Seyrek Alt Üçgen Matris Çözümü için Graf Dönüşümü ve Özelleşmiş Kod Üretimi

Buse Yılmaz[1], Didem Unat[1]

1. Bilgisayar Mühendisliği Bölümü
Koç Üniversitesi
{byilmaz,dunat}@ku.edu.tr

Öz

Seyrek Alt Üçgen Matris Çözümü, birçok bilim ve mühendislik uygulamasında, seyrek doğrusal cebir sistemlerin çözümünde kullanılan önemli bir yöntemdir. Seyrek Alt Üçgen Matris Çözümü'nü günümüz mimarilerinde paraleleştirilmek zorludur. Hesaplamalar arasındaki bağımlılıklardan kaynaklı sınırlı paralellik ve hesaplamaların düzensiz ölçekte olması, etkili bir yük dengeleme ve senkronizasyon mekanizması yaklaşımı ihtiyacı doğurmaktadır. Bu çalışmada seyrek alt üçgen matrisin sınırlı parallelik gösteren parçalarının paralellik derecesini, yeniden yazma yöntemiyle graf dönüşümü uygulayarak artıran, ve bu sayede senkronizasyon bariyeri sayısını azaltan, ek olarak, özelleşmiş kod üreterek, alana özgü optimizasyonları etkinleştiren bir çerçeve öneriyoruz.

1. Giriş

Seyrek doğrusal sistemler, yer kabuğu biliminden fizik ve kimyaya kadar, hesaplamalı akışkanlar dinamiği, rezervuar simülasyonu ve sonlu eleman modellemesi gibi bilim ve mühendisliğin çeşitli uygulamalarında bulunur. Seyrek Alt Üçgen Matris Çözümü, seyrek doğrusal cebir sistemleri için doğrudan ve yinelemeli yöntemler, ve LU, QR ve Cholesky seyrek matris ayrıştırma yöntemleri gibi, birçok sayısal çözüm sistemi için yapı taşıdır ve toplam yürütme süresinin önemli bir bölümünü oluşturur. Seyrek Alt Üçgen Matris Çözümü'nü paralelleştirme yöntemleri geçmişte kapsamlı bir şekilde incelenmiştir fakat yine de saniyede muazzam miktarda hesaplama sağlayan günümüzün son derece paralel ve özelleşmiş mimarilerinde bile zorlu bir problem olmaya devam etmektedir. Seyrek Alt Üçgen Matris Çözümü'nün temel zorlukları şunlardır: **1)** Hesaplamalar arasındaki bağımlılıklar nedeniyle sınırlı paralellik sergiler, **2)** Hesaplamalar düzensiz ölçektedir: iş parçacıklarına atanan iş yükleri düşük tanelidir ve matrisin seyreklik yapısına bağlı olarak iş yükü tane ölçekleri farklılık gösterir, **3) 1** ve **2** önemli senkronizasyon ve yönetim yükü ile sonuçlanır, **4)** Hesaplamalı bağımlılıklar dikkatli bir analiz gerektirir ve düşük taneli iş yükleri nedeniyle verimli bir yük dengeleme yaklaşımı gerekir.

Belirtilen zorluklar, Seyrek Alt Üçgen Matris Çözümü'nün doğasından ve matrisin seyreklik yapısından kaynaklanmaktadır: seyreklik yapısı genellikle matris boyunca farklılık gösterir. Bu nedenle, seyrek bir matris, farklı paralellik dereceleri sergileyen parçalara sahiptir. Seyrek Alt Üçgen Matris Çözümü'nün, çok az paralellik sergileyen parçaları günümüzün oldukça paralel mimarilerinden yararlanamaz. Bu parçalar genellikle seyrek matrisin yalnızca birkaç satırını barındırırlar ve bağımlılıklar nedeniyle sadece tek veya çok az sayıda işlemci çekirdeği kullanılabilir.

Bu çalışmada, bahsedilen zorlukları çözümlemek için:

- Düşük parallelik derecesine sahip seyrek alt üçgen matris parçalarının paraleliğini graf dönüşümü ile artırarak, seyreklik yapısını daha homojen hale getiren,
- Senkronizasyon noktalarına olan ihtiyacı azaltan,
- Çok çekirdekli işlemcilerde, Seyrek Alt Üçgen Matris Çözümü problemi için özelleşmiş kod üreterek, alana özel optimizasyonları etkinleştiren,

bir bağımlılık grafı dönüşümü ve özelleşmiş kod üretme çerçevesi geliştirmeyi öneriyoruz.

Paralellik derecesinin az olduğu parçalarda paralellik derecesini artırarak seyreklik yapısını değiştirmek için, her satırın temsil ettiği denklemleri yeniden yazarak, bağımlılık grafını dönüştürmeyi amaçlıyoruz. Satırları yeniden yazmak, satırların bağımlılıklarını ortadan kaldırmayı sağlayarak, onları graf içindeki yerlerini değiştirmenin güvenli bir yoludur. Seyrek Alt Üçgen Matris Çözümü için özelleşmiş kod üretmek, bellek erişimlerini azaltmak, dolaylı indekslemeden kurtulmak ve aritmetik optimizasyonlar gibi çeşitli optimizasyonları mümkün kılar.

2. Seyrek Alt Üçgen Matris Çözümü ve Literatür Özeti

Seyrek Alt Üçgen Matris Çözümü, seyrek bir alt üçgen matris *(L)* ve yoğun bir vektörü *(b)* alır ve *x* yoğun vektörü için $Lx = b$ doğrusal denklemini çözer.

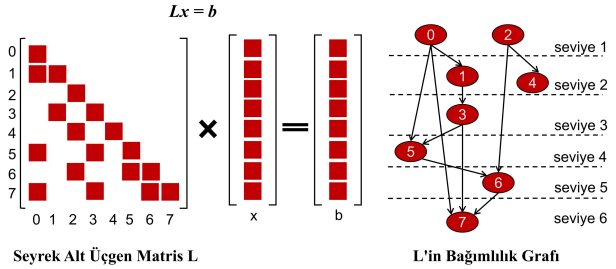
Şekil 1, $Lx = b$ işlemini göstermektedir. Seyrek bir alt üçgen matris olan L'nin bağımlılık grafı, düğümlerin satırları ve kenarların düğümler arasındaki bağımlılıkları temsil ettiği, yönlendirilmiş döngüsel olmayan bir graftır. Bu bağımlılık grafı, seviyelere bölünmüş haliyle sağda görülmektedir.

Literatür, Seyrek Alt Üçgen Matris Çözümü'nü optimize etme yaklaşımları açısından zengindir: seviye kümesi oluşturma yöntemleri [1,2,3], birbirine bağımlılığı olmayan satırları seviyelere gruplandırır. Bir seviyedeki satırlar paralel olarak yürütülen iş yüklerine ayrılır ve seviyeler birbiri ardına (seri olarak) yürütülür. [3]'te, CPU'lar için seviye kümesi oluşturma yöntemine dayalı bir algoritma sunulmuştur. Verilerin zayıf uzamsal yerelliğinin neden olduğu performans sorunu için katsayı matrisi üzerinde yeniden sıralama yöntemi kullanılmaktadır. Seyrek Alt Üçgen Matris Çözümü'ndeki hesaplamaların düşük taneli yapısı nedeniyle kendi kendine zamanlama yönteminin daha kötü performans göstereceğine [3]'te dikkat çekilmiştir.

Seviye kümesi oluşturma yöntemi [4] 'de ve [5] 'de NVIDIA GPU'lar için kullanılmıştır. Son zamanlarda yapılan bir çalışma [6], Sunway mimarisi için bu yaklaşımı, Seyrek

Seviye Döşemesi adlı, veri yerelliğini iyileştiren yeni bir veri düzenini de önererek genişletti.

Kendi kendine zamanlama yöntemleri, seviye kümesi oluşturma yöntemine alternatif olarak, her seviyenin sonundaki

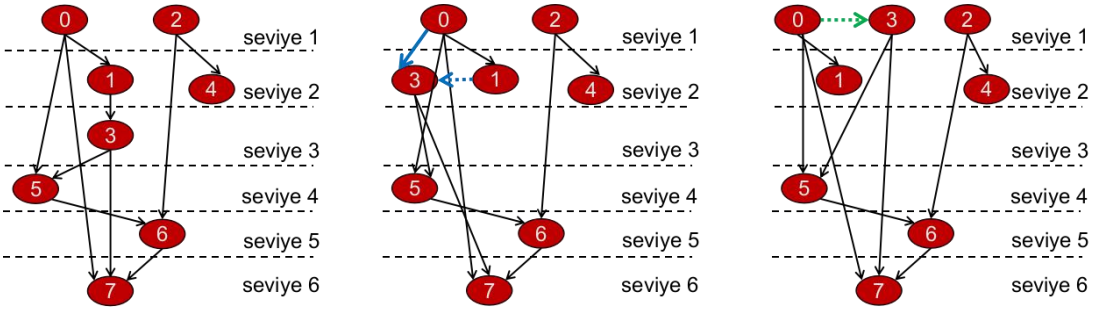


Şekil 1: Seyrek Alt Üçgen Matris Çözümü işlemi $Lx = b$. L'i bağımlılık grafı sağda seviyelere bölünmüş şekilde gösterilmektedir.

bariyer ihtiyacını ortadan kaldırmak için ortaya çıktı. Bu yaklaşımın ilk örnekleri CPU'lar içindi [7]. Daha sonra, GPU'lar üzerinde çeşitli uygulamalar önerildi [8,9,10,11]. [9] 'da, seviyeleri belirlemek için paralel bir topolojik sıralama algoritmasına ek olarak, her bir iplikçiğin yalnızca kendi iş yükündeki satırların bağımlılıklarını beklediği, sayaç tabanlı bir zamanlama mekanizması kullanılmıştır. [10] 'da, kendi kendine zamanlama yönteminin her bir satırın bağımlılık derecesi üzerine kurulduğu, basit bir ön işleme aşaması önerilmektedir.

Seyrek Alt Üçgen Matris Çözümü'nü optimize etmekle ilgili bir diğer önemli araştırma alanı ise derleyicilerdir. [14] 'de, yazarlar girdi seyrek matrisin özelliklerini inceleyerek onun için özelleşmiş bir matrisi yeniden sıralama kodu üretir. Sympiler [15], matrisi derleme işlemi esnasında sembolik olarak analiz eder ve çeşitli seyrek matris işlemleri için alana özgü kod üretir. Bu bildiride önerilen özelleşmiş kod üreteci, Sympiler'a benzer alana özgü optimizasyonlara ek olarak bellek erişimlerinin değerlerini sabitler olarak koda gömerek bellek erişimlerini azaltmayı amaçlamaktadır.

Graf renklendirme, ek ön işleme gerektiren bir NP-Tam problemdir ve Seyrek Alt Üçgen Matris Çözümü optimizasyonunda kullanılmaktadır. [16, 17, 18] 'da, etkin yük dengeleme için graf renklendirme kullanılmıştır. [16], CPU'lar üzerinde kullanılmak üzere, cebirsel blok çok renkli sıralama yöntemini önerir. [17] ve [18] ise graf renklendirmesini GPU üzerinde kullanır. Blok-diyagonel tabanlı yöntemler, Seyrek Alt Üçgen Matris Çözümü'nü iyileştirmek için kulanılan bir diğer yaklaşımdır ve daha çok CPU'larda uygulanan bir yöntemdir. Matris, diyagonal etrafında ve dışında yerelliği iyileştirmek için yoğun bloklar oluşturacak şekilde yeniden düzenlenir [19, 20, 21]. [21], yoğun BLAS işlemlerini diyagonal dışı yoğun blokları hesaplamak için kullanmaktadır. Yoğun matris hesaplamaları, seyrek emsallerinden daha yüksek paralelliğe sahiptir ve bu yüzden seyrek matrislerinde yoğun bloklar oluşturmak performans iyilestirmede etili bir yöntemdir. Bu teknikler, büyük ölçüde matris seyreklik yapısına bağlıdır.



a) Seyrek Alt Üçgen L'nin Bağımlılık Grafı

b) Satır 3'ün satır 1'e olan bağımlılığı kırıldı, satır 3 seviye 3'ten seviye 2'ye çıkarıldı

c) Satır 3'ün satır 0'e olan bağımlılığı kırıldı, satır 3 seviye 2'ten seviye 1'ye çıkarıldı

Şekil 2:

Satır denklemleri: x[0] = **b[0] / L[0][0],** x[1] = **(b[1] - L[1][0] * x[0]) / L[1][1]**
x[3] = (b[3] - L[3][1] * x[1]) / L[3][3]

Yeniden yazma işlemi:

b) x[3] = (b[3] - L[3][1] * ((**b[1] - L[1][0] * x[0]) / L[1][1]**)) / L[3][3]

c) x[3] = (b[3] - L[3][1] * ((**b[1] - L[1][0] * (b[0] / L[0][0])) / L[1][1]**)) / L[3][3]

[12]'de hem seviye kümesi oluşturma hem de kendi kendine zamanlama yöntemlerini kullanan hibrit bir yaklaşım CPU'lar için geliştirilmiştir. Bir önceki çalışmamızda [13], [12]'ye benzer bir yaklaşımı benimsedik. Her iki çalışma da gereksiz bağımlılıkları ortadan kaldırarak SpTRSV için CPU'larda senkronizasyon noktalarını azaltmayı amaçlamaktadır.

3. Yeniden Yazma Yöntemi

Bağımlılık grafı, her satırın temsil ettiği denklemlerin yeniden yazılmasıyla dönüştürülerek, ince seviyeler (sadece birkaç satırlı seviyeler) kalınlaştırılabilir, ek olarak, ince seviyeler tamamen kaldırılarak, senkronizasyon bariyerlerinin sayısı azaltılabilir.

x[3] = (b[3] - L[3][1] * ((b[1] - L[1][0] * (b[0] / L[0][0])) / L[1][1])) / L[3][3] (c)

A= b[0] / L[0][0] (1)
x[3] = (b[3] - L[3][1] * ((b[1] - L[1][0] * A) / L[1][1])) / L[3][3] (c')
x[3] = (b[3] - L[3][1] * (b[1]/ L[1][1] - L[1][0] * A / L[1][1]) / L[3][3] (c'')
B = b[1]/ L[1][1] (2)
C =L[1][0] * A / L[1][1] (3)
x[3] = (b[3] - L[3][1] * (B - C)) / L[3][3] (c''')
D = L[3][1] * (B - C) (4)
x[3] = (b[3] - D) / L[3][3] (c'''')
b[3]' = b[3] - D (5)
x[3] = **b[3]'** / L[3][3] (c''''')

Şekil 3: Şekil 2'deki **c)** eşitliğinin düzenlenmesi ve *Lx = b* formatına uygun hale getirilmesi. b vektör girdileri (b[t] ile ifade edilen) bu işlem sonucunda güncellenir. **(1-5)** eşitlikleri düzenlemenin kolay okunması amacıyla gösterilmektedir.

Dolayısıyla, paralelliği artırılan kalınlaşmış seviyelerde atıl çekirdekler kullanılabilir. Böylelikle, normalde bağımlılıklar nedeniyle daha sonra hesaplanacak olan satırlar, yeniden yazıldıklarında, artan kayan nokta işlemleri (FLOPS) ve bellek erişimi pahasına, çok daha önce hesaplanmış olur.

Şekil 2, satırların yeniden yazılmasına dayalı graf dönüştürme yaklaşımımızı göstermektedir. Bu örnekte satır 3'e 2 kere yeniden yazma işlemi uygulanmıştır. Satır 0 ve satır 1'in denklemleri yeşil ve mavi renkle gösterilmektedir. Birinci yeniden yazma işlemi de satır 1'i kullandığından mavi, ikincisi ise satır 0'ı kullandığından yeşil renkle gösterilmiştir. **a)** orijinal bağımlılık grafını, **b)** ve **c)** ise yeniden yazma işlemlerini göstermektedir. Renkli oklardan kesikli oklar kırılan bağları, kesikli olmayan ise yeni kurulan bağı gösterir.

a) 3. satırın 1. satıra ve 1. satırın da 0. satıra bağımlılığı vardır, **b)** x[3]'ü hesaplayan denklemde x[1]'i, x[1]'in denklemiyle değiştirerek, 3. satırın 1. satıra olan bağımlılığı kırılır ve artık 3. satır 1. satıra değil, 1. satırın bağımlı olduğu 0. satıra bağımlı hale gelir. Bu işlemin sonucu olarak, 3. satır, seviye 3'ten seviye 2'ye kaydırılır, seviye 3 artık boştur, **c)** işlemi tekrar ederek, 3. satır, seviye 1'e kaydırılır ve artık herhangi bir satıra bağlı değildir. Seviye 3 boş olduğundan senkronizasyon bariyeriyle beraber kaldırılabilir.

Belirtilmesi gereken önemli bir husus, **c)**'de belirtilen yeni eşitliğin henüz düzenlenmemiş ve *Lx = b* formatında olmayışıdır. Bu düzenleme ilerleyen zamanlarda yeniden yazma işlemine eklenecektir. **c)**'de belirtilen eşitliğin düzenlenmiş hali Şekil 3'te gösterilmiştir. Bu düzenleme sonucunda b vektör girdileri güncellenmekte ve FLOP sayısı azalmaktadır.

4. Matris Analiz Modülü ve Özelleşmiş Kod Üreteci

Matris analiz modülü, girdi matris'i analiz eder ve matris özelliklerini çıkarır. Satır sayısı, matris girdi sayısı, seviye başına düşen toplam bellek erişim sayısı, analiz modülünce toplanan, matrise özgü özelliklerden birkaçıdır.

Matris analizi tamamlandığında, toplanan bilgi, seviye kümesi yöntemini kullanan ve girdi matrisine özel kod üreten özelleşmiş kod üretecine gönderilir.

Geliştirdiğimiz prototip özelleşmiş kod üreteci henüz sadece bellek erişimlerini azaltma ve dolaylı indekslemeden kurtulma (sadece yeniden yazılan satırlar için) optimizasyonlarını uygulamaktadır. Yük dengesi ayarlama, aritmetik optimizasyonlar ve yoğun bloklar oluşturarak yoğun BLAS işlemleri kullanma gibi optimizasyonlar henüz eklenmemiştir. Ek olarak, şu an yeniden yazma yöntemi manuel olarak yapılmakta, dolayısıyla yöntemin uygulanacağı satırlar programcı tarafından seçilmektedir. Özelleşmiş kod üreteci C++'da yazılmıştır ve üretilen kod da C++ kodudur.

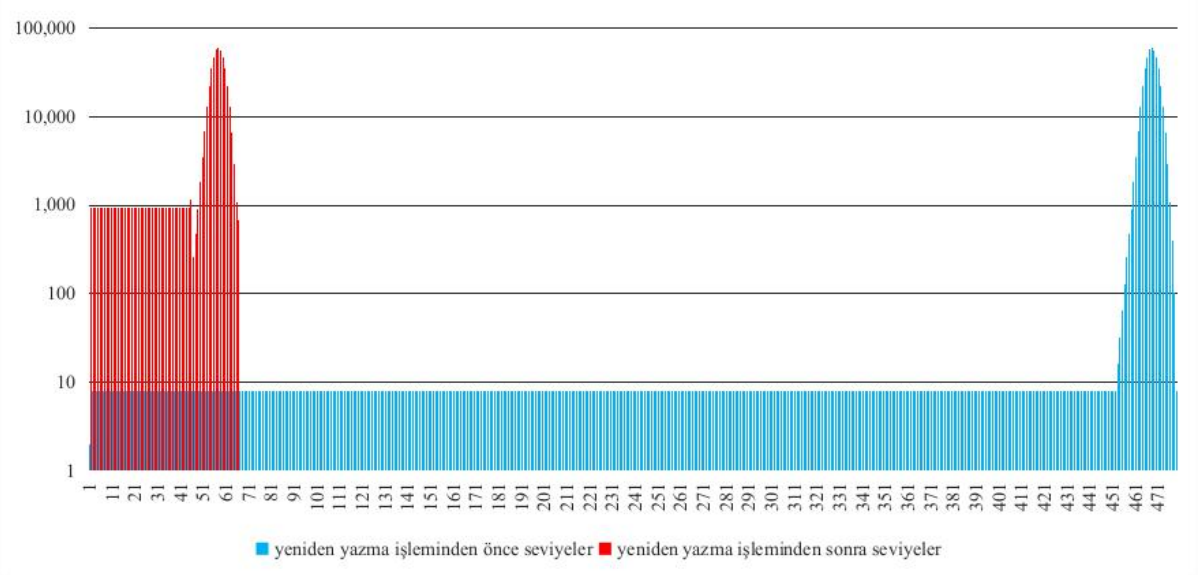
5. Performans Ölçümü

Yeniden yazma yönteminin bağımlılık grafındaki etkilerini gözlemlemek için iki performans deneyi yaptık. Bu deneyler için Suit Seyrek Matris Koleksiyonu'ndan [22] lung2 isimli matris'i seçildi. Lung2, 109,460 satıra ve 492,564 girdiye sahiptir. Seviye kümesi oluşturma yöntemi uygulandığında, 478 seviyeye, dolayısıyla 477 senkronizasyon bariyerine sahiptir ve bu seviyelerin %94'ü sadece 2 satırdan oluşmaktadır. Bu nedenle bu ince seviyeler parallelik bakımından oldukça fakirdirler.

Deneylerde, çift soketli bir Xeon Westmere makina kullandıldı. Westmere mimarisi, 2.53GHz çekirdek hızına sahip 12 çekirdek (24 iplik) ve 96GB DDR + 16 MCDRAM RAM'e sahiptir. Derleyici olarak clang 5.0. [23] kullanıldı.

Bölüm 4'te belirtildiği gibi, prototip henüz sadece yeniden yazılan satırlar için bellek erişimlerini azaltma ve dolaylı indekslemeden kurtulma optimizasyonlarını uygulamaktadır. Yük dengesi ayarlama gibi önemli optimizasyonlar henüz eklenmemiştir.

Birinci deneyde, geliştirdiğimiz prototip özelleşmiş kod üretecinin performansını, yeniden yazma yöntemi kullanmadan ve üretilen kodu seri yürüterek ölçtük. 10 iterasyonun ortalaması 1.98 ms olarak ölçüldü. Karşılaştırma yaptığımız, seviye kümesi oluşturma yöntemi uygulanan ve seri yürütülen kodun 10 iterasyonunun ortalaması 1.14 ms olarak ölçüldü. Prototipin performansının biraz düşük kalmasının sebebi, **1)** çok uzun bir kod üretilmesi, **2)** seri yürütmede, özelleşmiş kod üretiminin sağlayabileceği optimizasyonlardan yalnızca bellek erişimlerini matris girdileri



Şekil 4: Yeniden yazma işleminin FLOPS ve senkronizasyon bariyeri sayısı üstündeki etkileri.

için azaltma optimizasyonunun kullanılmış olması, **3)** prototip'in parallel yürütme için yazılmış olması ve seri yürütmede aslında gerek olmayan fonksiyon kullanımlarının performası düşürmesi. Bu deneyin amacı, üretilen koda eklenecek optimizasyonların fayda oranlarını gözlemlemek için bir kod iskeleti oluşturmaktır.

İkinci deneyde ise yeniden yazma yönteminin getiri ve götürülerini gözlemledik. Şekil 4'te görüldüğü gibi, ince seviyelere yeniden yazmayı uyguladık, çoğunu tamamen kaldırarak üst seviyelere kaydırdık. Yeniden yazma işleminin sonucunda 478 seviye 66'ya düşerek senkronizasyon bariyerlerinin %86'sını kaldırıldı. Gerçekleştirilecek toplam FLOP miktarı ise yalnızca % 10 arttı.

Seri yürütülerek gerçekleştirilen deneyde, 10 iterasyonun ortalaması 2.06 ms olarak ölçüldü. Prototip kod üretecinin önerilen optimizasyonların çok büyük bir kısmını uygulamıyor oluşu, etkin bir yük dengeleme yöntemiyle parallel yürütme yapılmadığı ve eşitliklerin düzenlenmediği göz önünde bulundurulduğunda, bu deney:

- Yeniden yazma işleminin senkronizasyon bariyerlerinin kaldırılması açısından ümit verici olduğunu,
- Paralel mimarilerde ve paralellikten yoksun parçalar nedeniyle hızlandırıcılarda zayıf performans sağlayan matrislerin, artık daha az sayıda fakat kalın seviyeler ile daha iyi performans sağlayabileceğini,
- Senkronizasyon bariyerlerinin sayısındaki azalmadan yararlanmanın ve daha fazla çekirdek kullanmanın mümkün olduğu, FLOP'lardaki artış kaynaklı performans maliyetini aşmasının oldukça muhtemel olduğunu, sonucun performans kazancı olacağını,
- Başta etkin bir yük dengeleme yöntemi olmak üzere, Bölüm 4'te sunulan optimizasyonların performans artışı için kritik önem taşıdığını göstermektedir.

6. Sonuç

Bu çalışmada Seyrek Alt Üçgen Matris Çözümü'nün performans optimizasyonu için yeni ve özgün bir yöntem olan, satırları yeniden yazma yöntemini öneriyoruz. Seyrek bir alt üçgen matrisin parallelik derecesi az olan parçalarını bu yöntemle dönüştürerek, bu parçaların parallelik derecesini artırmayı amaçlıyoruz. Böylece, graf dönüşümü, satır bağımlılıklarını esneterek ya da tamamen ortadan kaldırarak, daha çok çekirdek kullanımına olanak sağlar. Buna ek olarak, ince seviyeler bu yöntemle tamamen ortadan kaldırılarak senkronizasyon bariyeri ihtiyacını azaltmış olur. Önerdiğimiz yeniden yazma yöntemini özelleşmiş kod üreten ve bu sayede alana özgü optimizasyonları etkinleştiren bir kod üreteci ile pekiştiriyoruz. Performans sonuçları yeniden yazma yönteminin Seyrek Alt Üçgen Matris Çözümü'nün performans optimizasyonu için umut verici olduğunu göstermektedir.

7. Kaynakça

[1] Edward Anderson ve Yousef Saad. "Solving Sparse Triangular Linear Systems on Parallel Computers". *International Journal of High Speed Computing 1*, 1989.

[2] Joel H. Saltz. "Aggregation Methods for Solving Sparse Triangular Systems on Multiprocessors". *SIAM J. Sci. Stat. Comput. 11, 1*. 1990.

[3] Edward Rothberg ve Anoop Gupta. "Parallel ICCG on a hierarchical memory multiprocessor — Addressing the triangular solve bottleneck". *Parallel Comput. 18,7.* 1992.

[4] Maxim Naumov. "Parallel Solution of Sparse Triangular Linear Systems" *in the Preconditioned Iterative Methods on the GPU. Technical Report,* 2011.

[5] Ruipeng Li ve Yousef Saad. "GPU-accelerated preconditioned iterative linear solvers." *The Journal of Supercomputing 63, 2.* 2013.

[6] Xinliang Wang, Weifeng Liu, Wei Xue, ve Li Wu. "swSpTRSV: A Fast Sparse Triangular Solve with Sparse

Level Tile Layout on Sunway Architectures." *SIGPLAN Not. 53, 1.* 2018.

[7] Steven W. Hammond ve Robert Schreiber. "Efficient ICCG on a Shared Memory Multiprocessor". *International Journal of High Speed Computing 04*, 01 1992 .

[8] José I. Aliaga, Ernesto Dufrechou, Pablo Ezzatti, and Enrique S. Quintana-Ortí. "Accelerating the task/data-parallel version of ILUPACK's BiCG in multi-CPU/GPU configurations". *Parallel Comput. 85, 79 – 87,* 2019.

[9] Ruipeng Li. "On Parallel Solution Of Sparse Triangular Linear Systems in CUDA". *Technical Report. arXiv:1710.04985v1,* 2017.

[10] Weifeng Liu, Ang Li, Jonathan Hogg, Iain S. Duff, ve Brian Vinter. "A Synchronization-Free Algorithm for Parallel Sparse Triangular Solves". *In Proceedings of the 22Nd International Conference on Euro-Par 2016: Parallel Processing- Volume 9833. Springer-Verlag New York, Inc., New York, NY, USA, 617–630,* 2016.

[11] Weifeng Liu, Ang Li, Jonathan D. Hogg, Iain S. Duff, ve Brian Vinter. "Fast synchronization-free algorithms for parallel sparse triangular solves with multiple right-hand sides". *Concurrency and Computation: Practice and Experience 29, 21,* 2017.

[12] Jongsoo Park, Mikhail Smelyanskiy, Narayanan Sundaram, ve Pradeep Dubey. "Sparsifying Synchronization for High-Performance Shared-Memory Sparse Triangular Solver." *In Proceedings of the 29th International Conference on Super-computing - Volume 8488 (ISC 2014). Springer-Verlag New York, Inc., New York, NY, USA, 124–140,* 2014.

[13] Buse Yılmaz, Buğra Sipahioğlu, Najeeb Ahmad, ve Didem Unat. "Adaptive Level Binning: A New Algorithm for Solving Sparse Triangular Systems." *In Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region (HPCAsia2020). Association for Computing Machinery, New York, NY, USA, 188–198,* 2020.

[14] H. Rong, J. Park, L. Xiang, T. A. Anderson, ve M. Smelyanskiy. "Sparso : Context-driven optimizations of sparse linear algebra." *In 2016 International Conference on Parallel Architecture and Compilation Techniques (PACT). 247–259,* 2016.

[15] Kazem Cheshmi, Shoaib Kamil, Michelle Mills Strout, ve Maryam Mehri Dehnavi. "Sympiler: Transforming Sparse Matrix Codes by Decoupling Symbolic Analysis." *In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17). ACM, New York, USA, Article 13, 13 pages,* 2017.

[16] T. Iwashita, H. Nakashima, ve Y. Takahashi. "Algebraic Block Multi-Color Ordering Method for Parallel Multi-Threaded Sparse Triangular Solver in ICCG Method." *In 2012 IEEE 26th International Parallel and Distributed Processing Symposium. 474–483,* 2012.

[17] Maxim Naumov, Patrice Castonguay, ve Jonathan Cohen. "Parallel Graph Coloring with Applications to the Incomplete-LU Factorization on the GPU." *Technical Report NVR-2015-001 NVIDIA Research,* 2015.

[18] B. Suchoski, C. Severn, M. Shantharam, ve P. Raghavan. " Adapting Sparse Triangular Solution to GPUs." *In 2012 41st International Conference on Parallel Processing Workshops. 140–148,* 2012.

[19] Jan Mayer. "Parallel algorithms for solving linear systems with sparse triangular matrices." *Computing 86, 4 ,* 2009.

[20] Barry Smith ve Hong Zhang. "Sparse Triangular Solves for ILU Revisited: Data Layout Crucial to Better Performance." *Int. J. High Perform. Comput. Appl. 25, 4 (Nov. 2011), 386–391,* 2011.

[21] Ehsan Totoni, Michael T. Heath, ve Laxmikant V. Kale. 2014. "Structure-adaptive parallel solution of sparse triangular linear systems." *Parallel Comput. 40, 9, 454 – 470,* 2014.

[22] Timothy A. Davis and Yifan Hu. "The University of Florida Sparse Matrix Collection." *ACM Trans. Math. Softw. 38, 1, Article 1, 25 pages,* 2011.

[23] Chris Lattner and Vikram Adve. "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation". *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization (CGO '04). IEEE Computer Society, USA, 75,* 2004.