

Paralel Programlamada Kullanılan Temel Algoritmalar

Uğur Ercan¹, Hakan Akar¹, Abdülkadir Koçer²

¹ Akdeniz Üniversitesi, Enformatik Bölüm Başkanlığı, Antalya

² Akdeniz Üniversitesi, Teknik Bilimler MYO, Antalya

ugurercan@akdeniz.edu.tr, hakanakar@akdeniz.edu.tr, akocerc@akdeniz.edu.tr

Özet: Bilgisayar donanımları, yazılımların ihtiyaçlarına cevap vermekte zorlanmaktadır. Donanım parçalarındaki hafıza ya da bit derinliği arttırılabilirken işlemci hızı neredeyse fiziksel limitlere ulaşmıştır. Donanım üreticileri fiziksel limitlere dayanan işlemci hızı yerine, bilgisayarlarda kullanılan işlemci sayısını arttırmaktadırlar. Bilgisayar yazılımlarının çok işlemcili bilgisayarlardan daha verimli yararlanabilmesi için paralel olarak programlanması gerekmektedir. Bir yazılımın nasıl paralel olarak programlanabileceği yazılımın kendisiyle doğrudan ilgilidir. Genelde her yazılım çeşitli algoritmalarla paralelleştirilebilir. Günümüzde paralel programlamada kullanılan 3 temel algoritma bulunmaktadır. Bunlar; böl ve yönet, paralel işaretçi teknikleri ve randomizasyon. Sonuç olarak her paralel programlama algoritmasının kendine özgü kullanım alanı bulunmaktadır. Hangi algoritmanın kullanılacağı ya da programın paralel programlamaya uygun olup olmadığı programın yapısıyla doğrudan ilgilidir. Çok çekirdekli işlemcilerin kişisel ve taşınabilir bilgisayarlarda dahi yaygınlaşması paralel programlamaya duyulan ihtiyacı ve bu alanda yapılan araştırmaların önemini her geçen gün arttırmaktadır.

Anahtar Sözcükler: paralel programlama, algoritma, böl ve yönet, paralel işaretçi teknikleri, randomizasyon

Basic Algorithms Used in Parallel Programming

Abstract: Computer Hardware usually cannot meet the needs of software. While memory amount or bit depth in computer hardware can be increased, processor speed has almost reached the physical limits. Hardware manufacturers increase the number of processors used in computers instead of processor speed based on physical limits. Computer software should be programmed in parallel in order to benefit from multi-processor computers more efficiently. How to develop a parallel programmed software is directly related with the software itself. Almost all software can be parallelizable with the help of several algorithms. There are three basic parallel programming algorithms which are divide and conquer, parallel pointer techniques and randomization. As a result, every parallel programming algorithm has specific use area. Presence of multi-core processors even in personal and portable computers increases the need and importance of parallel programming day by day.

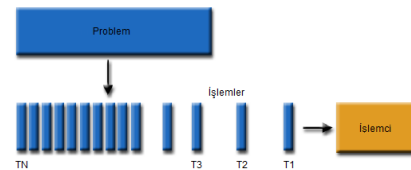
Keywords: parallel programming, algorithm, divide and conquer, parallel pointer techniques, randomization.

1. Giriş

Günümüzün hızla gelişen teknolojisi bilgisayarlar, yazılımların ihtiyaçlarına cevap vermekte zorlanmaktadır. Yazılımlar her geçen gün daha fazla hafıza, daha hızlı grafik kartı ve daha hızlı bilgisayarlar ihtiyacı duymaktadırlar. Daha fazla hafızaya ihtiyaç duymaktadırlar. Daha fazla hafızaya ihtiyaç duymaktadırlar. Daha fazla hafızaya ihtiyaç duymaktadırlar. Fakat bilgisayarların hızı fiziksel limitlere dayandığı için saat hızını arttırmak işlemcinin yanlış işlem yapmasına ya da ısınarak yanmasına sebep olmaktadır. Bu sorunu çözmek için mühendisler daha fazla işlemciyi paralel olarak aynı iş üzerinde çalışacak şekilde üretmeyi başarmışlardır. Günümüzde akıllı cep telefonlarında bile çift çekirdekli işlemciler oldukça yaygın olarak kullanılmaktadır. Bu paralel işlemci mimarisini etkin olarak kullanabilmek için yazılımların paralel olarak programlanması gerekmektedir. Peki, “paralel programlama” nedir?

Yapılacak olan işlemin tek bir bilgisayarda ve tek işlemci üzerinde çalıştırılmasına “Seri Programlama” denir. Burada aynı anda sadece tek bir işlem

yapılabilmektedir. Bir işlem bitmeden diğer bir işlem çalıştırılmaz. Seri programlamanın çalışma mantığı, Şekil-1’de gösterilmiştir[1].



Şekil 27. Seri Programlama Çalışma Mantığı

Seri programlamada her bir işlem sırayla işlendiği ve her bir işlem adımı için ayrı bir zaman ihtiyacı olduğu için, seri programlamada performans ve verimlilik gibi kısıtlamalar gözlenmektedir.

Paralel programlama ise, bir problemi çözmek için birden fazla bilgisayar kaynaklarının aynı anda kullanılmasıdır. Bir işlem çoklu işlemci kullanarak çalıştırılır ya da çoklu çekirdek teknolojisine sahip işlemci üzerinde gerçekleştirilir. Bir problem parçalara ayrılarak aynı zamanda çözülebilir. Her parça farklı

algoritma en az bilgisayar kadar çok paralellik içermelidir, aksi halde kaynaklar yetersiz kalabilir. Fakat karşıt her zaman geçerli olmaz, bazı paralel bilgisayarlarda algoritma çok sayıda paralellik içerse bile tüm algoritmaları verimli olarak yürütemez.

Bilgisayarlar paralelliğin bazı formlarını kapsadıkça, algoritma tasarımındaki vurgu seri algoritmalarından paralel algoritmalara yöneldi.

Bu alandaki gelişmeler şu şekilde sıralanabilir;

- Hesaplamanın paralel modelleri,
- Paralel algoritmik teknikler,
- Paralel karmaşıklık teorisi.

Paralel programlamada aynen seri programlamada olduğu gibi çeşitli algoritma teknikleri vardır. Bu teknikler karşılaşılan problemin durumuna göre farklılıklar gösterir. Bu algoritma tekniklerinden en çok kullanılanı ve bilineni böl ve yönet, paralel işaretçi teknikleri ve rastgeleleştirme algoritmalarıdır.

2.1. Böl ve Yönet (Divide & Conquer)

“Böl ve Yönet” terimi basitçe büyük toplulukları küçüklere ayırmak ve onları yönetmek demektir. Bu eski yöneticiler ve krallar tarafından kendi vilayetlerini yönetmek ve hatta yeni yerler fethetmek için kullanılan tekniktir. Vilayetleri din, iş, kast ya da mezhep gibi farklı adlara bölerlerdi[4].

Böl ve Yönet algoritması, orijinal problemi daha kolay çözmek için, problemi alt problemlere bölen, bölünen alt problemleri çözen, orijinal problemin çözümünü oluşturmak için, alt problemlerin çözümlerini birleştiren bir algoritmadır[5].

Böl ve yönet paradigması program modüleritesini yükseltir, genellikle basit ve verimli algoritmalara yol açar. Bu nedenle sıralı algoritma tasarımcıları için güçlü bir araç olduğu kanıtlanmıştır. Böl ve yönet paralel algoritma tasarımı daha da önemli bir rol oynamaktadır. İlk adımda oluşturduğunuz alt problemler genellikle bağımsız olduğundan, bunlar paralel olarak çözülebilir. Genellikle alt problemler özyinelemeli olarak çözümler ve böylece bir sonraki bölme adımında paralel olarak çözülecek daha çok alt problem elde edilir[5].

Ancak bilinmelidir ki, yüksek derecede bir paralel algoritma elde etmek için, böl ve yönet’in bölme ve birleştirme adımlarının paralelize (birlikte yürütülmesi) olması gerekir. Orijinal problemi mümkün olduğunca çok sayıda alt probleme bölerek, paralel olarak çözmek paralel algoritmalarla oldukça yaygındır. Paralel Böl ve Yönet örneği için, sıralı Merge Sort algoritması düşünün. Merge sort, n adet girilmiş elemanı alır ve onları sıralayarak geri verir. Bu algoritma, n elemanlı bir dizi iki parçaya bölünüp, bölünen her dizi öz yineli olarak sıralanıp ve sıralanmış yarı diziler tekrar birleştirilerek çalışır[5].

Merge Sort’un sıralı çalışma zamanını analiz etmek için, n/2 elemanlı iki sıralı dizi O(n) sürede birleştirilebileceği unutulmamalıdır.

Dolayısıyla çalışma süresi belirtilebilir,

$$T(n) = \begin{cases} T\left(\frac{n}{2}\right) + O(n), & n > 1 \\ O(1), & n = 1 \end{cases}$$

Burada çözüm

$$T(n) = O(n \log n)$$

Paralel bir algoritma olarak tasarlanmış olmasa da, merge sort onları paralel yapılmasına olanak tanıyacak iki bağımsız öz yineleme olduğundan, doğal paralelliği vardır[5].

Algoritma: mergesort(A)

1. if (|A| = 1) then return A
2. else
3. in parallel do
4. L := mergesort(A[0..|A|/2])
5. R := mergesort(A[|A|/2..|A|])
6. return merge(L,R)

$$W(n) = 2W\left(\frac{n}{2}\right) + O(n)$$

$$D(n) = \max\left(D\left(\frac{n}{2}\right), D\left(\frac{n}{2}\right)\right) + O(n)$$
$$= D\left(\frac{n}{2}\right) + O(n)$$

Beklendiği gibi, bu algoritmanın çözüm için gerekli olan zaman karmaşıklığı $W(n)=O(n \log n)$ ’dir, sıralı ardışık algoritması için de zaman aynıdır. Derinlik için çözüm $D(n)=O(n)$, fakat bu çözümünden daha küçüktür. Bir algoritmanın paralelliğini, derinliğine çalışma oranı olarak tanımlarız. Bu nedenle iyi olmasa da bu algoritmanın paralelliği $O(n \log n)$ ’dir. Burada ki sorun birleştirme adımının sıralı kalmasıdır ve performans sorunu olduğudur. Daha önce belirtildiği gibi böl ve yönet algoritmasının paralelliği, bölme ve/veya birleştirme adımlarının paralelleştirilmesi ile genellikle iyileştirilebilir. Paralel birleştirme kullanılarak sıralanmış iki dizinin, karmaşıklığı ve derinliği birleştirilebilir[5].

Bu birleştirme algoritması kullanarak, birleştirme derinliğinin yinelenmesi

$$D(n) = D\left(\frac{n}{2}\right) + O(\log \log n),$$

$$D(n) = O(\log n \log \log n) \text{ olur.}$$

Pipelined Böl ve Yönet tekniği kullanılarak, Merge sort’un derinliği $O(n \log n)$ azaltılarak $O(\log n)$ ’e indirilebilir. Buradaki esas fikir, üst öz yineleme adımlarını tamamlamadan birleştirmenin en üst seviyede başlamasıdır. Böl ve fethet, paralel problemlerin çözümü için geliştirilmiş en güçlü tekniklerden biri olduğu kanıtlanmıştır[5].

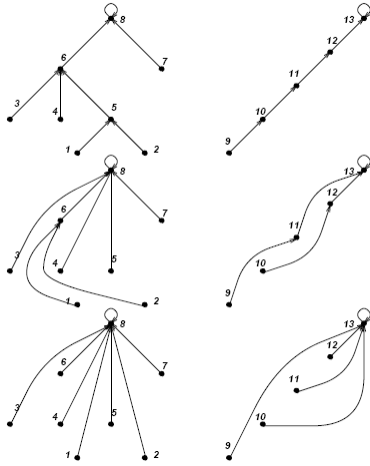
2.2. Paralel İşaretçi Teknikleri

Listeler, ağaçlar, graflar gibi geleneksel sıralı tekniklerin bir çoğunu paralel tekniklere çevirmek kolay değildir. Örneğin bağlı liste elementlerin dolaşmak, ağaç düğümlerini sıralı gezmek, graflarda derinlik öncelikli dolaşmak sıralı tekniklerdir. Şans eseri bu teknikler, neredeyse aynı kuvvetteki paralel tekniklerle genellikle yer değiştirilebilir. [5]

2.2.1. İşaretçi Atlama

Paralel algoritmaların tasarımı, iş karmaşıklığının en aza indirecek düzeyde olması önemlidir. İşaretçi atlama tekniği bağlı liste ve ağaç yapıları için kullanılan yaygın bir teknik olup paralel programlama tekniklerinde sık kullanılan bir yöntemdir [6]. Basit olarak ifade etmek gerekirse işlem dizisinde bir elemanın adresini diğerinin değerine atama işlemidir. Örnek bir algoritma yapısı aşağıda ve ilgili iterasyon dağılımı şekil 3 'de verilmiştir [7].

1. Forall $i \in 1 : n$ do
2. $S[i] \leftarrow P[i]$
3. While $S[i] \neq S[S[i]]$ do
4. $S[i] \leftarrow S[S[i]]$
5. Endwhile
6. Enddo



Şekil 3. İki Ağaç (Tree) Ve 13 Başucu Noktasının Üç İterasyon Dağılımı

2.2.2. Euler Tur

Yönlendirilmiş graflarda Euler turu her bir düğümün bir kez ziyaret edildiği graf dolaşma turudur. Yönsüz grafta zıt yöndeki iki tane düğüm ile yer değiştirir. Yönsüz ağaçta Euler turu ağacın çevresini dolaşırken her bir düğümü bir aşağı inerken, bir yukarı çıkarken olmak üzere iki kez ziyaret eder. Ağaçtaki Euler turu, temsil eden bağlı yapıyı koruyarak her bir alt ağacın miktarı gibi ağaçtaki bazı özellikleri hesaplamak mümkündür. Bu teknik ağacın derinliğinden bağımsız olarak paralel derinliği ve sıralı işi kullanır. Euler turu, derinlik öncelikli dolaşma gibi, standart ağaç dolaşma teknikleri yerine kullanılabilir[5].

2.2.3. Graf Küçültmesi

Graf küçültmesi, grafin orijinal yapısını koruyarak daha küçük boyutlara indirmeye yarar. Genellikle graf küçültme operasyonu gerçekleştirdikten sonra problem küçültülen grafta özyinelemeli olarak çözülür.

Küçültülen graftaki problemin çözümü, sonuç çözümünü oluşturmakta kullanılır. Örneğin, bir grafi onu oluşturan parçalara ayırmanın bir yolu düğümleri komşu düğümlerle birleştirerek küçültülmüş graftaki bileşenlerin bağlantılarını bularak ve küçültme işlemini geri alarak yapılır. Bir çok problem ağaçları küçültülerek çözülebilir. Böyle bir durumda da bu tekniğe ağaç küçültme tekniği denir. [5]

2.2.4. Kulak Ayırıştırma

Bir grafta kulak ayırıştırma, graftaki köşelerin parçalanarak sıralı yol koleksiyonu haline getirilmesidir. İlk yol çember, diğerleri kulak olarak isimlendirilir. Her bir kulağın bitiş noktası önceki yollara bağlanır. Bir graftaki kulak ayırıştırma bir kez bulunduğu, iki noktanın aynı çember üzerinde yer aldığı belirlemek zor değildir. Bu bilgi ikili, üçlü ve çoklu bağlantılılığı belirlemek için algoritmalarda kullanılabilir. Kulak ayırıştırma, grafin yapısından bağımsız olarak logaritmik derinlik ve sıralı iş kullanılarak paralel olarak bulunabilir. Böylece bu teknik derinlik öncelikli arama gibi problemlerin çözümünde standart sıralı tekniklerin yerine kullanılabilir. [5]

2.3. Rastgeleleştirme (Randomization)

Paralel algoritmalarda rastgele sayılar kullanılır. İşlemciler rastgele sayıları kullanarak genelde iyi sonuçlara ulaşmak amacıyla komutları yerel olarak işleyebilirler. Rastgele sayıların paralel programlamada en yaygın kullanılan 3 örneği; örnekleme, simetri kırılması ve yük dengelemedir.

2.3.1.Örnekleme

Rastgeleliliğin paralel programlama kullanımının bir çeşidi de bir element setinden temsilci örneğin seçilmesidir. Genellikle problem seçilen bu örnekle çözülür. Bu örnekte kullanılan çözüm yöntemi, orijinal veri setinin çözümü için bir rehber oluşturmuş olur. Örneğin, elimizde bir tamsayı kümesi var ve biz bunu sıralamak istiyoruz. Bunu tamsayı kümesini alt kümelere bölerek ve her bir kümeyi kendi içinde sıralayarak bu işi yapabiliriz. Bu algoritmanın iyi çalışması için, kümelerin birbiri ile örtüşmeyen sayı değerleri ile temsil edilmesi ve her bir kümenin yaklaşık olarak eşit sayıda anahtar içermesi gerekmektedir. Rastgele örnekleme, aralık sınırlarının belirlenmesinde kullanılır. Öncelikle her bir işlemci kendi rastgele örnekleme anahtarlarını seçer. Sonra seçilen bütün anahtarlar birlikte sıralanır. Son olarak bu anahtarlar sınır değerler olarak kullanılır. Bu tarz rastgele örnekleme ayrıca birçok paralel hesaplama geometrisinde, graflarda ve kelime eşleştirme algoritmalarında da kullanılmaktadır.

2.3.2. Simetri Kırılması

Rastgeleliliğin bir başka kullanımı da simetri kırılmasıdır. Örneğin, bir çizgede bağımsız düğümler setinin seçilmesi problemini düşünelim. (birbiri ile komşu olamayan düğümler seti bağımsız düğümlerdir.) diğer bütün düğümlerle paralel olarak her bir düğümün bağımsız düğümler setine katılıp katılmaması gerektiğine karar verilmesi gerektiğini düşünün. Eğer düğümlerden birisinin bağımsız düğümler setine katılmasına karar verilirse, ona komşu olan diğer bütün düğümlerin sete katılmaması gerekmektedir. Eğer her bir düğümün yerel yapısı aynıysa, her bir düğüm eşit sayıda komşuya sahipse, hangi düğümlerin sete katılacağına anlık olarak karar vermek oldukça zordur. Böyle zor bir çıkmaz durum, düğümler arasındaki simetriyi kırmak için rastgeleliliğin kullanılmasıyla çözülebilir[8].

2.3.3. Yük dengeleme

Rastgeleliliğin üçüncü kullanım şeklide yük dengelemedir. Çok büyük miktardaki veri kümesini yaklaşık olarak eşit büyüklükte alt kümelere hızlıca ayırmanın bir yolu da her bir maddeyi rastgele olarak alt kümelere atamaktır. Bu teknik en iyi, alt kümelerin ortalama boyutu en azından logaritmik olarak ilk veri kümesinin boyutlarında olduğu zamanlarda çalışır.

3. Sonuç

Günümüzde kullanılan bilgisayar donanımları çok çekirdekli olmasına rağmen, kullandığımız yazılımların çok az bir kısmı paralel olarak programlanmıştır. Bu sebeple seri programlanan bu yazılımlar donanımı etkili olarak kullanamamakta ve kullanıcının zaman kaybetmesine sebep olmaktadır. Paralel programlamada 3 temel algoritma kullanılmaktadır. Paralel programlama algoritmalarının kendine özgü kullanım alanı bulunmaktadır. Hangi algoritmanın kullanılacağı ya da programın paralel programlamaya uygun olup olmadığı programın yapısıyla doğrudan ilgilidir. Bu sebeple herhangi bir paralel programlama algoritmasının diğerinden daha iyi ya da kötü olduğunu söylemek mümkün değildir. Fakat paralel programlamanın temel mantığında bir problemi küçük parçalara ayırmak olduğu için en sık kullanılan algoritmanın “böl ve yönet” algoritması olduğu söylenebilir. Çok çekirdekli işlemcilerin kişisel ve taşınabilir bilgisayarlarda dahi yaygınlaşması paralel programlamaya duyulan ihtiyacı ve bu alanda yapılan araştırmaların önemini her geçen gün arttırmaktadır.

4. Kaynaklar

[1] MPI Programlamaya Giriş ve Motivasyon. http://www.uybhm.itu.edu.tr/documents/basarim09sunum/01_Giris_ve_Motivasyon_akinci_v3.pdf , Erişim Tarihi 26.11.2012

[2] Akçay, M., & Erdem, H. A. (2010). Paralel Hesaplama ve Matlab Uygulamaları. Akademik Bilişim 2010. Muğla: Muğla Üniversitesi.

[3] Altıntaş, V., & Yeğenoğlu, E. D. (2011). Görüntü işlemede seri ve paralel programlamanın performansı. 6. International Advanced Technologies Symposium.

[4] Divide And Conquer Method. [www.curriki.org: http://www.curriki.org/xwiki/bin/view/Coll_nishantgupta/Lesson2DivideandConquerMethod?viewer=print](http://www.curriki.org:www.curriki.org/xwiki/bin/view/Coll_nishantgupta/Lesson2DivideandConquerMethod?viewer=print) , Erişim Tarihi 26.11.2012

[5] Guy, E. B., & Bruce, M. M. Parallel Algorithms. www.cmu.edu: http://www.cs.cmu.edu/~guyb/papers/BM04.pdf , Erişim Tarihi 26.11.2012

[6] Wyllie, J. C. (1979). The Complexity of parallel computations. Technical Report TR-79-387. Ithaca, NY: Department of Computer Science, Cornell University.

[7] Chatterjee, S., & Jan Prins, J. Parallel Computing PRAM Algorithms. [www.unc.edu: http://www.cs.unc.edu/~prins/Classes/633/Handouts/pram.pdf](http://www.cs.unc.edu/~prins/Classes/633/Handouts/pram.pdf) , Erişim Tarihi 26.11.2012

[8] Luby, M. (1986). A simple parallel algorithm for the maximal independent set problem. *SIAM Journal of Computing* , 1036-1054.