PAMUKKALE ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ
PAMUKKALE UNIVERSITY ENGINEERING COLLEGE
MÜHENDİSLİK BİLİMLERİ DERGİSİ
JOURNAL OF ENGINEERING SCIENCES
YIL : 1998
CİLT : 4
SAYI : 3
SAYFA : 837-841

KISMİ DİFERANSİYEL DENKLEMLERİN PARALEL ÇÖZÜM YÖNTEMLERİ

Korhan KARABULUT, Mustafa İNCEOĞLU, Levent TOKER
Ege Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Bornova/İzmir

ÖZET

Kısmi diferansiyel denklemler bilimin ve mühendisliğin hemen her alanında ortaya çıkar. Bu denklemlerin çözümü için harcanan bilgisayar zamanı, herhangi başka bir problem sınıfını çözmek için harcanan zamandan daha fazladır. Bu nedenle, kısmi diferansiyel denklemler, çok büyük bir işlemci gücü sağlayan paralel bilgisayarlar üzerinde çözümlenmeye çok uygundur. Bu çalışmada kısmi diferansiyel denklemlerin Jacobi, Gauss-Siedel, SOR (Ardışık aşırı yumuşatma) ve SSOR (Simetrik SOR) algoritmalarıyla paralel olarak çözülmesi incelenmiştir.

Anahtar Kelimeler : Kısmı diferansiyel denklemler, Paralel işleme, Hiperküp

PARALLEL SOLUTION METHODS OF PARTIAL DIFFERENTIAL EQUATIONS

SUMMARY

Partial differential equations arise in almost all fields of science and engineering. Computer time spent in solving partial differential equations is much more than that of in any other problem class. For this reason, partial differential equations are suitable to be solved on parallel computers that offer great computation power. In this study, parallel solution to partial differential equations with Jacobi, Gauss-Siedel, SOR (Succesive Over-Relaxation) and SSOR (Symmetric SOR) algorithms is studied.

Key Words : Partial differantial equations, Parallel processing, Hypercube

1. GİRİŞ

Birçok fiziksel olayda ve mühendislik alanında ortaya çıkan problemlerin modellenmesinde kısmi diferansiyel denklemler kullanılır. Bu alanlara örnek olarak hava tahmini, supersonik akışın modellenmesi, elastikiyet çalışmaları, ısı transferi verilebilir (Gerol ve Wheatley, 1984). Kısmi diferansiyel denklemler, birçok pratik problem için analitik olarak çözülemez. Bu nedenle, sayısal yöntemler yaklaşık sonuçların hesaplanmasında çok önemlidir. Bu sayısal yöntemlerde genelde Ω problem alanının dikdörtgensel olduğu varsayılır. Daha karmaşık alanlar için, ilgili kısmi diferansiyel denklemleri ayrıklaştırmak için özel yöntemlere gereksinim duyulur. Örneğin, hesapsal akışkanlar dinamiğinde (Computational Fluid Dynamics - CFD) fiziksel alanlarda ortaya çıkan karmaşık geometriler dikdörtgensel veya kübik alanlara sayısal ızgara yaratımı yöntemleri ile çevrilebilir (Wylie ve Benett, 1995).

Kısmi diferansiyel denklemlerin çözümünde kullanılan sonlu fark yöntemleri (finite difference methods) bu yüzyılın başından beri biliniyordu. Ama sayısal bilgisayarların kullanılmaya başlanması ile sayısal yöntemler kısmi diferansiyel denklemlerin çözümünde önemli bir araç haline geldi. Bu adımdan sonra sonlu eleman (finite element), sonlu hacim (finite volume), çoklu ızgara (multigrid) yöntemleri gibi yeni sayısal algoritmalar ortaya çıktı. Tüm bu

algoritmalar sayısal bilgisayarlar üzerinde çözülmeye uygundur. Ancak bu çözümler için harcanan bilgisayar zamanı herhangi başka bir sınıf problemi çözmek için harcanan bilgisayar zamanından çok daha fazladır. Bu nedenle kısmi diferansiyel denklemlerin çözümü için çok büyük işlemci gücü sağlayan paralel bilgisayarlar kullanılabilir.

2. MATERYAL VE YÖNTEM

Çalışmada herbir paralel algoritma için zamanlama alınırken hata oranı 10^{-5}'den başlanarak, 10^{-9}'a kadar düşürülmüş ve her bir adım sırasıyla 1, 2, 4, 8, 16, 32 ve 64 işlemci için beşer kere tekrarlanmıştır. Oluşan her beş sonuçtan en büyük ve en küçük olanlar atılmış, kalan üç değerin ortalaması alınmıştır.

Çalışmada, UNIX işletim sisteminde çalışan, Intel tarafından yazılmış olan Intel iPSC/2 hiperküp benzetleyicisi kullanılmıştır. Benzetleyici, kişisel bilgisayarlar için yazılmış bir UNIX işletim sistemi olan, LINUX işletim sistemi çalıştıran 16 Mbyte bellekli, 150 Mhz hıza sahip Pentium türdeki bir bilgisayarda kullanılmıştır. Programların yazımında standart C derleyicisinin yanında hiperküp benzetleyicisinin programlama kütüphaneleri kullanılmıştır.

Jacobi, Gauss-Siedel, SOR ve SSOR yöntemlerinin hepsi yumuşatma (relaxation) tipi algoritmalardır. Bu algoritmaların ortak özellikleri (1)'deki gibi belirtilebilmeleridir (Zhu, 1994).

$$u_k = Tu_{k-1} + c,\ k = 1, 2, \cdots \qquad (1)$$

Burada u_0 hesaplamalara başlamak için kullanılan ilk tahmin, c ise bir sabittir. Tekrarlı T matrisinin, A katsayı matrisinden oluşturulmasında kullanılan yönteme göre Jacobi, Gauss-Siedel veya SOR yumuşatma algoritmaları oluşur. Bu işlem, yeni u_k değerlerinin u_{k-1} değerleri cinsinden belirtilmesinden oluşmaktadır. Yeni değerlerin hesaplanması önceden belirtilen bir hata değerine ulaşılana kadar sürer.

Jacobi algoritması yeni değerlerin tümünün eski değerler üzerinden hesaplanmasına dayanır. Merkezi fark yöntemi kullanılarak Jacobi algoritmasında bir noktanın değerinin nasıl güncellendiği (2)'de belirtilmiştir.

$$u_{i,j}^{(k)} = \frac{u_{i,j-1}^{(k-1)} + u_{i-1,j}^{(k-1)} + u_{i+1,j}^{(k-1)} + u_{i,j+1}^{(k-1)} - h^2 f(x_i, y_i)}{4} \quad i = 1, 2, \cdots NI, \qquad j = 1, 2, \cdots .NJ,\ k = 1, 2, \cdots . \qquad (2)$$

Jacobi algoritmasında $u_j^{(k)},\ j = 1, 2, \cdots, i-1$ hesaplanırken $u_i^{(k)}$ hesaplanmış olur. Gauss-Siedel algoritması bu yeni değerlerin kullanılmasına dayanmaktadır. Bu algoritmada her bir noktanın güncelleme (3)'de verilmiştir.

$$u_{i,j}^{(k)} = \frac{u_{i,j-1}^{(k)} + u_{i-1,j}^{(k)} + u_{i+1,j}^{(k-1)} + u_{i,j+1}^{(k-1)} - h^2 f(x_i, y_i)}{4},\ i = 1, 2, \cdots NI,\ j = 1, 2, \cdots NJ,\ k = 1, 2, \cdots . \qquad (3)$$

Gauss-Siedel algoritmasında elde olan en güncel değerler kullanıldığından Jacobi algoritmasından daha iyi yakınsadığı düşünülür. Gerçekten de Gauss-Siedel algoritması Poisson denkleminde ve ısı denkleminde Jacobi algoritmasından iki kat daha hızlı yakınsar (Zhu, 1994).

Hem Jacobi hem de Gauss-Siedel algoritmasında $u_i^{(k)}$ değeri hesaplanırken $u_i^{(k)}$ noktasının komşularından yararlanılır. Hesaplamada $u_i^{(k-1)}$ değerinden yararlanılmaz. SOR algoritmasının temel düşüncesi SOR algoritmasının temel fikri eldeki güncellenmiş $u_i^{(k)}$ ve daha önceki adımda hesaplanmış $u_i^{(k-1)}$ değerlerinden yararlanmaktır (4).

$$u_{i,j}^{(k)} = (1-\omega)\ u_{i,j}^{(k)} + \omega\ \frac{u_{i,j-1}^{(k)} + u_{i-1,j}^{(k)} + u_{i+1,j}^{(k-1)} + u_{i,j+1}^{(k-1)} - h^2 f(x_i, y_i)}{4} \quad i = 1, 2, \cdots NI,\ j = 1, 2, \cdots NJ,\ k = 1, 2, \cdots . \qquad (4)$$

$\omega = 1$ durumunda, SOR algoritması Gauss-Seidel algoritmasına dönüşür. SOR algoritması pozitif tanımlı simetrik matrisler için $0 < \omega < 2$ arasında yakınsaktır. ω parametresinin seçimi zordur ve

yakınsama oranı için önemlidir. Sadece çok özel durumlarda ω değerinin analitik gösterimi elde edilebilir (Mathews, 1992).

SSOR algoritması da SOR algoritması ile yakından ilgilidir. Tam bir SSOR adımı iki SOR adımından oluşur. İlk adım bilinen SOR adımıdır. İkinci adım birinci adımdan elde edilen değerlerin tersten uygulandığı bir SOR adımıdır. Bazı özel problemler için SSOR metodu, SOR metodundan iki kat veya daha hızlı yakınsar.

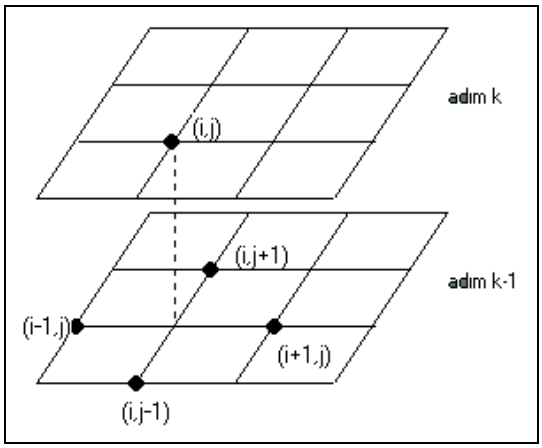
Bu algoritmaların paralelleştirilmesinde ilk olarak haberleşme masrafını azaltmak için her bir düğüm, düğüm numaraları arasında bir bitlik bir fark olacak biçimde (bu yerleşim Graycode olarak anılır) yerleştirilmiştir.

Tüm paralel algoritmalar paralelleştirilirken, çözüm bulunacak tüm ızgaranın işlemcilere dağıtılma mantığı da aynıdır: ızgaranın satırları işlemci sayısına bölünür. Herbir işlemciye

$$\frac{\text{Toplam satır sayısı} \ * \ \text{işlemci sayısı}}{\text{toplam sütun sayısı}}$$

kadar bir ızgara parçası düşer.

Jacobi algoritmasında $u_{i,j}^{(k)}$ değerlerinin hesaplanması için sadece bir önceki adımdaki değerler kullanılır. Şekil 1, k ve k-1 adımlarındaki değerler arasındaki ilişkiyi göstermektedir.



Şekil 1. Jacobi algoritmasında k ve k-1 adımları arasındaki ilişki

k numaralı adım başlamadan önce k-1 numaralı adımdaki tüm değerler hesaplanmış olacağından, k numaralı adımdaki tüm değerler birbirinden bağımsız olarak ve aynı anda hesaplanabilir. Yani Jacobi algoritması doğal olarak paraleldir.

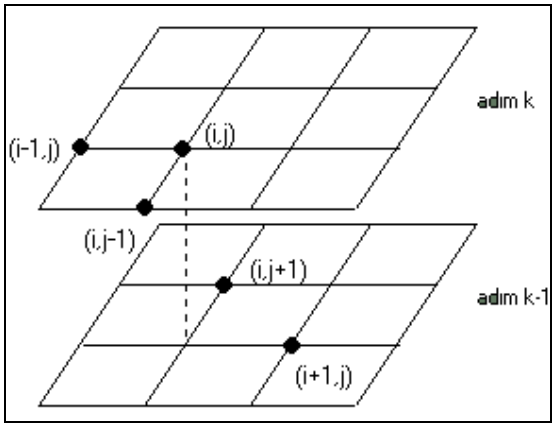
Herbir işlemcide, sınırdaki noktaların değerlerinin hesaplanması için diğer işlemcilere atanmış olan noktaların değerleri gereklidir. Bu noktalara "hayalet noktalar" (ghost points) da denir (Zhu, 1994). Benzer olarak bir işlemcinin elindeki değerlerin de diğer işlemciler tarafından erişilebilir olması gereklidir.

Bu algoritmada, herbir işlemci, elindeki sınır noktaların değerleri dışında, kendi alt kümesinde oluşan en büyük hata oranını da diğer işlemcilere belirtmek zorundadır. İşlem, hata oranı tüm işlemciler üzerinde önceden belirlenen bir oranın altına düşene kadar sürer.

```
Değerleri ilkle
   Hata oranı > ε iken yinele
   Tüm noktaların değerlerini (2)'yi kullanarak güncelle
   Sınırlardaki güncellenmiş değerleri komşu işlemcilere gönder
   İlgili komşudan sınırlardaki güncelleşmiş değerleri al
   Hata oranını hesapla
Son
```

Algoritma 1. Paralel jacobi algoritması

Gauss-Siedel algoritmasının paralelleştirilmesi Jacobi algoritması kadar açık değildir. İlgili denklemden görülebileceği gibi, $u_{i,j}^{(k)}$ değerinin hesaplanması için bir önceki adımdan $u_{i+1,j}^{(k-1)}$ ve $u_{i,j+1}^{(k-1)}$ değerlerinin yanı sıra, şimdiki adımdan $u_{i-1,j}^{(k)}$ ve $u_{i,j-1}^{(k)}$ değerlerine gereksinim duyulur (Şekil 2). Bu bağımlılıktan dolayı $u_{i,j}^{(k)}$ değerinin hesaplanmasına $u_{i-1,j}^{(k)}$ ve $u_{i,j-1}^{(k)}$ hesaplanmadan başlanamaz.



Şekil 2. Gauss-Siedel algoritmasında adımlar arasındaki ilişki

Bu algoritmanın paralelleştirilmesinde kullanılan yöntem, kırmızı-siyah (red-black ordering) veya tek-çift (odd-even ordering) sıralamasıdır. Sıralama, bir noktanın aynı renkteki diğer bir noktaya doğrudan bağlantısı olmayacak biçimde yapılır (Ortega ve Voigt, 1985). Bu sıralama sonucunda, ilk adımda tüm siyah noktalar, daha sonra kırmızı noktalar paralel olarak hesaplanır. Eğer bir kırmızı veya siyah nokta işlemcinin elindeki alt kümenin sınırlarında yer alıyorsa, bu noktanın güncellenmiş değeri diğer noktalara her bir adımdan sonra gönderilmelidir.

```
Değerleri ilkle
   Hata oranı > ε  iken yinele
   Tüm siyah noktaların değerlerini güncelle
   Sınırlardaki güncellenmiş değerleri komşu
   işlemcilere gönder
   İlgili komşudan sınırlardaki güncelleşmiş değerleri al
   Tüm kırmızı noktaların değerlerini güncelle
   Sınırlardaki güncellenmiş değerleri komşu
   işlemcilere gönder
   İlgili komşudan sınırlardaki güncelleşmiş değerleri al
   Hata oranını hesapla
Son
```

Algoritma 2. Paralel Gauss-Siedel algoritması

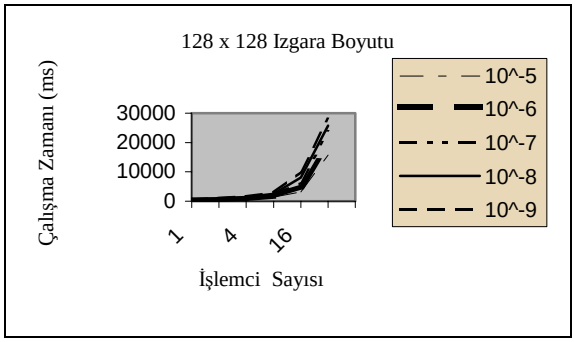
SOR algoritması da aynen Gauss-Siedel algoritması gibi kırmızı-siyah sıralaması ile paralelleştirilebilir. Burada, güncel $u_{i,j}^{(k)}$ değerini hesaplamak için kullanılacak denklemler değiştirilir.

SSOR algoritması da aynen SOR algoritması gibi paralelleştirilebilir. Zaten SSOR algoritması, hatırlanacağı gibi, SOR algoritmasının bir adımda iki kere (birisi normal olarak, birisi de tersten) uygulanmasıdır. Bu nedenle burada haberleşme sayısı, SOR algoritmasının iki katıdır.

SOR ve SOR algoritmalarının paralelleştirilmesinden elde edilen algoritmalar Algoritma 2 benzeridirler. Bu nedenle, adı geçen algoritmalar burada belirtilmemiştir.

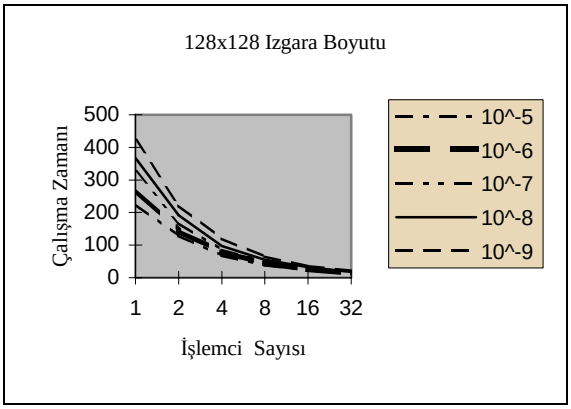
4. BULGULAR VE TARTIŞMA

Jacobi algoritmasında alınan sonuçlar değerlendirildiğinde, programın çalışma zamanının işlemci sayısı ile birlikte arttığı görülmektedir. İşlemci sayısı arttıkça çalışma zamanın artması, Jacobi algoritmasının oldukça yavaş yakınsamasından kaynaklanmaktadır. Bu nedenle programın işleyiş zamanı artmakta ve işlemciler arasındaki haberleşme maliyeti oldukça yükselmektedir (Şekil 3).



Şekil 3. 128 x 128 ızgara boyutunda Jacobi algoritması için çalışma zamanı / işlemci sayısı grafiği

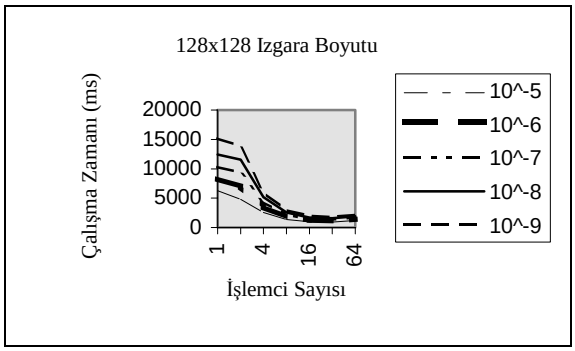
Gauss - Siedel algoritmasında işlemci sayısı arttıkça çalışma zamanı azalmaktadır. Çalışma zamanı beklenen biçimde bir eğri çizmektedir. Gauss - Siedel algoritması ile en iyi sonuçlar elde edilmiştir (Şekil 4).



Şekil 4. 128x128 ızgara boyutunda Gauss-Siedel algoritması için çalışma zamanı / işlemci sayısı grafiği

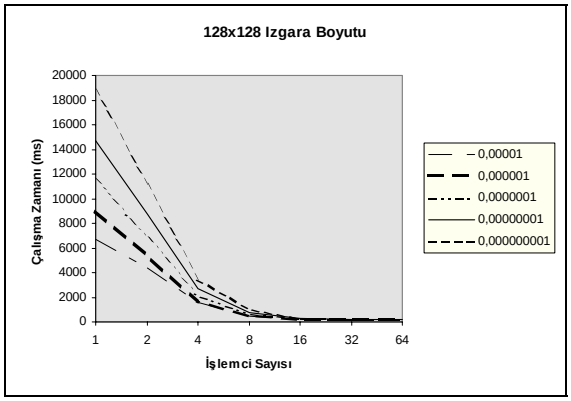
SOR algoritması teorik olarak Gauss-Siedel algoritmasından daha iyi yakınsadığı halde iki adımdan oluştuğundan her bir adımda iki kat haberleşmeye gereksinim duymaktadır. Bu da, programın çalışma zamanını arttırmaktadır. Ayrıca bu algoritmada daha önce de belirtildiği gibi ω parametresinin seçimi oldukça önemlidir. Bu programda kullanılan ω değeri, deneysel olarak elde edilen en iyi sonuçtur. Bu değer, çözülen problemin cinsine göre değişmektedir. Bu algoritmada 2 işlemci için çalışma zamanı artmakta, daha sonra 4,

8, 16, 32 ve 64 işlemci için çalışma zamanı işlemci sayısıyla bağlantılı olarak azalmaktadır (Şekil 5).



Şekil 5. 128 x 128 ızgara boyutunda SOR algoritması için çalışma zamanı / işlemci sayısı grafiği

SSOR algoritması teorik olarak en hızlı yakınsayan algoritmadır. Ama algoritmanın bir adımı iki tane SOR adımından oluştuğundan, algoritma daha iyi yakınsamasına rağmen, her bir adımda dört tane haberleşme yapıldığı için, programın çalışma zamanı artmaktadır. Yine de 32 ve 64 işlemciler için elde edilen sonuçlar oldukça etkindir. Bu algoritmada da, SOR algoritmasında olduğu gibi ω parametresinin değerinin etkin olarak elde edilmesi problemi vardır. Bu parametrenin probleme uygun olmayan değerde kullanılması, algoritmanın yakınsama oranını azaltmaktadır (Şekil 6).



Şekil 6. 128 x 128 ızgara boyutunda SSOR algoritması için çalışma zamanı / işlemci sayısı grafiği

5. SONUÇ

Bu çalışmada incelenen ve paralelleştirilen algoritmalar arasında en iyi başarımı ve etkinliği Gauss-Siedel algoritması elde etmiştir. Jacobi algoritması doğal olarak paralel olmasına rağmen düşük yakınsama oranı nedeniyle çok fazla işlemciler arası haberleşmeye gerek duymakta ve bu nedenle, işlemci sayısı arttıkça çalışma zamanı da artmaktadır.

SOR ve SSOR algoritmaları ise yüksek yakınsama oranlarına sahip olmalarına rağmen, her bir adımdaki haberleşme sayısının fazla olması nedeniyle çalışma zamanları yüksek çıkmıştır. Yine de bu algoritmalar 32 ve 64 işlemci için Gauss-Siedel algoritmasına yakın zamanlamalar elde etmişlerdir. Burada değinilmesi gereken ise ω parametresinin değeridir. Bu değer doğrudan olarak elde edilemediğinden deneme yoluyla elde edilmiştir. Bu nedenle, bu değer hataya açıktır.

Gauss-Siedel algoritmasından hareketle yazılan paralel program en iyi çalışma zamanlarını elde etmiştir. Bu program, işlemci sayısı arttıkça daha iyi çalışmıştır. Bunun nedeni ızgaranın işlemciler arasında bölünmesidir. Burada özellikle sınıra yakın olan ızgara parçalarının, daha çabuk yakınsadığı görülmüştür.

6. KAYNAKLAR

Gerol, C. F. and Wheatley, P. O. 1984. Applied Numerical Analysis, Third Edition.

Mathews, J. H. 1992. Numerical Methods for Mathematics, Science and Engineering, Second Edition.

Ortega J. M., Voigt, R. G. 1985. Solution of Partial Differential Equations on Vector and Parallel Computers, Philedelphia.

Wylie, C. R. and Bennett, L. C. 1995. Advanced Engineering Mathematics, Sixth Edition.

Zhu, J. 1994. Solving Partial Differential Equations on Parallel Computers, Singapore, Rive.