydırılması için bloklu iletişim fonksiyonu olarak adlandırılan *MPI_Sendrecv_replace* çağrısı kullanılmıştır. Bu Şekil 5.4' de görüldüğü gibi işlemcilerin bloklanmasına ve iletişim tamamlanıncaya kadar bir sonraki işlemciyi beklemesine neden olmaktadır. İlgili kod parçası

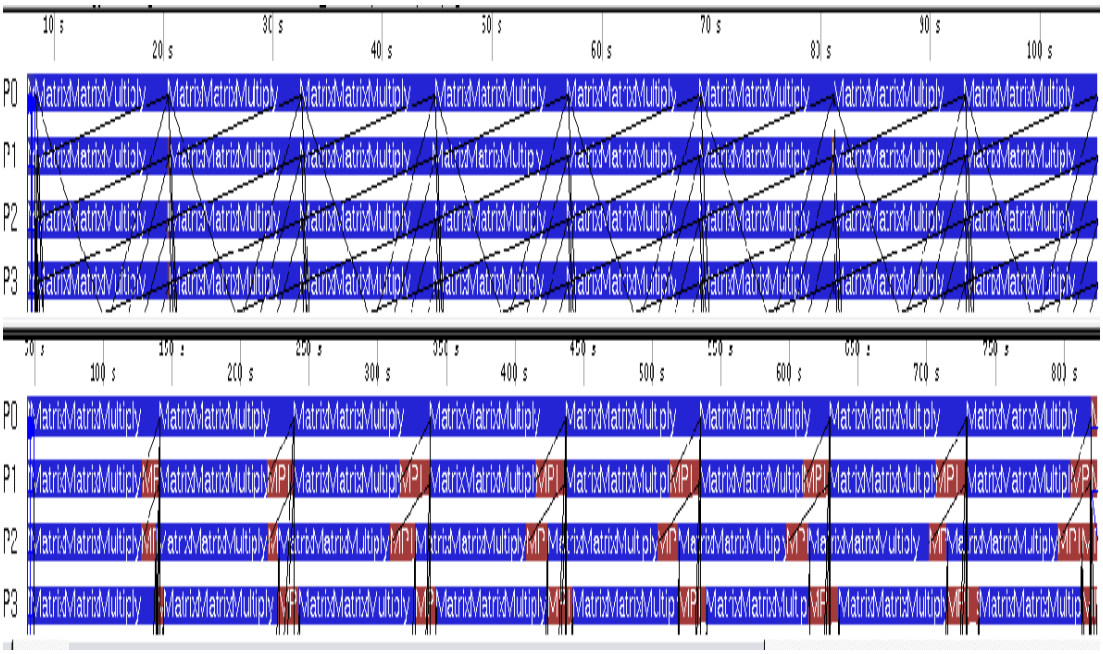
```
/* Ana hesaplama döngüsü */
for (i=0; i<grid->proc; i++) {
  MatrixMultiply(nlocal, a, b, c); /*c=c+a*b*/
 /* A matris blokunu bir sola kaydır */
 MPI_Sendrecv_replace(a, nlocal*nlocal,
MPI_DOUBLE,leftrank,1,rightrank,1,grid->comm_2d, &status);
 /* B matris blokunu bir yukarı kaydır */
 MPI_Sendrecv_replace(b, nlocal*nlocal, MPI_DOUBLE,uprank, 1, downrank, 1,
grid->comm_2d, &status);
}
/* Kod parçası burada sonlanır */
```

olarak verilebilir. İşlemcilerin bloklanmasına neden olan bu yapıyı değiştirmek için MPI bloksuz iletişim fonksiyonları olan *MPI_Isend* ve *MPI_Irecv* çağrıları kullanılabilir. Böylece bir işlemci yaptığı hesaplamalara devam edebilmek için diğerlerini beklemek zorunda kalmayacaktır. İlgili kod parçası

```
/* Ana hesaplama döngüsü */
for (i=0; i<grid->proc; i++) {
  MPI_Isend(a_buffers[i%2], nlocal*nlocal, ....);
  MPI_Isend(b_buffers[i%2], nlocal*nlocal, ....);
  MPI_Irecv(a_buffers[(i+1)%2], nlocal*nlocal, ....);
  MPI_Irecv(b_buffers[(i+1)%2], nlocal*nlocal, ....);
  /* c=c+a*b */
  MatrixMultiply(nlocal, a_buffers[i%2], b_buffers[i%2], c);
  for (j=0; j<4; j++)
    MPI_Wait(&reqs[j], &status);
 }
/* Kod parçası burada sonlanır */
```

olarak verilebilir.

Şekil 5.5' de optimizasyondan önce bloklu (alt resim) ve optimizasyondan sonra bloksuz (üst resim) test edilmiş Cannon uygulama CPU zamanları görülebilir. 8192 boyutlu problem için 64 işlemci kullanıldığında bloklu yapıya nazaran bloksuz yapının işlemcilerde beklemeye neden olmadığı görülebilir (Kırmızı yada koyu renkli bölgelerin yok olduğunu gözlemlemek yeterli).

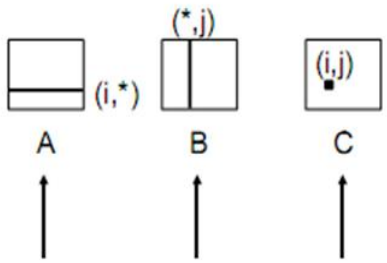


Şekil 5.5 : Optimizasyon öncesi ve sonrası Cannon matris çarpımı MPI iletişimi.

5.2.3 Seri Matris Çarpımında Cache Optimizasyonu (Base) Yapılması

C programlama dilinde diziler bellekte satır satır depolanıp cache'den de o yapıdan çekilmektedir. Yapılan ilk testlerde Cannon algoritması tarafından kullanılan seri matris çarpımı için iç-içe gerçirilmiş klasik [ijk] döngü yapısı kullanılmıştı. Bu her bir yinelemede sonuç matrisinin bir elemanının hesaplanabilmesi için B'nin her bir satırının baştan taranıp cache'den çekilmesini zorunlu kılar. Şekil 5.6' da bunu görmek mümkündür.

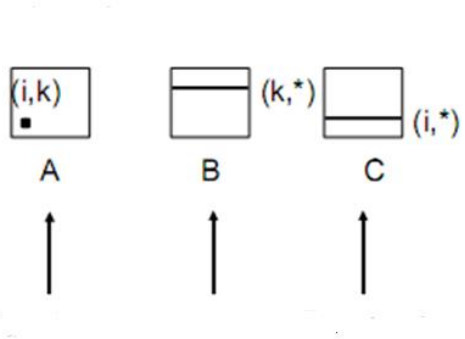
```
/* ijk */
for (i=0; i<n; i++)  {
  for (j=0; j<n; j++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```



Şekil 5.6 : ijk döngü yapısında matris çarpımı.

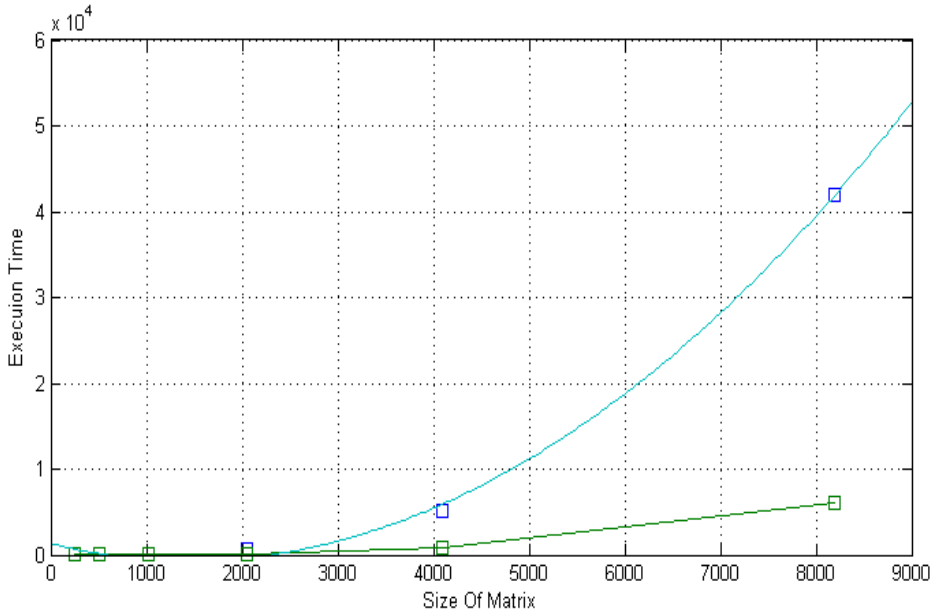
Döngü yapısında yapılacak ufak bir değişikle, döngülerin her seferinde matris satırlarını taramadan verileri cache'den çekmeleri mümkün olacaktır. Şekil 5.7' de [ikj] yapısına göre derlenmiş döngü yapısı görülebilir.

```
/* ikj */
for (i=0; i<n; i++) {
  for (k=0; k<n; k++) {
    r = a[i][k];
    for (j=0; j<n; j++)
      c[i][j] += r * b[k][j];
  }
}
```



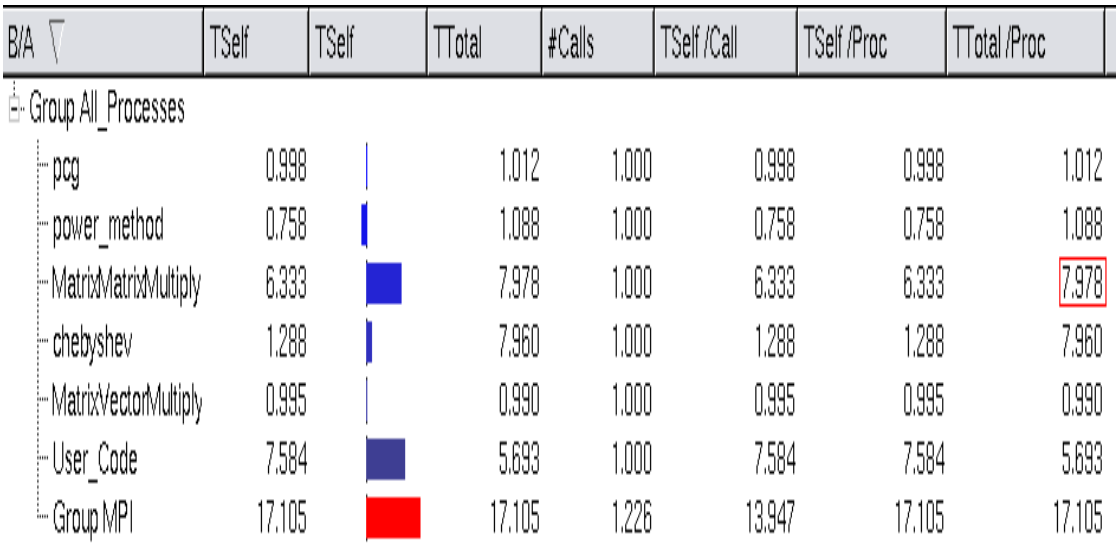
Şekil 5.7 : ikj döngü yapısında matris çarpımı.

Döngü değişikliği yapıldıktan sonra mevcut test oratımı üzerinde farklı boyuttaki matrisler ile bir performans test çalışması yapılmıştır. Şekil 5.8' de bu test çalışmasının sonuçları görülebilir. Buna göre ikj döngü yapısı ijk'ye göre CPU time verimliliği açısından daha iyi olduğu görülebilir. Özellikle matris boyutu artırıldığında her iki uygulama arasındaki performans farkının ciddi derecede açıldığını görmek mümkündür.



Şekil 5.8 : ijk (mavi) ve ikj (yeşil) döngü yapısındaki matrsi çarpımları.

Şekil 5.9' da optimize edilmiş yeni döngü yapısıyla gerçekleştirilmiş Cannon matris çarpımı ile ilgili test sonuçları görülmektedir. Bu şekilde 8192 boyutlu problemde optimizayondan önceki parallel matris çarpımı CPU zamanı maliyetinin optimizayon sonrası olan değerlere oranı görülmektedir. Optimizasyon sonrası 64 işlemci için CPU zamanı olarak yaklaşık 8 kat iyileştirme sağlanmıştır.



B/A	TSelf	TSelf	TTotal	#Calls	TSelf /Call	TSelf /Proc	TTotal /Proc
Group All_Processes							
pcg	0.998		1.012	1.000	0.998	0.998	1.012
power_method	0.758		1.088	1.000	0.758	0.758	1.088
MatrixMatrixMultiply	6.333		7.978	1.000	6.333	6.333	7.978
chebyshev	1.288		7.960	1.000	1.288	1.288	7.960
MatrixVectorMultiply	0.995		0.990	1.000	0.995	0.995	0.990
User_Code	7.584		5.693	1.000	7.584	7.584	5.693
Group MPI	17.105		17.105	1.226	13.947	17.105	17.105

Şekil 5.9 : Optimizayon öncesi / sonrası Cannon CPU zaman oranları.

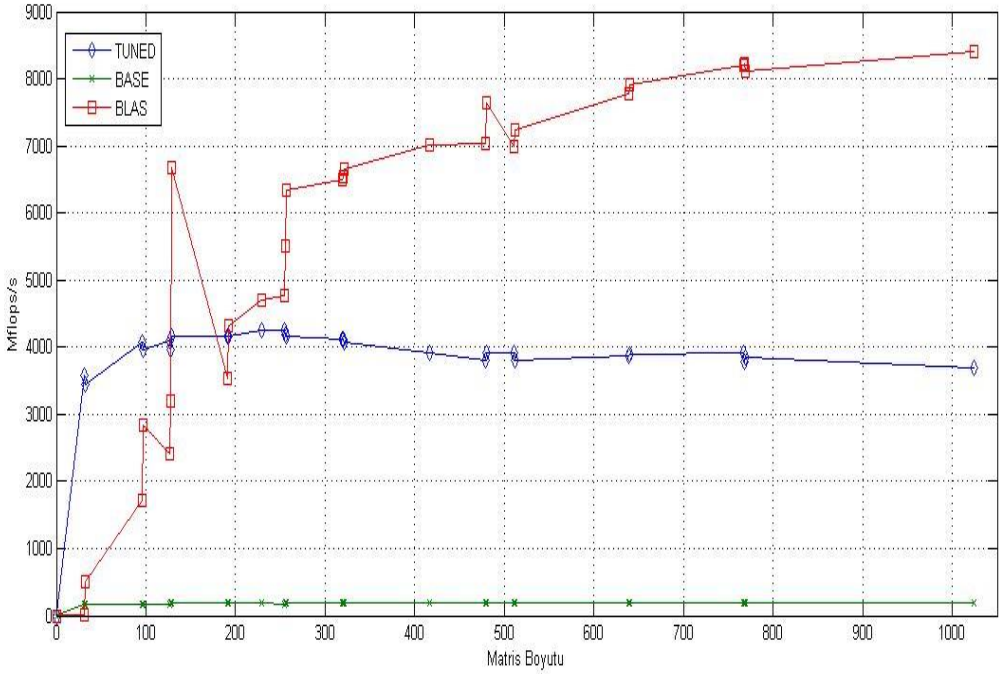
5.2.4 Seri Matris Çarpımının Bloklu Optimize (Tuned) ve BLAS Uygulaması

Bu optimizasyonu uygulamada kullanılan temel yaklaşım $C = C + A * B$ şeklinde gerşekleştirilecek matris çarpımı için 3 seviyeli bellek yapısına; register, L2 ve L3' e en uygun boyutta olacak ve yapılacak işlemi verimli kılacak blok yapısının bulunmasıdır.

Register bloklama işlemi bize C matrisini en alt seviyede bellek blokuna nasıl yerleştirildiğini yada dizildiğini ifade etmektedir. Örneğin 2x2 boyutlu register blokunun manası, C matrisini 2 satır ve 2 sütundan oluşan parçalar şeklinde hesaplayacağımız anlamına gelmektedir. Ek A.1 ' de bu hesaplamayı gerçekleştiren kod parçası bulunabilir. Yapılan çalışma kapsamında 1x1' de 8x8 parça boyutuna kadar performans testleri gerçekleştirilmiştir. Test sonuçları en verimli blok parça boyutunun 3x2 olduğunu göstermektedir.

İkinci aşama olarak L2 ve L3 cache'ler için en iyi blok sayısını seçmek gelmektedir. Bunun için L3 cache için en dışta bulunan döngü bloku 4 den 1024'e, L2 cache için içte bulunan döngü bloku 4'den L3'e olacak şekilde testler gerçekleştirilmiştir. Elde edilen sonuçlar 512x256 blok yapılı sistemin en iyi sonuç verdiği yönünde olmuştur. Elde edilen bu optimize versiyona *Tuned* adı verilmiştir. Fonksiyon *tuned_mm (L3,L2,M, A, B, C)* ismi ile adlandırılmış olup sırasıyla L3 blok, L2 blok, A matrisi, B matrisi ve C matrisi değerlerini almaktadır.

Bir diğer test fonksiyonu olarak BLAS 3. seviye hesaplama fonksiyonlarından olan *cblas_dgemm* kullanılmıştır.



Şekil 5.10 : Seri matris çarpımı Base, Tuned ve BLAS performans değerleri.

Şekil 5.10' da 3 farklı optimizasyon sonrası elde edilen matris çarpım sonuçlarının performans değerleri görülebilir. Matris boyutu 1000' e kadar yapılan testlerde en iyi performans'ın beklenildiği gibi platforma yönelik optimize edilmiş olan BLAS fonksiyonundan elde edildiğini görebiliriz. Bloklu matris optimizasyonu yapılan versiyonun verimliliğinin standart çarpım yöntemine göre oldukça iyileştiğini gözlemlemek mümkün.

5.2.5 Bloklu Çalışan Tüm MPI Çağrılarının Optimize Edilmesi

power_method() ve *PC_gradient()* fonksiyonları tarafından yinelemeler içinde çağrılan bloklu MPI yapıları bloksuz çağrılarla değiştirilebilinir. Örneğin,

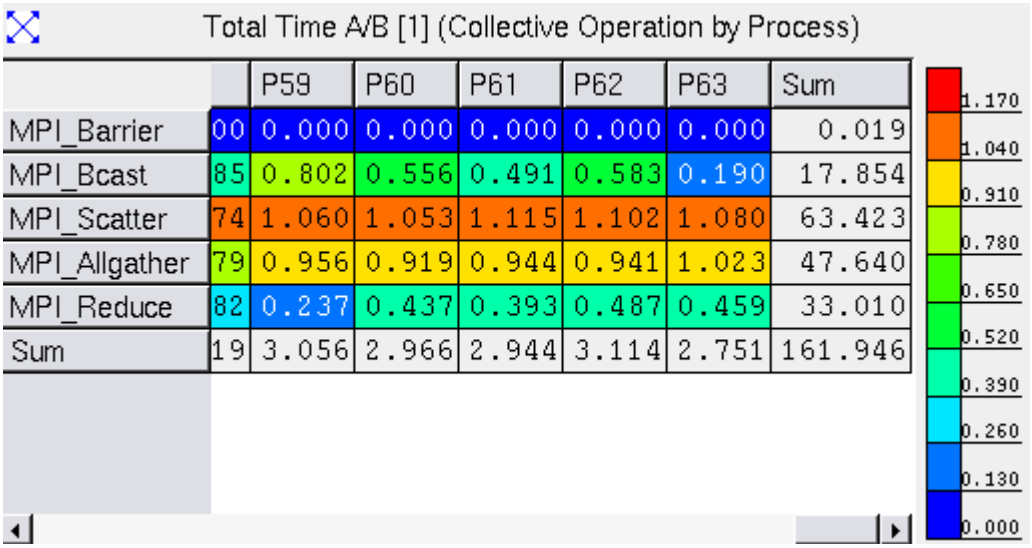
MPI_Reduce(e1_Local, &e1, 1, MPI_DOUBLE, ...);

MPI_Bcast(&e1, 1, MPI_DOUBLE, 0, ...);

kod parçası yineleme döngüleri içersinde farklı değerlerin tüm işlemciler üzerinde güncellenmesi için kullanılmaktadır. Bu yapıyı aynı değerleri güncellemek için,

MPI_Allreduce(e1_Local, &e1, 1, MPI_DOUBLE, ...);

çağrısıyla değiştirdiğimizde, veri iletişim verimliliğine yönelik önemli bir değişiklik yapılmış olur.



Şekil 5.11 : MPI kolektif çağrılarınnın, optimizasyonlar öncesinin sonrasına oranı.

Yapılan bütün bu optimizasyon değişikliklerinin uygulamaya olan yansıması Şekil 5.11' de verilmektedir. 8192 boyutlu problemde optimizasyonlar sonrası 64 işlemci için, tüm MPI iletişim ve kolektif çağrılarında ortalama olarak CPU zamanında $161.946/64 = 2.5$ kat iyleştirme sağlanmıştır.

6. SONUÇLAR VE TARTIŞMA

Bölüm 5' de detaylı olarak gerçekleştirilen optimizasyon işlemlerinden sonra farklı boyutlardaki problemler test edilmiştir. İlk olarak Base olarak tanımladığımız döngü değişimi ile optimize ettiğimiz seri matris çarpım fonksiyonunu kullanarak test sonuçları elde edilmiştir. Daha sonra Tuned ve BLAS versiyonlar için test sistemler kurulup detaylı performans analizleri yapılmıştır. Tüm testler için CPCG yönteminde ön koşullayıcı polinom derecesi 2, güç yöntemi iterasyon sayısı 50, CG yöntemi iterasyon sayısı olarak 450 seçilmiştir.

6.1 Test Sonuçları

6.1.1 Temel Optimizasyon Test Sonuçları

Bu test için Bölüm 5'de tanımlanan tüm optimizasyonlar kullanılmış olup Cannon algoritmasında kullanılan seri matris çarpım fonksiyonu için Base optimize fonksiyon kullanılmıştır. Çizelge 6.1' de CPCG yönetminin farklı problem boyutlarında, 2 boyutlu mesh topolojisinde 256 işlemciye kadar yapılan testlerinin CPU zamanları görülebilir.

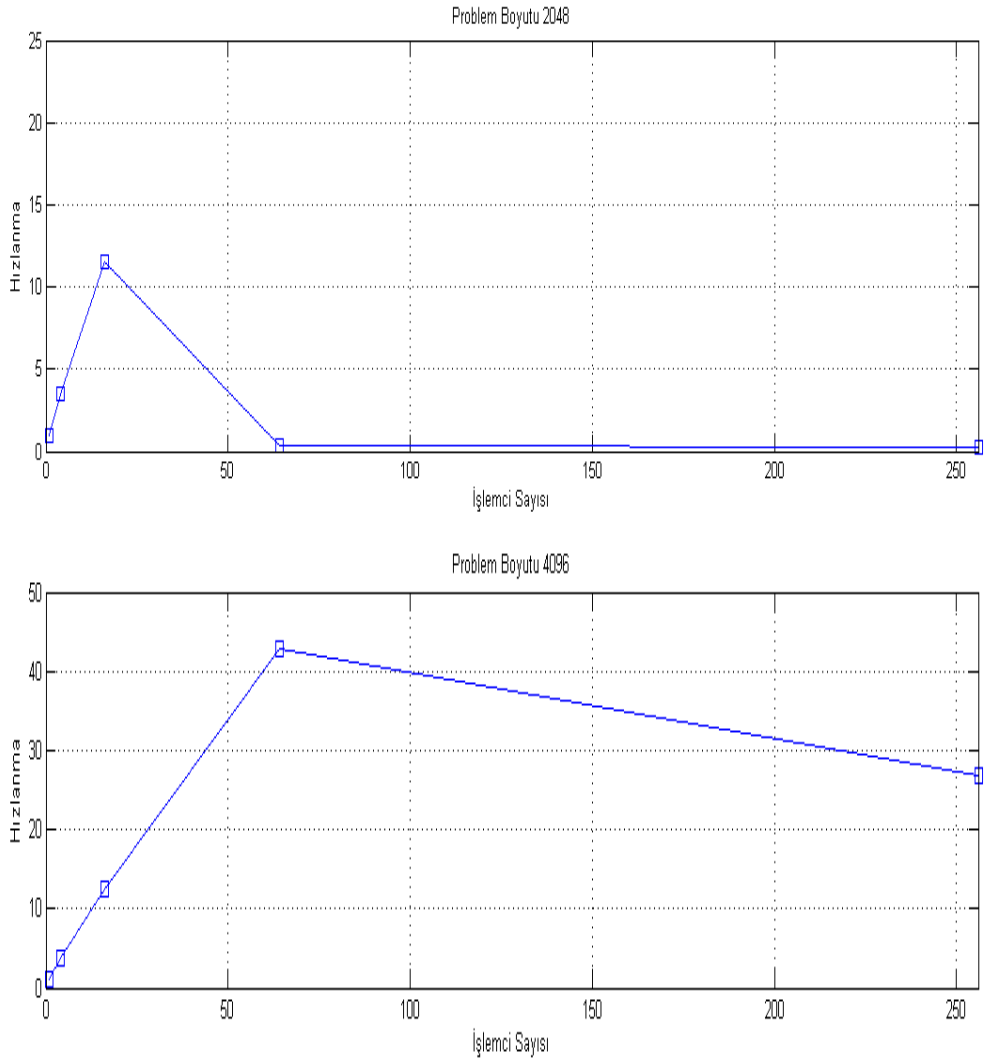
Çizelge 6.1 : Problem boyutu ve işlemci sayısına karşı CPU zamanları.

	2048	4096	8192	12800
1	136	1071	7731	31821
4	38	285	2034	8159
16	11,75	85	577	2305
64	356	25	154	600
256	521	40	64	178

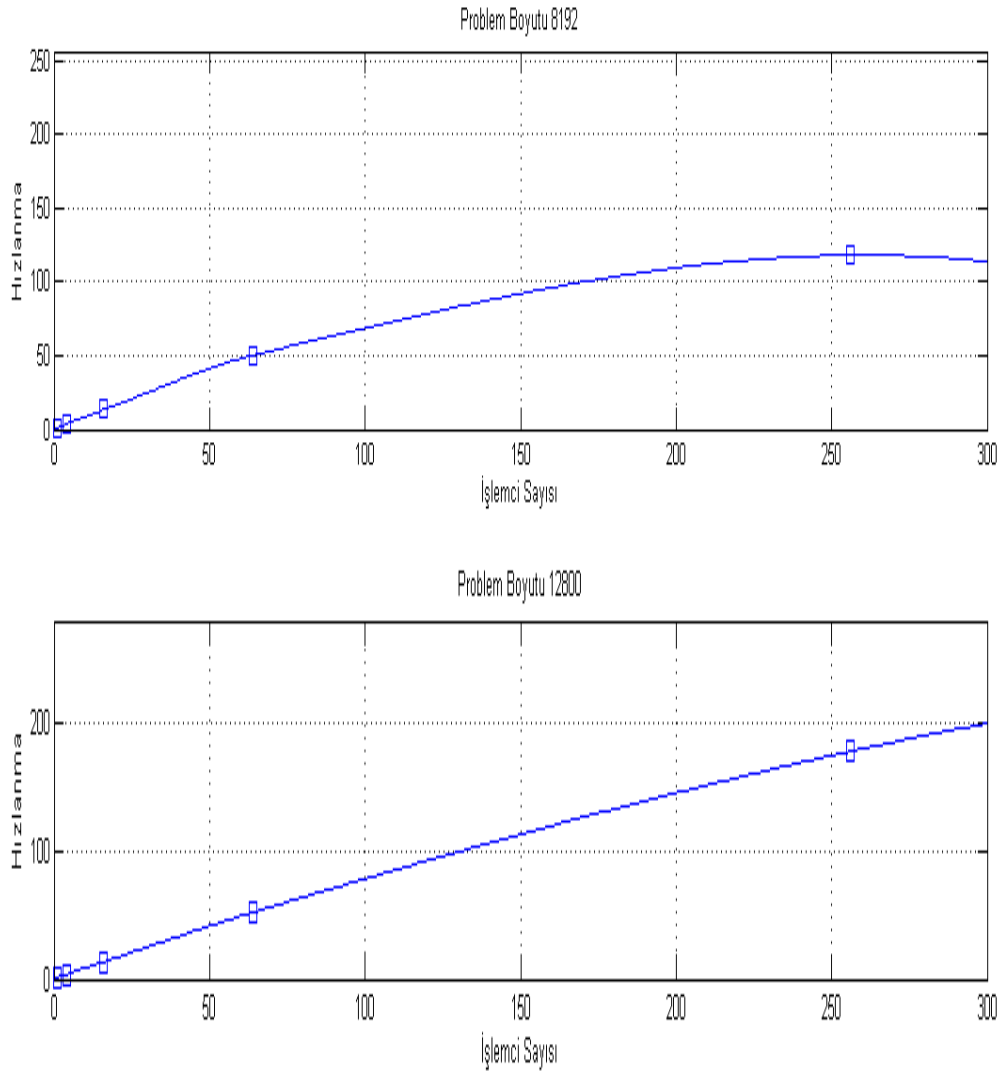
Çizelge 6.2 : Problem boyutu ve işlemci sayısına karşı hızlanma değerleri.

	2048	4096	8192	12800
1	1.0	1.0	1.0	1.0
4	3.5	3.75	3.8	3.9
16	11.5	12.6	13.4	13.8
64	0.4	42.8	50.2	53.0
256	0.2	26.7	120.1	178.2

Çizelge 6.2’ de bu CPU zamanlarına karşı gelen hızlanma değerleri görülebilir. Yapılan çalışma sonuçları Şekil 6.1 ve Şekil 6.2’ de bir araya getirildiğinde görüleceği gibi problem boyutu çok küçük olduğunda parallel algoritma büyük işlemci sayılarında ölçeklenirliğini kaybetmektedir. Bunun sebebi işlemciler arasında transfer edilecek veri mikarının çok sayıdaki işlemci arasında döndürülecek kadar büyük olmaması. Problem boyutu artırıldıkça işlemciler üzerindeki blok veri boyutları büyümekte ve bunun sonucu olarak işlemci sayısının artırılmasıyla orantılı olarak ölçeklenebilirliği ve hızlanma değerleri de artış göstermektedir.

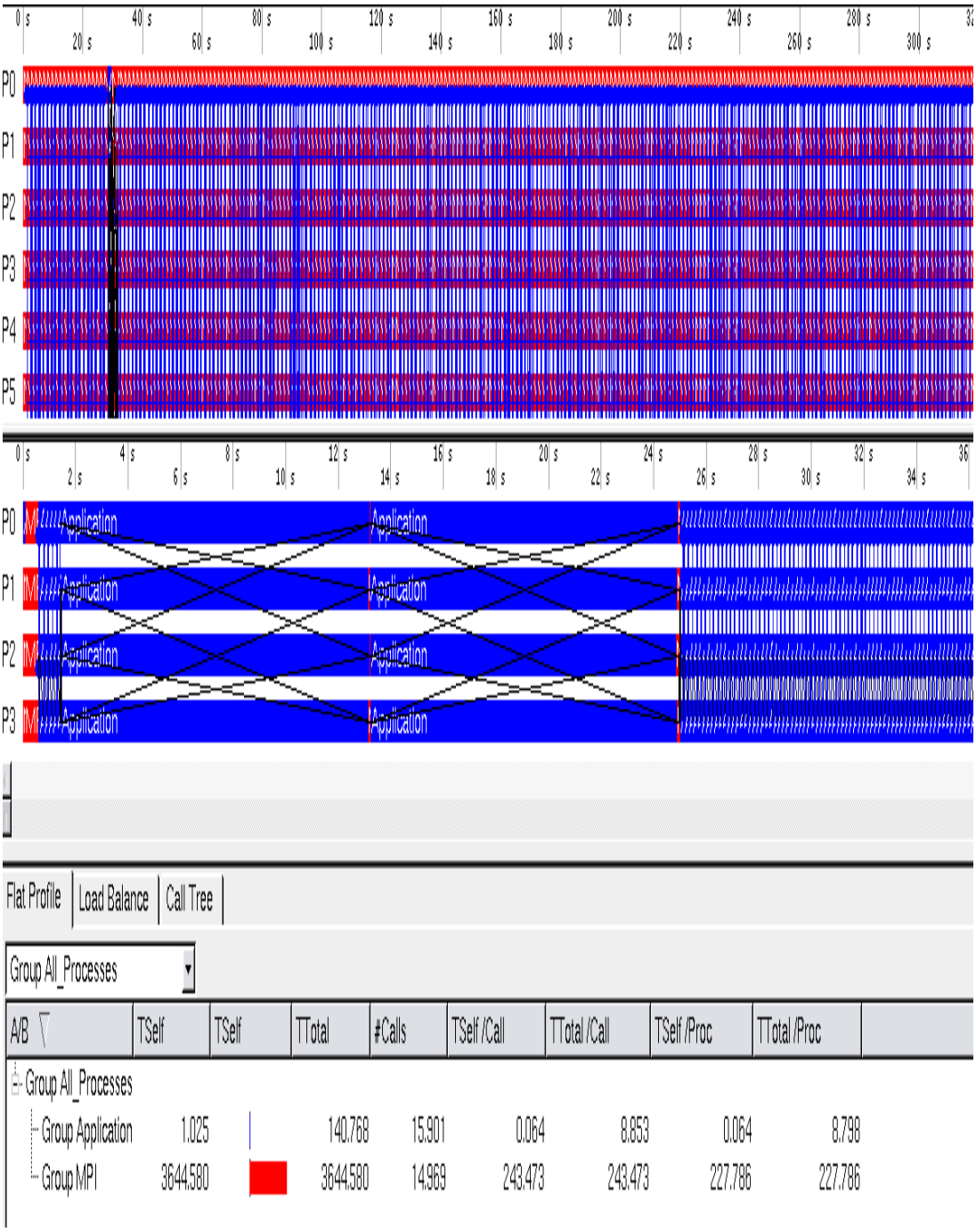


Şekil 6.1 : 2048 ve 4096 problem boyutları için hızlanma değerleri.



Şekil 6.2 : 8192 ve 12800 problem boyutları için hızlanma değerleri.

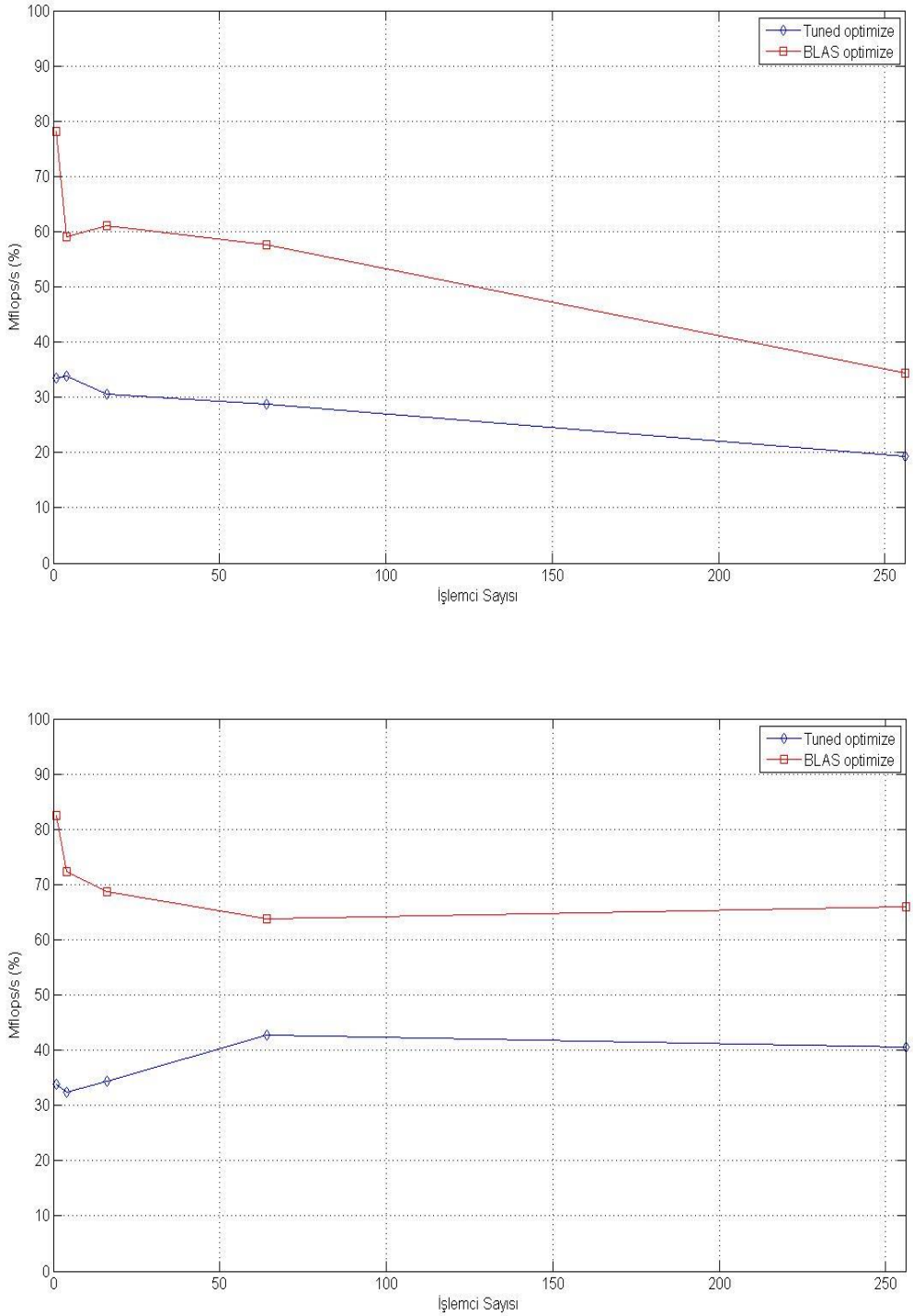
Şekil 6.1' de verilen grafikte hesaplanan sistem boyutları sırası ile 2048 ve 4096' dır. Belli bir işlemci sayısının üzerine çıkıldığında hızlanma değerinin büyük bir düşüşe geçtiği görülmektedir. Yukarıda da değinilmeye çalışılan bu saptamanın nedenleri Şekil 6.3' de daha net olarak gözlemlenebilir. 2048 boyutlu problemin 64 işlemci ile (üst resim) elde edilen sonuçlarının 4 işlemci ile (alt resim) elde edilenlerle karşılaştırılması bu resimde görülebilir. CPU üzerinde koşan uygulamanın performansında 8 kat bir iyileşme gözlenirken işlemciler arası iletişimi gerçekleştiren MPI kolektif çağrılarının 243 kat kötüleştiğini gözlemleyebiliyoruz.



Şekil 6.3 : 2048 boyutlu sistemin 4 ve 64 işlemcili test sonuçları.

6.1.2 Gelişmiş Test Sonuçları

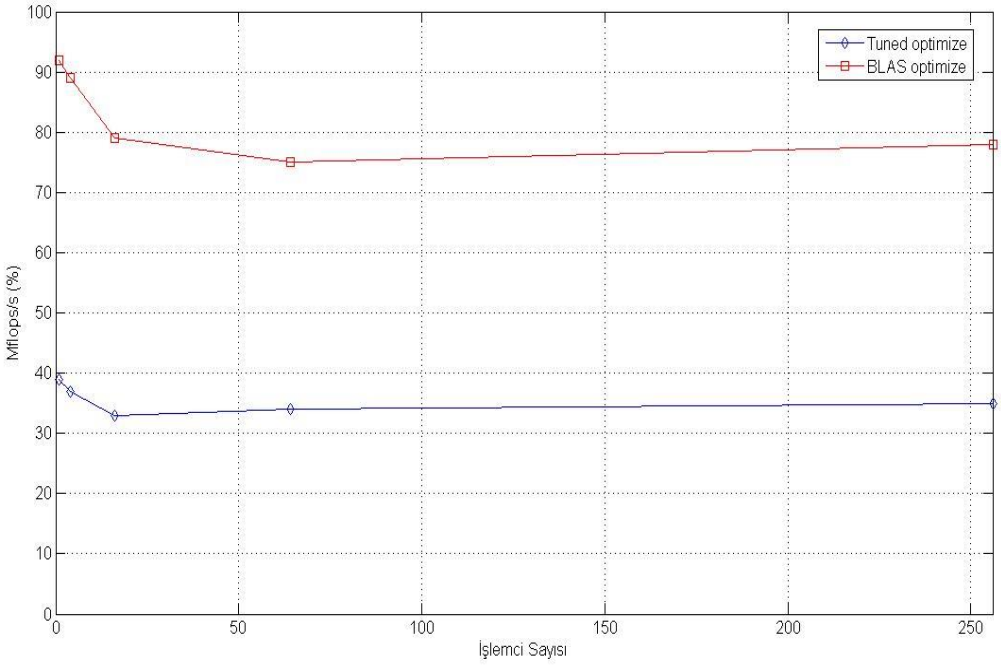
Bu bölümde Cannon matris çarpımı içinde çağrılan seri çarpım fonksiyonunun Tuned ve Blas versiyonları kullanılmıştır. Diğer optimizasyon teknikleri Bölüm 5'de anlatıldığı şekli ile uygulanmıştır. Sonuçlar Mflop ve hızlanma değerleri cinsinden detaylı olarak verilmektedir. Test 256 işlemciye kadar 2 boyutlu mesh topolojisi kullanılarak gerçekleştirilmiştir. Performans değerleri tek bir işlemciye karşılık gelen ortalama değerler olarak verilmektedir.



Şekil 6.4 : 3520 ve 7040 boyutlu problemin 256 işlemci ile performans değerleri.

Şekil 6.4 ve Şekil 6.5' de CPCG yöntemi için 3520, 7040 ve 14080 boyutlu problemlerin 256 adete kadar işlemciler ile yapılan testlerinin performans sonuçları görülebilir.

Problem boyutu arttıkça işlemciler üzerinde koşan kodun performans değerlerininde arttığını görebilmekteyiz. Şekillerde verilen performans değerleri tek bir işlemci üzerinde koşulan kodun ortalama performans değerinin teorik en büyük performans değerine oranın % olarak göstermektedir.



Şekil 6.5 : 14080 boyutlu problemin 256 işlemci ile performans değerleri.

Çizelge 6.3' de CPCG yönteminin 3520, 7040 ve 14080 boyutlu problemler için Tuned ve Blas optimizasyonların CPU zamanları görülebilir.

Çizelge 6.3 : CPCG yönteminin Tuned ve BLAS optimize CPU zamanları.

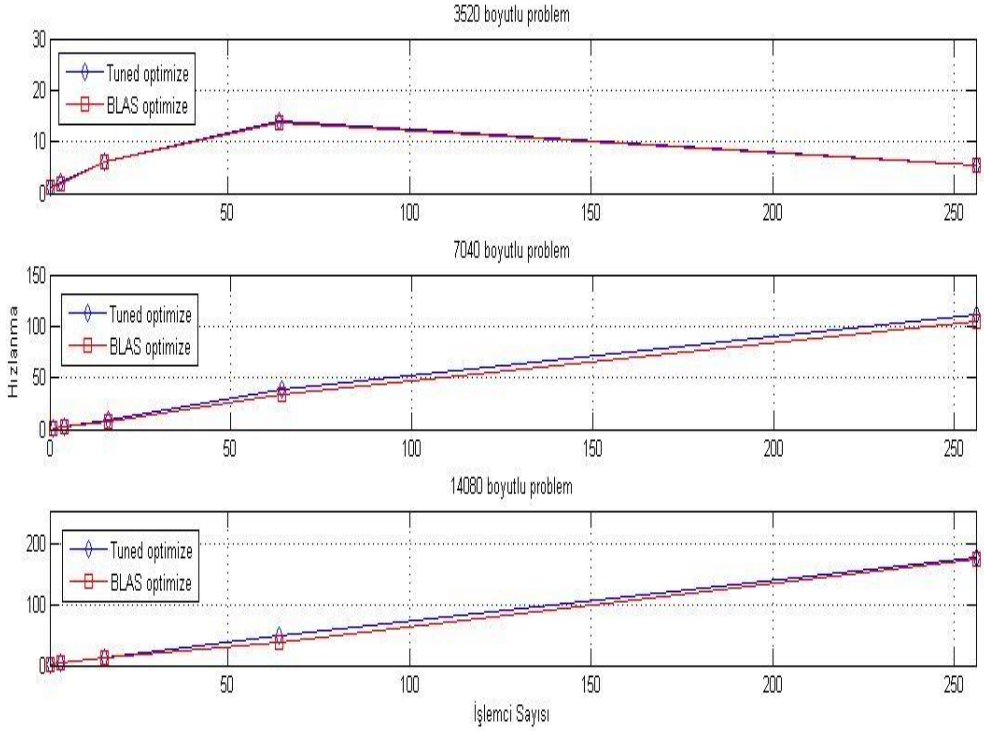
	Tuned			BLAS		
	3520	7040	14080	3520	7040	14080
1	99	510	3784	83	374	2448
4	48	216	1287	43	183	731
16	16	52	273	13	49	209
64	7	13	79	6	11	63
256	18	4.6	21	15	3.6	14

Çizelge 6.4' de CPCG yönteminin Çizelge 6.3' de verilen CPU zamanlarına karşılık gelen hızlanma değerleri verilmektedir.

Çizelge 6.4 : CPCG yönteminin Tuned ve BLAS optimize hızlanma değerleri.

	Tuned			BLAS		
	3520	7040	14080	3520	7040	14080
1	1.00	1.00	1.00	1.00	1.00	1.00
4	2.06	2.36	3.10	1.93	2.04	3.34
16	6.18	9.80	13.86	6.00	7.63	11.71
64	14.14	39.23	47.89	13.83	34.00	38.85
256	5.50	110.86	180.10	5.53	103.88	174.85

Şekil 6.6' da detaylı olarak hızlanma değer grafikleri bütün problem boyutları için görülebilir. Problem sayısı arttıkça problemin ölçeklenilebilirliğinin arttığı açıkça görülmektedir.



Şekil 6.6 : CPCG yönteminin test problemleri için hızlanma değerleri.

6.2 Algoritma Analizi

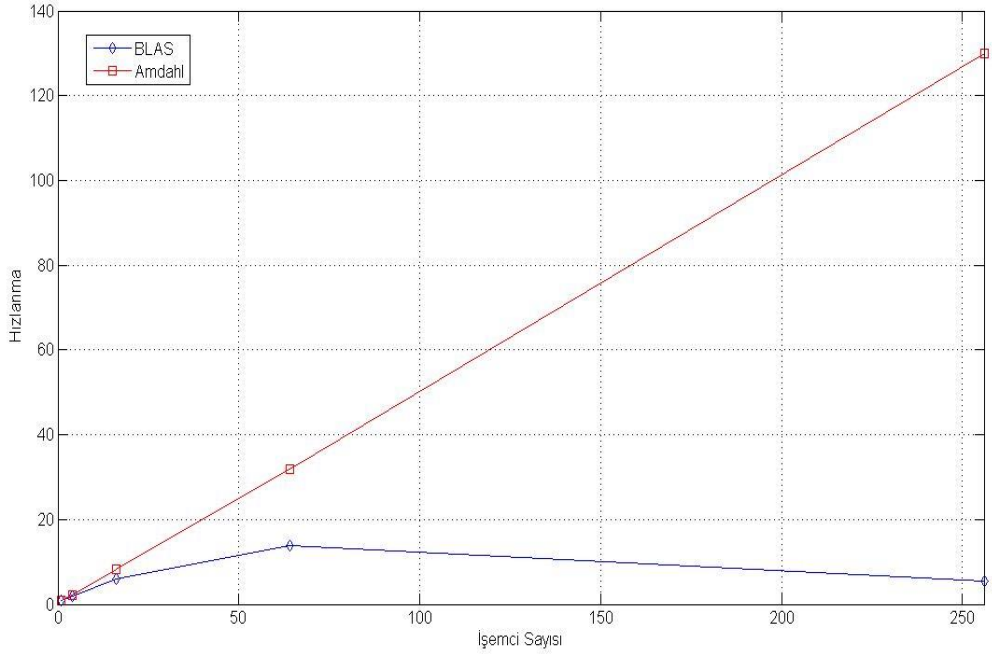
Test sonuçlarını daha iyi analiz edebilmek için öncelikle CPCG yönteminin daha ayrıntılı incelemekte fayda var. CG yönteminin zaman ihtiyacı (time complexity), n problemi teşkil eden matris boyutunu ve κ durum sayısı göstemek üzere $O(\sqrt{\kappa}n^2)$ olarak verilmektedir. Yine aynı yöntem için bellek ihtiyacı (space complexity) $O(n^2)$ olarak tanımlanabilir [3].

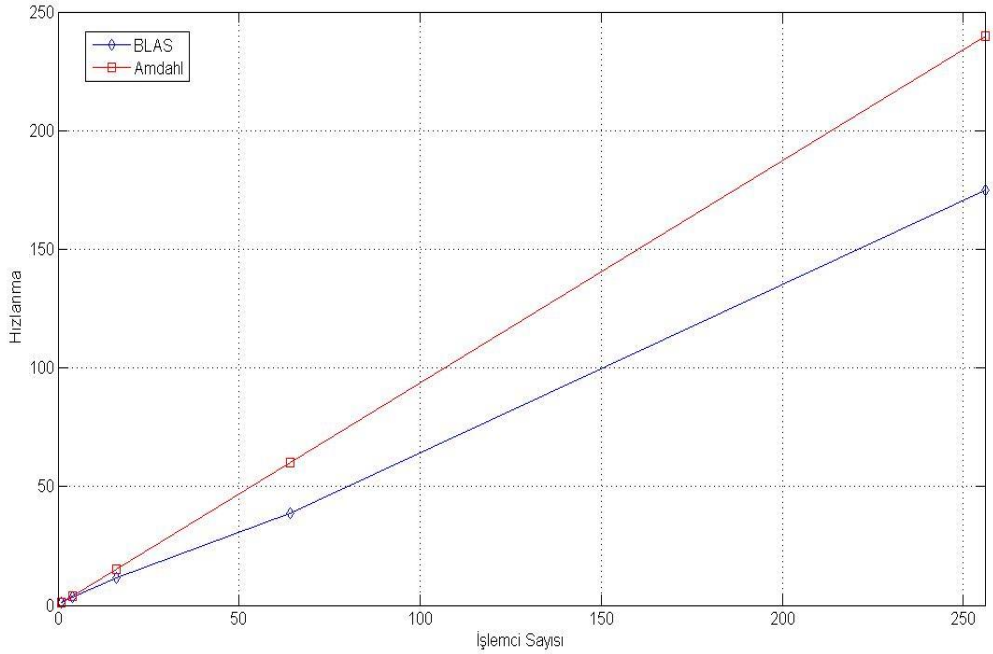
Verilen değerlerden görüldüğü üzere CG yönteminin baskın hesaplama çekirdeği olan matris-vektör çarpımı bu ihtiyaçları tanımlamaktadır. Yine aynı yaklaşımla ön koşullayıcı Chebyshev' in baskın hesaplama çekirdeğinin matris-matris çarpımı olduğu göz önüne alındığında zaman ve bellek ihtiyacı sırası ile $O(n^3)$ ve $O(2n^2)$ olarak verilebilir. Genel anlamı ile yöntem, depolama gerkeksinimi olarak klasik seri bir kod'da tam bir A katsayılar matrisi ve b sağ yan vektörünün bellek üzerinde saklanmasını gerektirir. Ama bu çalışmada ayrıntılandırılan ve Bölüm 4' de detaylı olarak anlatılan paralel programlama tekniği ile işlemci sayısı artırıldıkça problem için kullanılması gereken işlemci başına bellek miktarı azalmaktadır. Eğer matris boyutları NxN ve işlemci sayısı p ise ayrıştırılan matrislerin blok boyutları dolayısıyla her bir işlemci düğümünde depolanacak veri miktarı $\left(n/\sqrt{p}\right)x\left(n/\sqrt{p}\right)$ olacaktır. Aynı yaklaşım saklanacak olan sağ yan vektörü içinde düşünülebilir.

6.2.1 Amdahl Kanunu ve Verimlilik

Amdahl kanunu genel itibari ile, belli bir kısmı paralelleştirilmiş bir algoritmanın ulaşabileceği en yüksek hızlanma değerini tanımlamakta kullanılabilir. Pratikte bu değeri yaklaşık olarak bulabilmek için; $S_p = \frac{1}{(1-P)+\frac{P}{N}}$ formülü kullanılabilir.

Burada P algoritmanın paralelleştirilebilen kısmının işlem zamanı olarak yüzdesi olup *(1-P)* seri kısmın işlem zamanı yüzdesi olarak ifade edilir. N işlemci sayısına karşılık gelir. Bu çalışmada kullanılan ve Çizelge 4.1' de detaylı olarak verilen çekirdek fonksiyonlar içersinde paralel uygulaması olmayan fonksiyonlar detaylı olarak görülebilir. Mevcut algoritma üzerinde Amdahl kanununu uygulayabilmek için paralel uygulaması olmayan çekirdek fonksiyonların toplam zamanın tüm işlem zamanına oranı alınarak P değeri bulunabilir. Şekil 6.7' de 3520 ve 14080 boyutlu problemler için Amdahl teorik tepe hızlanma değerleri ile testler sonucunda elde edilen değerler karşılaştırılmıştır.



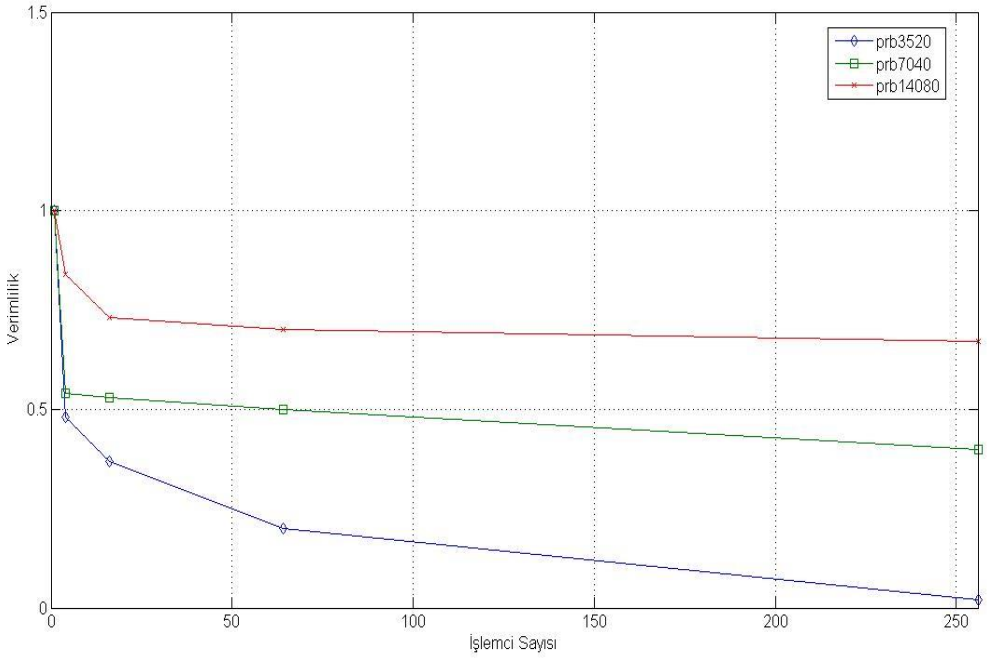


Şekil 6.7 : 3520 ve 14080 boyutlu problemler için Amdahl karşılaştırması.

Şekil 6.7' de problem boyutu artırıldıkça Amdahl teorik değerlerine yaklaşıldığı gözlemlenebilmektedir.

Verimlilik açısından analiz etmek istersek; $E_p = S_p/p$ formülü ile ifade etmek mümkündür. Burada S_p, p işlemciye karşılık gelen hızlanma değerini göstermektedir.

Teorik olarak $E_p \leq 1$ olacaktır. 1' e eşit olma durumu algoritmanını mükemmel şekilde paralel programlamaya uygun olduğunu ifade edecektir. Şekil 6.8' de sırasıyla 3520, 7040 ve 14080 boyutlu problemlerin verimlilik değerleri grafik olarak analiz edilebilir



Şekil 6.8 : Farklı problem boyutları için CPCG algoritmasının verimliliği.

Şekil 6.8' de görülebileceği gibi algoritmaya verilen problem boyutu büyüdükçe paralel olarak çalışan kodun verimliliğide artmaktadır.

6.3 Tartışma

Bir çok mühendislik disiplinine ait problemler büyük boyutlu $Ax = b$ denklem sistemlerinin çözümünü gerektirmektedir. Bu denklemlerin çözümü için geleneksel seri kodların kullanılması işlem hızı, zamanı, performansı ve işlemin ölçeklenebilirliği açısından problem teşkil etmektedir. Bu handikaptan kurtulabilmek için paralel programlama büyük boyutlu problemler için verimli bir yol olarak kabul edilebilir. Bu bağlamda yapılan bu çalışmada CG yöntemi için Chebyshev ön koşullayıcısının parallel uygulaması tanıtılmıştır. Problem boyutu artırıldıkça yöntemin ölçeklenebilirliğinin iyileştiği gösterilmiştir.

Yine problem boyutunun artırılması ile birlikte, uygun optimizasyon araç ve yöntemleri kullanılarak, yöntem için kullanılan paralel kodun performans değerinin

arttığı gösterilmiştir. Ön koşullu CG yöntemi matris-vektör çarpımı, iç çarpım ve vektör güncellemesi gibi basit vektör işlemleri gerektirdiğinden paralel işlem için oldukça uygundur .

Bölüm 3' de detaylı olarak anlatıldığı üzere Chebyshev ön koşullayıcısı CG yöntemi ile birlikte kullanıldığında iterasyon sayısını önemli ölçüde azaltmaktadır. Dolayısıyla Chebyshev ön koşullayıcısının CG yönetmi ile birlikte paralel programlama ortamında kullanılmasının çok iyi derecede ölçeklenilebilir olduğu çeşitli seviyelerde optimize edilmiş kodun çalıştırılması ile Çizelge 6.2, Çizelge 6.4, Şekil 6.1, Şekil 6.2 ve Şekil 6.6' da gösterilmiştir.

Bu çalışma aynı zamanda paralel program geliştirme esnasında karşılaşılan optimizasyon problemleri üzerine yoğunlaşmış ve daha verimli bir paralel kod yazılabilmesi için yapılması gereken değişiklikleri deneysel olarak gerçekleştirip bunları mevcut kod üzerine uygulamıştır.

Yapılan çalışmaları elde edilen sonuçlar açısından özetlemek gerekirse, $Ax = b$ formundaki lineer sistemleri parallel bir ortamda çözerken;

- Doğrudan çözüm yöntemleri yerine (Gauss, LU ayrıştırma vb.) özellikle büyük boyutlu sistemler için yinelemeli yöntemleri kullanmak gerekebilir.
- Ön koşullayıcı olarak LU ayrıştırmasına dayanan yöntemler yerine, matris-matris çarpımına dayanan (bu çalışmada matris değerli Chebyshev ön koşullayıcısı verilmiştir) ön koşullayıcının, parallel ortamdaki avantajlarından dolayı kullanılması gerekebilir.
- Chebyshev ön koşullayıcısının derecesinin artırılmasıyla toplam CPU zamanınında artmasına rağmen, ön koşullu CG yönteminin yineleme sayısı kayda değer bir derecede azalmaktadır. Bundan dolayı, eğer ön koşullayıcı farklı sağ yan vektörleri için defalarca kullanılacaksa (örneğin farklı sınır değerler için diferansiyel denklemler) büyük dereceli Chebysev ön koşullayıcısını hesaplamak ve bu problemler için defalarca kullanıp toplam CPU zamanından kazanmak verimli olacaktır (bizim çalışmamızda polinom derecesi 2 seçilip tek matris-matris çarpımı yapılmıştır). Eğer ön koşullayıcı bir kez kullanılacaksa Chebyshev polinom derecesi ufak olmalıdır.

KAYNAKLAR

[1] **Dağ H. and Semlyen A.,** 2003: A new preconditioned conjugate gradient power flow. *IEEE Transactions on Power Sytems*, 18(4), November

[2] **Hestenes M. R. and Stiefel E.,** 1952: Methods of conjugate gradients for solving lineer systems. J .*Res. Nat. Bur. Stand*., 49:409-436

[3] **Shewchkuk J. R.,** 1994: An introduction to the conjugate gradient method without the agonizing pain. School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.

[4] **Saad Y. and Schultz M.,** 1986: GMRES: A generalized minimal residual algorithm for solving non-symmetric linear systems. *SIAM Journal on Scientific and Statical Computing*, (7):856-869, July.

[5] **Galiana F. D., Javidi H., and McFee S.,** 1992: On the applications of a pre-conditioned conjugate gradient algorithm to power network analysis, *IEEE Transactions on Power Systems*, (2):629-636.

[6] **Mori H., Tanaka H., and Kanno J.,** 1994: A preconditioned fast decoupled power flow method of contingency screening, *IEEE Transactions on Power Systems*, 11(1):357-363, February.

[7] **Kershaw D.,** 1978: The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations. *Journal of Computational Physics*, (36):43-65.

[8] **Manteuffel T.,** 1980: An incomplete factorization technique for positive definite linear systems. *Mathematics of Computation*, (34):473-492, April.

[9] **Dağ H. and Alvarado F. L.,** 1994: The effect of ordering on the preconditioned conjugate gradient method for power system applications. *In Proceedings of the North American Power Symposium*, Manhattan, September.

[10] **Alves A. B., Asada E. N. and Monticelli A.,** 1997: Critical evaluation of direct and iterative methods for solving ax=b systems in power flow calculations and contingency analysis. *In the IEEE/PES Summer Meeting*, Berlin, Germany, July.

[11] **Barret R., Berry M. and Chan F.,** 1994: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, *SIAM*, Philadelphia, PA.

[12] **Vuik C.,** 2002: Iterative solution methods. Technical report, Delft University of Technology, Faculty of Information Systems and Technology.

[13] **Hageman L. and Young D.,** 1981: Applied iterative methods, *Academic Press*, New York.

[14] **Young D.,** 1971: Iterative Solution of Large Linear Systems, Academic Press, New York.

[15] **Quinn C. M.,** 2004: Parallel Programming in C With MPI and OpenMP, McGraw-Hill Inc, New York.

EKLER

EK A.1 : 2x2 Register Bloklu Matris Çarpım Kod Parçası

EK A.1

```
void reg_2x2 (const int bck, const int N, const double *A, const double *B, double
*C) {

int i, j, k;
int bi, bj, bk;
int loop1, loop2, loop3;
int r  = 2; /* register blok satır sayısı */
int c  = 2; /* register sutun satır sayısı */
loop1  = N/r;
loop2  = N/c;
loop3  = N; /* matris boyutu */

for (bi=0; bi<loop1; bi++) {
        i = bi*r;
        const register double *Ai_ = A + i*bck;
        for (bj=0; bj<loop2; bj++) {
              j = bj*c;
              const register double *B_j = B + j*bck;
              register double cij_1 = C[((j+0)*bck)+i+0];
              register double cij_2 = C[((j+0)*bck)+i+1];
              register double cij_3 = C[((j+1)*bck)+i+0];
              register double cij_4 = C[((j+1)*bck)+i+1];
             for (k=0; k<K; ++k) {
                    cij_1 += Ai_[k+0*bck]*B_j[k+0*bck];
                    cij_2 += Ai_[k+1*bck]*B_j[k+0*bck];
                    cij_3 += Ai_[k+0*bck]*B_j[k+1*bck];
                    cij_4 += Ai_[k+1*bck]*B_j[k+1*bck];
             }/*En iç Döngü Sonu */
         C[((j+0)*bck)+i+0] = cij_1 ;
         C[((j+0)*bck)+i+1] = cij_2 ;
         C[((j+1)*bck)+i+0] = cij_3 ;
         C[((j+1)*bck)+i+1] = cij_4 ;
        }
}/*En dış Döngü Sonu */

}/*Fonksiyon Sonu */
```

ÖZGEÇMİŞ

Ad Soyad: ÇAĞATAY AKÇADOĞAN

Doğum Yeri ve Tarihi: İSTANBUL – 14.11.1977

Lisans Üniversite: İSTANBUL TEKNİK ÜNİVERSİTESİ

Yayın Listesi:

- **Akçadoğan Ç.** and Dağ H., 2003: A Parallel Implementation of Chebyshev Preconditioned Conjugate Gradient Method. *IEEE Computer Soceity – ISPDC'03*, October 13-14, 2003, Ljubljana, Slovenia.