

YILDIZ TEKNİK ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

Paralel Algı Tasarımı'nın Temelleri

YÜKSEK LİSANS TEZİ

Füsun Yaman

1990

YILDIZ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

bilgi
50000 T

PARALEL ALGORİTMA TASARIMI'NIN
TEMELLERİ

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜH. FÜSUN YAMAN

TEZ TÖNETİCİSİ
PROF.DR. M.YAHYA KARSLIGİL

İSTANBUL , 1990

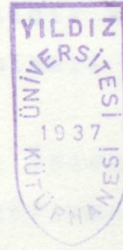
YILDIZ TEKNİK ÜNİVERSİTESİ
KÜTÜPHANE DOKÜMANTASYON
DAİRE BAŞKANLIĞI

Kot	: R 368
Alındığı Yer	: 59
	: Fen Bil. Ens.
Tarih	: 13.4.95
Fatura	:
Fiyatı	: 60.000.-
Ayniyat No	: 1-4
Kayıt No	: 51038
UDC	:
Ek	:

YILDIZ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

0107

PARALEL ALGORİTMA TASARIMI'NIN
TEMELLERİ



YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜH. FÜSUN YAMAN

TEZ TÖNETİCİSİ

PROF.DR. M.YAHYA KARSLIĞİL

İSTANBUL , 1990

ÖZET

80'li yılların başlarında paralel bilgisayarlar sadece araştırma laboratuvarlarında görülmekte idiler. Bugün ise ticari olarak kullanılmaktadırlar. Yüksek performanslı bilgisayarlara ihtiyaç pekçok alanda artmakta bundan dolayı da gerek bilimsel çalışmalarda gerekse diğer alanlarda kullanılan standart ardaşıl algoritmaların paralellik kavramı göz önünde tutularak tekrardan değerlendirilmesi, tamamen farklı bir yaklaşım, gerekmektedir. Paralel bilgisayar mimarilerinin ortaya çıkması, bunlarda çalışacak efektif nümerik algoritmaların tasarımı, ve algoritmaların ifade edileceği programlama dillerinin sunulmasına neden olmuştur. Şu unutulmamalıdır ki paralellikte algoritmalar ve programlama dili en az mimari kadar önemlidir.

Bu proje 'Paralel Veri İşleme' konusunda geniş bir özetle birlikte, paralel algoritmaların tasarımı için mümkün yaklaşımları vermekte ve bunların gerçekleştirilmesi için kullanılacak olan , tamamen farklı bir programlama metodunu - Concurrent Programlamayı- anlatmaktadır. Projenin amacı paralel algoritma tasarımının prensiplerini teorik olarak anlatmak, ardaşıl ve paralel algoritmaların karşılaştırmasını yaparak, her iki türün de avantaj ve dezavantajlarını sunmaktır.

ABSTRACT

A few years ago parallel computers could be found only in research laboratories. Now they are available commercially. High-performance computers are increasingly in demand in many areas. The standard sequential algorithms for scientific computing and information manipulation therefore have to be reconsidered in a parallel context. For the science of algorithms it means that a new, conceptually different, approach is needed in design of algorithms. The advent of highly parallel architectures also introduces the problem of designing numerical algorithms that execute efficiently on them, and computer languages in which these algorithms can be expressed. It has to be regarded that algorithmic and language aspects of parallelism as being equally important as the architectural.

This project gives a general survey of parallel processing as well as presenting possible approaches in implementing parallel algorithms and explaining a completely different strategy -Concurrent Programming-. The aim of preparing this project is to tell the principles of designing parallel algorithms teoritically and comparing parallel algorithms with the sequential ones to find out the advantageous and disadvantageous both of them.

III. CONCURRENT PROGRAMLAMA

3.1 Genel Tanım	3.1
3.2 Concurrent Olarak Çalıştırma	3.2
Çoklu İşlemci İçin Kullandığımız	
3.3 Concurrent Programlama Notasyonu	3.3

	3.4 Kritik Bölgeler	3.7
	3.5 Karşılıklı Yasağına (Mutual Exclusion)	3.10
	3.5.1 Problemin Tanımlanması	3.12
	3.5.2 İlk Adım	3.12
	3.5.3 İKİNCİ ADIM	3.16
	3.5.4 Üçüncü Adım	3.17
	3.5.5 Dördüncü Adım	3.19
Özet	3.5.6 Dekker Algoritması	3.20
Abstract	3.5.7 Sonuç	3.23
	3.6 Senkronizasyon Metodları	
	3.6.1 Paylaşılan Değişkenlerle	3.26
I.	PARALEL VERİ İŞLEME	
	1.1 Giriş	1.17
	1.2 Paralel Veri İşleme	1.28
	1.3 Paralel Yaklaşım	1.59
	Increment And Decrement	3.30
	Fetch And Add	3.30
II.	PARALEL BİLGİSAYARLARIN TANIMI VE SINIFLANDIRILMASI	
	2.1 Tanımlar	2.2
	2.2 Flynn'in Sınıflandırması	2.3
IV.	2.3 Feng'in Sınıflandırması	2.10
	2.4 Hengler'in Sınıflandırması	2.13
III.	CONCURRENT PROGRAMLAMA	
	3.1 Genel Tanımlar	3.1
	3.2 Concurrent Olma Özelliğinin Çoklu İşlemciler İçin Kullanımı	3.2
	3.3 Concurrent Programlamada Notasyonlar	3.3

V.	3.4	Kritik Bölgeler	3.7
	3.5	Karşılıklı Yasaklama (Mutual Exclusion)	3.10
	3.5.1	Problemin Tanımlanması	3.12
	3.5.2	İlk Adım	3.12
	3.5.3	İkinci Adım	3.16
	3.5.4	Üçüncü Adım	3.17
	3.5.5	Dördüncü Adım	3.19
VI.	3.5.6	Dekker Algoritması	3.20
	3.5.7	Sonuç	3.23
	3.6	Senkronizasyon Metodları	3.24
	3.6.1	Paylaşılan Değişkenlerle	3.26
	3.6.2	Senkronizasyon ve	3.27
	3.6.3	Primitive'ler	3.27
	3.6.4.1	Lock—Unlock	3.27
	3.6.4.2	Test And Set	3.28
	3.6.4.3	Compare And Swap	3.29
	3.6.4.4	Increment And Decrement	3.30
	3.6.4.5	Fetch And Add	3.30
	3.6.2	Semaphore'lar ile Senkronizasyon	3.31
	3.6.3	Monitorlar ile Senkronizasyon	3.31
VII.	3.6.4	Mesaj Yollama ile Senkronizasyon	3.32
	3.7	Senkronizasyon ve	3.33
	3.8	Senkronizasyon ve	3.34
	3.9	Senkronizasyon ve	3.35
IV.		PARALEL VERİ İŞLEMEDE KULLANILAN TEMEL	4.0
		BİLGİSAYAR MODELLERİ	4.0
	4.1	Seri Bilgisayar Modeli	4.1
VIII.	4.2	Paylaşılan Bellekli SIMD Bilgisayarlar	4.2
	4.3	İçbağlantı Ağılı SIMD Bilgisayarlar	4.5
	4.4	Senkronizasyon ve	4.6
	4.5	Senkronizasyon ve	4.7
	4.6	Senkronizasyon ve	4.8
	4.7	Senkronizasyon ve	4.9
	4.8	Senkronizasyon ve	5.0

V.	ALGORİTMALARIN ANALİZİ	8.12
	5.6 CREW Tipi Bilgisayarda Sıralama	8.16
	5.1 Algoritmaların Sınıflandırılması	5.1
	5.2 Hesaplanabilir Komplekslik	5.5
IX.	5.3 Alt Sınır ve Üst Sınır	5.6
	9.1 Ayırık Fourier Dönüşümü	9.1
VI.	PARALEL ALGORİTMALAR	9.1
	9.3 Ardaşıl Hızlı Fourier Dönüşümü Algoritması	9.4
	6.1 Genel Özellikler	6.1
	6.2 Senkron Paralel Algoritmalar	6.12
	6.3 Senkron İteratif Algoritmalar	6.5
	6.4 Asenkron Paralel Algoritmalar	6.12
X.	6.4.1 Statik Ayırıştırma	6.13
	6.4.2 Dinamik Ayırıştırma	6.13
	6.4.3 Concurrent Quick Sort Algoritması	16.18
	6.4.4 Kısmi Türevli Diferansiyel Denklem	16.20
	10.3 Shuffle Çözümü	10.8
	10.4 CREW Transpose Algoritması	10.12
	10.5 Ardaşıl Matris Çarpımı Algoritması	10.16
VII.	SEÇME (SELECTION) ALGORİTMASI	10.17
	10.7 Küp Matris Çarpımı Algoritması	10.20
	7.1 Program ve Alt Sınır Algoritması	7.10
	7.2 Seçme İçin Ardaşıl Bir Algoritma	7.4
	7.3 Seçme İçin Paralel Bir Algoritma	7.9
XI.	LINEER DENKLEM SİSTEMLERİNİN ÇÖZÜMÜ	
VIII.	SIRALAMA (SORTING) ALGORİTMASI	11.1
	11.2 SİNE GAUSS-JORDAN Metodu	11.10
	8.1 Program ve Alt Sınır Metodu	8.10
	8.2 Ardaşıl Quick Sort Algoritması	8.3
XII.	8.3 ODD-EVEN TRANSPOSITION Metodu	8.4
	8.4 MERGE-SPLIT Algoritması	8.8

8.5	CRCW Tipi Bilgisayarda Sıralama	8.12
8.6	CREW Tipi Bilgisayarda Sıralama	8.16

Kaynaklar

IX. FOURIER DÖNÜŞÜMÜ

9.1	Ayrık Fourier Dönüşümü	9.1
9.2	Hızlı Fourier Dönüşümü	9.1
9.3	Ardaşıl Hızlı Fourier Dönüşümü Algoritması	9.4
9.4	Paralel Ayrık Fourier Dönüşümü	9.6
9.5	MESH FFT Algoritması	9.12

X. MATRİS İŞLEMLERİ

10.1	Ardaşıl Matris Transpoze Algoritması	10.2
10.2	Mesh Transpoze Algoritması	10.3
10.3	Shuffle Transpoze Algoritması	10.8
10.4	EREW Transpoze Algoritması	10.12
10.5	Ardaşıl Matris Çarpımı Algoritması	10.14
10.6	Mesh Matris Çarpımı Algoritması	10.17
10.7	Küp Matris Çarpımı Algoritması	10.20
10.8	CRCW Matris Çarpımı Algoritması	10.30

XI. LİNEER DENKLEM SİSTEMLERİNİN ÇÖZÜMÜ

11.1	Genel Kavramalar	11.1
11.2	SIMD GAUSS-JORDAN Metodu	11.10
11.3	MIMD GAUSS-SEIDEL Metodu	11.10

XII. PARALEL BİLGİSAYARLARDA PERFORMANS

Sonuç

ÖZSÖZ

Kaynaklar

Paralellik günümüz yaşamımızda oldukça sık karşılaştığımız bir kavramdır. İyibirlik olarak, bir işi, belli bir zamanda olan iki kişinin paralel ve eş zamanlı çalışması ile, bir kişinin yapması için gereken sürenin yarısı kadar zamanda yapabileceğini düşüncüdür. Bu kavramları ilgilendiren işlerin çoğu (fakat hepsi değil) için geçerlidir. Hızlı kılınması, hızlıca değiştirilmesi, Grotin Bandı kullanan bir fabrikadaki çalışma v.b. işler hep paralellik kavramının avantajlarının kullanıldığı yerlerdir. Bu durumlarda çalışan kişilerin sayısının artırılması işin daha erken bitirilmesini sağlar. Tabii ki bunun da bir sınırı vardır, bir belli değerinden sonra, çalışanların sayısı ne kadar artırılırsa artırılacak işin tamamlanması için gerekli zamanda daha fazla azalmaz. Bundan daha kötüye bazı işler tamamen tersi karakterdedir ve birden fazla kişi tarafından yapılamazlar. Örneğin bir maraton koşucusu daha iyi bir derece yapmak için koşması gereken yolu ikiye bölüp başkasıyla paylaşarak altın madalyayı kazanmay. Aynı şekilde

VLBI teknolojisinde yapılan çok gelişmiş yüksek performanslı paralel bilgisayarların tasarımı mümkün değildir. Bilgisayar hesaplamaları ve bilgi işleme ve verilerinin kullanıldığı algoritmaların paralellik kavramının işi altında farklı değerlendirilmeleri gerekir. Bu algoritma bilimi için, algoritmaların tasarımı yeni bir yaklaşıma sahiptir.

Yüksek performanslı bilgisayarlara gereksinime dayanan sistemler: yapısal analiz, hava tahmini, petrol arama, nükleon enerjisi uygulamaları, yazıcı zeka, uzay sistemleri, endüstriyel otomasyon, genetik mühendisliği, sosyokonomi gibi bilimsel alanlardır.

ÖNSÖZ

Paralellik günlük yaşamımızda oldukça sık karşılaştığımız bir kavramdır. İçgüdüsel olarak, bir işi, eşit beceride olan iki kişinin paralel ve eşzamanlı çalışması ile, bir kişinin yapması için gereken sürenin yarısı kadar zamanda yapabileceğini düşünürüz. Bu, insanları ilgilendiren işlerin çoğu (fakat hepsi değil) için geçerlidir. Hasat kaldırılması, mektup dağıtılması, üretim bandı kullanan bir fabrikadaki çalışma v.b. işler hep paralelliğin avantajlarının kullanılabildiği yerlerdir. Bu durumlarda çalışan kişilerin sayısının arttırılması işin daha erken, bitirilmesini sağlar. Tabii ki bunun da bir sınırı vardır, bir eşik değerinden sonra, çalışanların sayısı ne kadar arttırılırsa arttırılsın işin tamamlanması için gerekli zamanda daha fazla azaltılamaz. Bundan daha kötüsü bazı işler tamamen ardaşıl karakterdedir ve birden fazla kişi tarafından yapılamazlar. Örneğin bir maraton koşucusu daha iyi bir derece yapmak için koşması gereken yolu ikiye bölüp başkasıyla paylaşarak altın madalyayı kazanmayı ümit edemez!

İSTANBUL, 1990

VLSI teknolojisinde yapılan son gelişmeler yüksek performanslı paralel bilgisayarların tasarımını mümkün kılmıştır. Bilimsel hesaplamalar ve bilgi işlemenin tüm safhalarında kullanılan ardaşıl algoritmaların paralellik kavramının ışığı altında tekrar değerlendirilmeleri gerekir. Bu algoritma bilimi için, algoritmaların tasarımına yeni bir yaklaşım demektir.

Yüksek performanslı bilgisayarlara gereksinme duyulan alanlar: yapısal analiz, hava tahminleri, petrol aramaları, füzyon enerjisi araştırmaları, yapay zeka, uzman sistemler, endüstriyel otomasyon, genetik mühendisliği, sosyoekonomi gibi bilimsel alanlardır.

1.1

Süper bilgisayarlar olmaksızın, insanın medeniyet düzeyini yükselten bu alanlardaki çalışmaların hiçbiri kabul edilebilir sürelerde bitirilemezler. Performansın arttırılması sadece hızlı ve güvenilir donanımların kullanılmasına değil bilgisayar mimarisine ve veri işleme tekniklerine de bağlıdır.

1.2. PARALEL VERİ İŞLENME

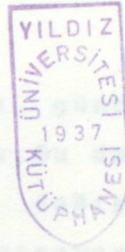
Bundan dolayı alışılmış ardaşıl algoritmalara paralelliği bilen bir gözle bakmak gerekir. Bu çalışmada da bu gerçekleştirilmiş ve bilinen ardaşıl algoritmalara karşı paralel algoritmalar ortaya konmuş ve ana hedef olan yüksek performans elde edilip edilemediğinin irdelenmesi için algoritmalar arasında karşılaştırmalar yapılmıştır.

Bu çalışmamda değerli görüşleriyle bana yol gösteren sayın Prof.Dr. M.Yahya KARSLIGİL'e , bana bu konuyu tanıtan ve projemin her safhasında destek veren Arş. Grv. Armağan ÖZKAYA'ya teşekkürü bir borç bilirim.

YILDIZ ÜNİVERSİTESİ

FÜSUN YAMAN

İSTANBUL, 1990



PARALEL ALGORİTMALARIN TASARIM ESASLARI

I. PARALEL VERİ İŞLEME

1.1 Giriş

Her zaman daha yüksek performanslı veri işleme sistemlerine karşı çeşitli talepler olmuştur. Yıllar boyunca bilgisayar alanında J. Von Neumann adlı bir matematikçi tarafından geliştirilmiş olan bilgisayar tipi kullanılmıştır. Von Neuman modelinde komutlar (instructions) ve veriler (data) aynı bellekte saklanır; tek bir işlemci (processor) komutları bellekten getirerek işler ve veriler üzerinde değişiklikler yapar. Bu işlemler birbiri ardına tek tek yapılırlar. Sonraları daha ucuz, hızlı, fiziksel olarak küçük, mantıksal olarak büyük sistemler gerçekleştirildiyse de bunların da temellerinde Von Neumann modeli yatmaktaydı. Bu tip bilgisayarların elektrik sinyallerinin yayılım hızlarının sınırlılığı nedeniyle 1 saniyede en fazla 10 üstü 10 işlem yapılabilecekleri hesaplanmıştır. Buna rağmen bu sayı teoride kalmıştır. Günümüzde en hızlı bilgisayarlar saniyede 10 üstü 8 işlem yapılmaktadır. Bugün yapısal analiz, hava tahminleri, petrol ve enerji araştırmaları, tıbbi teşhis, aerodinamik simülasyonlar, uzman sistemler, yapay zeka, endüstriyel otomasyon, sivil savunma, genetik mühendisliği v.b. konularda daha yüksek performanslı sistemlere ihtiyaç vardır. Bu süper güç olmasa yukarıda sayılan işlemlerin kabul edilebilir bir zamanda çözümlenmesi imkansızdır. Uygulamaların böyle gitgide daha kapsamlı olması ve daha büyük hızlara ihtiyaç duyulması, çok fazla sayıda işlem gerektiren problemlerin belirli, kısa bir

zamanda çözülmesinin gerekliliği günümüzde bilgisayarların kullanımını yetersiz kılmaktadır. Bu darboğazların sonucunda bilgisayar yapısında kaydadeğer değişiklikler başlamıştır. Bu yüksek performans, yeni bilgisayar mimarileri ve yeni veri işleme teknikleri ile sağlanmıştır. Bilgisayar donanımındaki maliyet düşüşleri sonunda gelişmiş bilgisayar mimarileri paralel veri işleme kavramı çevresinde oluşmuştur.

1.2 Paralel Veri İşleme

Paralel veri işleme bir takım çalışması olarak tanımlanabilir. Bir futbol takımında her oyuncunun kendine ait bir görevi vardır. Tüm bu işler aynı anda gerçekleştirilmektedir, fakat hepsi birbirinden farklıdır, hepsi farklı oyuncular tarafından yapılır. Bu paralel veri işleme (PARALLEL PROCESSING) dir. Bir caddede dökülen yaprakları tırmıkla toplayan bir grup insan düşünün hepsi aynı işi yaparak iş için gerekli zamanı azaltmaktadırlar. Bu da paralel veri işlemedir. Aynı temel kavram bilgisayara uygulanırsa: Paralel çalışan birden fazla işlemci bir işi, ama tüm işleri değil, tek bir işlemciden daha çabuk yaparlar. Mantıksal olarak ardışıl olan bir program veya modüler hale getirilebilen bir programın farklı bölümleri farklı işlemciler üzerinde çalıştırılarak hız arttırılabilir. Paralel olarak çalışan bir bilgisayarda işler birden fazla işlemci üzerinde eş zamanlı (simultaneously) olarak gerçekleştirilebilir. Von Neumann tipi bilgisayarların en büyük darboğazı veri bellekten getirilip işlemci saklayıcılarına (processor registers) saklanırken CPU 'nun boş kalmasıdır.

Paralel veri işleme, hesaplamalarda concurrent olaylardan yararlanan efektif bir veri işleme formudur. CONCURRENCY paralellik (Parallelism), eşzamanlılık (simultaneity) ve pipelining kavramlarını ifade eder. (Bundan dolayı çalışma

içinde concurrent ve concurrency kelimeleri orijinal şekilde kullanılacaktır). Paralel olaylar birden fazla kaynaktan aynı zamanda meydana gelebilirler. Eşzamanlı olaylar aynı anda oluşurlar. Pipelined olaylar üst üste binmiş bir zaman bandında görülürler. Bu concurrent olaylara bir bilgisayar sisteminde farklı işlem düzeylerinde erişilir. Paralel veri işleme pek çok programın bilgisayarda concurrent olarak yürütülmesini gerektirir. Ardaşıl (sequential) veri işlemenin tersine paralel veri işleme concurrent aktiviteler sayesinde sistemin performansını yükseltir. Paralel veri işlemenin en üst düzeyi pek çok iş veya programı çoklu programlama (multiprogramming), zaman paylaşım (time sharing), çoklu işlemcili (multiprocessing), ortamda yönetmektir.

Bu düzey, paralel işlem gören PARALEL ALGORİTMALARIN geliştirilmesini gerektirir. Paralel algoritmaların geliştirilmesi kısıtlı yazılım ve donanım kaynaklarının büyük bir bilgisayarla çözümünde kullanılan pek çok programa efektif olarak tahsis edilmesine bağlıdır.

Bir sonraki düzeyde paralel veri işleme aynı program içindeki procedure ve işlerin (tasks) yönetilmesidir. Bu procedure ve işlere programın bölümleri (program segments) denebilir. Bu tip paralellik programın pek çok işe ayrıştırılmasını (DECOMPOSITION) gerektirir.

Üçüncü düzey, concurrent olma özelliğinden birden fazla komut arasında yararlanmaktır. Komutlar arasındaki paralellik açığa çıkartmak için veri bağımlılığı analizi yapılır.

Son olarak her komut (instruction) içinde daha hızlı concurrent operasyonlar isteyebiliriz. Özet olarak paralel veri işleme dört dört programlama düzeyinde belirlenir.

* İş (Job) veya program düzeyi

* Task veya procedure

* Komutarası (Interinstruction)

* Komut içi (Intrainstruction)

En yüksek dereceden iş düzeyi algoritmik olarak gerçekleştirilir. Belirtilen sırada yukarıdan aşağıya gidildikçe yüksek seviyeli paralel veri işlemekten alçak seviyeli paralel veri işlemeye doğru gidilir ve paralellik yazılımla değil donanımla sağlanmaya başlanır. Paralel bir sistem tekli işlemcili (uniprocessor) veya çoklu işlemcili (multiprocessor) olabilir. Tekli işlemcili sistemlerde paralellik yardımıyla performans sağlama imkanı sınırlıdır.

Kısacası bir anda sadece bir komut yürütebilen bir makina kullanımından pek çok sayıda işlemin eşzamanlı (simultaneously) olarak yürütülmesi fikri ortaya çıkmıştır. Bu fikirler PARALELLİK PRENSİBİ adı altında yeni bilgisayar mimarilerinin geliştirilmesine yol açmıştır. Paralel veri işleme (PARALLEL PROCESSING) de yapılan araştırmalar üç ana alanda yapılmaktadır. Bunlar:

1.) Bilgisayar mimarisi (donanım ve işletim sistemi)

2.) Algoritmalar ve uygulamalar

3.) Programlama dilleri

Paralel veri işlemede paralel bilgisayarların tasarımı kadar kullanılacak algoritmalar da önem kazanmaktadır. Bu çalışmada paralellik prensibine uygun yazılan programların

karakteristikleri tanıtilacak, metodlar ve örnekler verilecek, proformanslar karşılaştırılacaktır. Paralel algoritmalar üzerinde çalışacakları bilgisayarın mimarisine son derece bağımlı olduklarından sistem mimarileri gözönünde bulundurulduğunda paralel algoritmalar belli başlı üç gruba ayrılır. Bunlar 1.) Array İşlemciler için 2.) Çoklu işlemciler için 3.) Tekli işlemciler için. Bu çalışmada genellikle ikinci gruptaki algoritmalar incelenmiştir.

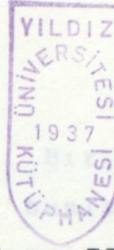
Yapılması gereken uygulamada ne tip bir yapıya (GRANULARITY) gerek duyulduğu karar verektir. (GRANULARITY) parçaları parçaları yapıya olarak kullanılmıştır. 1.3 Paralel Yaklaşım

Paralel veri işleme özelliğine sahip sistemlerde en büyük sorun birden fazla işlemcinin efektif ve etkili olarak aynı zamanda nasıl kullanılabileceğidir. Paralel programlamada pek çok işlemci (processor) arasında verilerin ve sonuçların gidip gelmesi, her işlemcinin ihtiyaç duyduğu veriyi gerektiği zaman elde etmesi için işlemciler arasında iletişim ve koordinasyon gerekir.

Sistemin efektifliği, problemin paralellik prensibine uyum sağlamasına, algoritmaya ve uygun bir mimari yapıya bağlıdır. Birden fazla işlemcinin aynı bellek bölgesine erişmesinin, aynı anda buradaki bir veriyi güncelleştirmek istemesinin sorunlar yaratacağı aşıkardır. Bundan dolayı çoklu işlemcili sistemlerin tek işlemcili sistemlere göre daha hızlı olacağı fikri ancak sistem (yukarıdaki problemlerin üstesinden gelebiliyorsa ve uygun senkronizasyon mekanizmasına sahipse doğrudur.

Paralel algoritmalar yazan kişiler mutlaka hedef mimariyi gözönünde bulundurmalıdır. Programcı bazen programa iki zıt yönden bakmalıdır. Algoritmanın dizaynı için Top-Down, gerçekleştirme için Bottom-Up. Kullanım için A.B.D. deki uçak şirketlerinde yapılan rezervasyonlar örnek

verilebilir. Rezarvasyon işlemleri sırasında günde 200-250 bin değişiklik olabilmektedir. Eğer böylesine büyük bir sorunla karşı karşıya bir programcı iseniz önce işi büyükçe dilimlere ayırır sonra da bu dilimler arasındaki senkronizasyonu nasıl sağlarız diye düşünürsünüz. Şanssızlık eseri bir çözüm için kullandığınız yaratıcılığınız işletim sistemi tarafından veya hedef donanım mimarisi tarafından engellenebilir. Böyle bir durumda ilk yapılması gereken uygulamanın ne tip bir ayrıştırmaya (GRANULARITY) gerek duyduğuna karar vermektir. (GRANULARITY parça parçalık, ayrıştırma olarak kullanılmıştır.) Ayrıştırma ile paralel bir sistemde bu parçalar arasında iletişim kurma arasında yakın bir ilişki vardır. Kabaca bölümlere ayrılmış bir uygulamada paralel veri işleme sistemi bireysel işlemciler arasındaki iletişim için harcanan küçük bir zaman ile geniş bağımsız iş birimlerinden oluşur. Yukarıda söz konusu olan küçük zaman diyelimki iki işlemci arasında saniyede 100 iletişim yapılan bir zamandır. İyi ayrıştırılmış (bölümlenmiş) bir sistemde işlemler arasında iletişim ve senkronizasyon daha fazladır (saniyede milyonlarca iletişim). Programcı çözümünü bulunan mimariye dayamalıdır. Uçak örneğinde iyi ayrıştırılmış bir sistemde bir işlemci (processor) bir uçuşa kabaca ayrıştırılmış bir sistemde bir işlemci bir uçağa karşı gelir. Bir kez bu ayrıştırmaya karar verildikten sonra işlemler, dolayısıyla işlemciler arasındaki ilişkilerin nasıl olacağına karar verilir. Bu Paylaşılan bellek (SHARED MEMORY) üzerinden veya dağıtık bir sistemde (DISTRIBUTED SYSTEM) mesaj geçirme (MESSAGE PASSING) metodu ile olur. İşlemciler ilk metodu yani ortak belleği kullanıyorlarsa burada yer alan ortak bir veri yapısı üzerinden işlemler yapılır. Bunun yanı sıra atomik operasyonlar (LOCK'lar, SEMAPHORE'lar, MONITOR'lar) işlemlerin uygun olarak senkronizasyonları için kullanılırlar ki bunlar ileride ayrıntılı olarak açıklanacaktır.



II. PARALEL BILGISAYARLARIN TANIMI ve SINIFLANDIRILMASI

Sayısal bilgisayarların gelişmeye başladığı ilk yıllardan beri, bilgisayarın yaptığı işlemlerin hızını arttırmak tasarımcıların amacı olmuştur. Bunu gerçekleştirmenin birkaç yolu vardır. Bunlardan biri bilgisayar devrelerinin yapımında kullanılan teknolojinin iyileştirilmesidir. Şu anda mevcut teknoloji büyük bir yol olarak bugünkü durumuna gelmiştir ve çalışmalar devam etmektedir. Fakat bu konuda doğal sınırlamalar vardır; hiçbir sinyal ışık hızından daha hızlı yayılamaz.

Işığın boşluktaki yayılma hızı $3 \cdot 10^8$ m/s'dir. Şimdi saniyede 10^{12} işlem yapabilen bir elektronik devre düşünelim. Birbirinden 1 milimetrenin yarısı kadar uzaklıkta olan böyle iki devre arasında bir sinyalin gıtmesi daha uzun zaman alacaktır. Bir başka deyişle, süper hızlı elektronik bileşenlerin yapılmasıyla sağlanan hız bir devre bileşeni diğerinden mesaj beklerken kaybedilecektir. O halde devre elemanları birbirine yaklaştırılsın denebilir. Ama fizikçilere göre elektronik devreler arasındaki uzaklık azaltıldıkça öyle bir noktaya gelinirki devreler arası etkileşim başlar ve sinyallerin güvenilirliği kaybolur.

İkinci yol, bilgisayar sistemlerinde kullanılan lojik devre tasarımını iyileştirmektir. Buna örnek olarak toplamada "'Carry Look Ahead'" ve çarpmada "'Booth Algoritmaları'" verilebilir. Çeşitli problemlerin çözümü için kullanılan algoritmaların iyileştirilmesi de hızı artırıcı etki yapar.

Geleneksel Von Neumann mimarisinde bir bilgisayarda

işlemler seri esasa göre yapılır. Bir komut alınır, kodu çözülür eğer operandları varsa bunlarda alınır, komut işlenir. Sonuç saklanır. Bir ikinci işlem ancak kendinden önceki işlemin bütün bu aşamaları bitince işe başlıyabilir. İki işlemin çalışmaları sırasında üstüste binme zamanı yoktur. Her CPU bir ALU 'ya sahiptir ve bu ALU sistemin tüm veri işleme işlerini yapar.

Daha sonra hızı arttırmak için ''önceden getirme'' PREFETCH ve pipeline kavramları kullanılmıştır. Ama bütün bu yollar denendikten sonra bile bilgisayarda işlemlerin hızını arttırmak için daha iyi bir yol vardır. Mümkün olduğu kadar fazla sayıda işlem, ard arda yapılacağı yerde, aynı anda, eşzamanlı yapılırsa hesaplamaların hızı artar.

2.1 TANIM

Paralel processing kavramı bir bilgisayar sisteminde işlemlerin iki veya daha fazlasının eşzamanlı (simultanesusly ve concurrent) çalışmasını sağlayan bir organizasyon metodu olarak tanımlanabilir.

Paralel veri işleme çok genel bir kavramdır. Gerçekleştirilmesi için pekçok yöntem kullanılabilir. Sadece, Paralel Veri İşleme sözü hangi tip bir sistemde kullanıldığını ve nelerin gerçekten eş zamanlı yapıldığını anlatmaya yetmez. Paralel veri işleme sistemlerinin sınıflandırılması gerekir.

Bu tezde paralel olarak bilgisayar mimarileri üç ayrı sınıflandırma metodu incelenmiştir. Bunlar;

- 1) Michael J.Flynn'ın sınıflandırılması(1966)
- 2) Tse-yun Feng'ın sınıflandırılması(1972)
- 3) Wolfgang Höndler'ın sınıflandırılması(1977) dir.

2.2. FLYNN'IN SINIFLANDIRILMASI:

Sınıflandırma metodları arasında geniş kabul görmüş ve çok bilinen bir metoddur. Flynn'in Sınıflandırması makinanın yapısına değil, makinadaki veri ve komutlar arasındaki ilişki temeline dayanır. Veri ve komutların tek veya çok olmasına dikkat edilmiştir.

Flynn'in metodunda aşağıdaki kavramlar kullanılır.

-Pr : Bir ALU ve gerekli ek lojik donanım (Stream) sahip işlemci (processor) verilerle işlem yapan donanımın çoğunluğu ile karakterize edilmiştir.

-CU : Kontrol ünitesi (Control Unit). İşlemcileri ve diğer alt sistemleri, lojik kontrol sinyallerinin iletişimi ile kontrol eder. Komutların kod çözümünü (decode) yapar ve işlenmesi için işlemciye gönderir.

-MM : Ana bellek (Main memory) genellikle belli sayıda bellek modüllerine bölünür.

-LM : Yöresel bellek (Local Memory) Sadece belli bir işlemciye ait bellek.

-I : Komut, (instruction) veya komut dizisi, kontrol sinyali.

-D : Veri dizisi (data). İşlemci ve bellek arasında iki yönde hareket ederler.

-P : Hem bir ALU 'ya hemde CU 'ya sahip tam bir işlemci (processor).

Flynn, mimarileri, komutları ve verileri işleyen donanımın

tekliğine veya çokluğuna göre kategorize etmiştir. Çokluk eşzamanlı işlemlerin mümkün olan maksimum sayısı olarak veya organizasyonun en zorlayıcı parçasındaki aynı yürütme safhasındaki operandların sayısı olarak alınır.

Bilgisayar ile yapılan temel işlem bir veri kümesi üzerinde yapılan bir sıra komuttur. Komut kümesi her adımda bilgisayara yapacağı işleri bildiren komutlardan oluşur (Instruction stream). Veri kümesi giriş veya geçici ara değerler olup komut kümesi tarafından çağrılarak kullanılır, sonuçlardan etkilenir (Data Stream). Bilgisayarın organizasyonu komut ve verilerle işlem yapan donanımın çokluğu ile karakterize edilmistir.

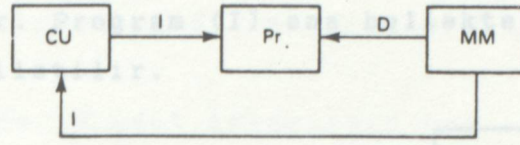
FLYNN' ın 4 TİP MİMARİSİ

1. SISD :

(Single Instruction Stream Single Data Stream)

Temel, tek işlemcili (Uniprocessor) sistemdir. Geleneksel seri, Von Neumann bilgisayarıdır. Tek bir komut kumesi (dolayısıyla pratikte bunu işleyecek tek bir işlemci) vardır ve her aritmetik komut mantıksal olarak birbirine bağlı argüman ve sonuçları üzerinde işlem yapan aritmetik işlemi başlatır.

Hesaplamalar sırasında, her adımda, kontrol birimi, bellek biriminden alınan tek bir veri üzerinde işlem yapacak olan, tek bir komutu işlemciye gönderir. Bu tip bir komut işlemciye veri üzerinde aritmetik veya lojik bir işlem yaptıktan sonra sonucu tekrar belleğe yazmasını isteyebilir. Bugünkü bilgisayarların çoğu 1940' ların sonunda Von Neumann tarafından ortaya konan bu esasa göre çalışmaktadır

FIGURE 1.1a. SISD.

Bu tip mimariye göre tasarlanmış bilgisayarlar için kurulan algoritmalar seri (sequential) algoritmalar denir. Yapıda hiç bir paralellik yoktur. Bunun nedeni ise gayet açıktır. Sadece 1 işlemci vardır. Bu Bilgisayardan paralel işlem yapmasını istemek için en az 2 işlemciye sahip olması gerekir.

Örneğin, bu tip bir bilgisayar n tane sayının toplamının bulunması için belleğe ard arda n defa erişmek ve her defasında bir sayı almak gerekir. Aynı zamanda $n-1$ adet toplama işlemi ile sonuç hesaplanır. Bundan dolayı bu hesaplama toplam olarak n .mertebeden işlem gerektirir.

2. SIMD :

(Single Instruction Stream Multiple Data Stream)

Bu tip mimariye sahip bir bilgisayarda N adet, birbirinin tamamen aynı, işlemci bulunur. Her işlemcinin kendine ait, hem verileri hemde komutları saklayabildiği bir yöresel belleği (Local Memory) vardır. Tüm işlemciler tek bir kontrol biriminin (CU) kontrolü altındadır. Eşdeğer olarak, her işlemcinin bir programın (I işlenecek komutlar) kopyasını kendi yöresel belleğine kopyaladığı kabul edilir. Ayrıca her işlemci bu yöresel belleğinde üzerinde işlem yapacağı, kendine özgü, diğer işlemcilerinkinden farklı, veri dizisini (Data Stream)de saklar. Her biri birbirinden farklı, N adet veri dizisi vardır.

İşlemciler ve yöresel bellekler arasında veri iletişimi için kullanılan iki yönlü bir yol (bus), yine yöresel bellekler

ve ana bellek (MM) arasında verilerin iletildiği iki yönlü bir yol vardır. Program (I) ana bellekte bulunur ve gerekli zaman CU 'e iletilir.

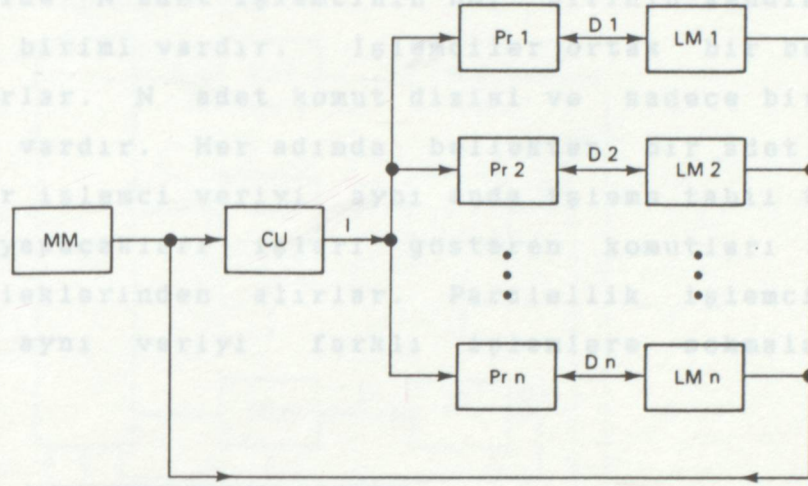


FIGURE 1.1b. SIMD.

İşlemciler senkron çalışırlar. Her adımda, tüm işlemciler farklı veri üzerinde aynı işlemi yaparlar. Komut iki sayıyı toplama veya carpma gibi basit olabileceği gibi, iki sayı dizisinin merge edilmesi gibi kompleks bir işlem de olabilir. Aynı şekilde veri tek bir sayı olabileceği gibi bir dizi de olabilir. Bu tip sisteme Array Processor adı da verilebilir; çünkü işlemciler $Pr_i = i = 1, 2, \dots, n$ olarak bir dizi formundadır. Bilgisayar tek bir komutu yürütür fakat bu bir vektör komutudur ve birden çok işlemi baslatır. Vektörün her bir elemanı ayrı bir veri dizisidir. Bu tip bilgisayarların hepsinde vektör özelliği bulunur. Vektör işleme özelliğinin pipelining ile mi yoksa array processorler ile mi sağlandığı şu anda bizi ilgilendirmiyor. Bu tip bilgisayarlarda kullanılan word-slice, bit-slice, interconnection network, shared memory v.b. konular parallel bilgisayarların donanımlarının daha detaylı incelendiği IV.Bölüm de anlatılmıştır.

3. MISD : (Multiple Instruction Stream Single Data Stream)

Bu tip mimaride N adet işlemcinin her birinin kendine ait bir kontrol birimi vardır. İşlemciler ortak bir belleği (MM) paylaşırlar. N adet komut dizisi ve sadece bir tane veri dizisi vardır. Her adımda bellekten bir adet veri alınır ve her işlemci veriyi aynı anda işleme tabii tutar. İşlemciler yapacakları işleri gösteren komutları kendi yöresel belleklerinden alırlar. Paralellik işlemcilerin aynı anda, aynı veriyi farklı işlemlere sokmalarıyla sağlanır.

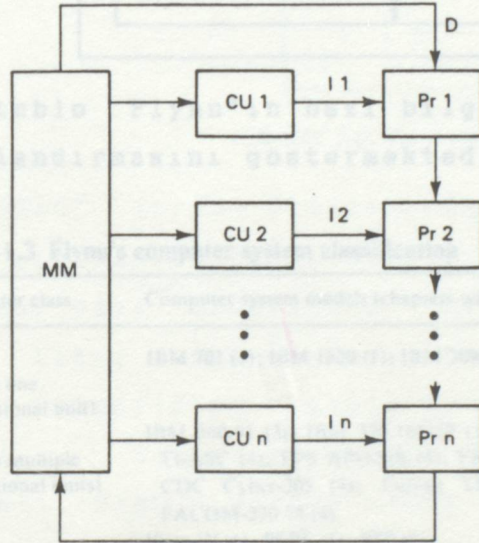


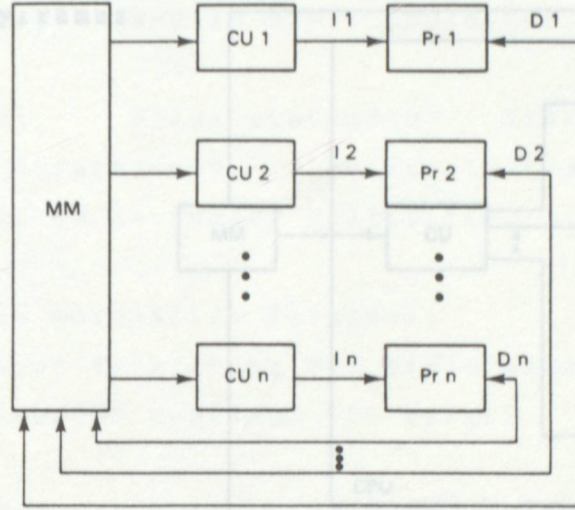
FIGURE 1.1c. MISD.

4. MIMD : (Multiple Instruction Stream Multiple Data Stream)

Mimaride N adet işlemci ve her işlemcinin kendine ait kontrol birimi, yöresel belleği ve aritmetik lojik birimi bulunur. Her işlemci kendi kontrol biriminin gönderdiği komutları yapar. Bundan dolayı, bir problemin alt problemlerini çözerken, tüm işlemciler farklı işler yaparlar. Bu da onların asenkron çalışması demektir. İşlemciler arası iletişim yüksek ise bunlara (Tightly

coupled), ortak bellek kullanan MIMD bilgisayarlar, düşük ise (Loosely coupled), interconnection network kullananlar, bilgisayarlar denir.

FIGURE 1.1d. MIMD.



Aşağıdaki tablo Flynn'in bazı bilgisayarları bu dört tipe göre sınıflandırmasını göstermektedir.

Table 1.3 Flynn's computer system classification

Computer class	Computer system models (chapters where the system is quoted or described)
SISD (uses one functional unit)	IBM 701 (1); IBM 1620 (1); IBM 7090 (1); PDP VAX11/780 (1).
SISD (with multiple functional units)	IBM 360-91 (3); IBM 370.168UP (1); CDC 6600 (1); CDC Star-100 (4); TI-ASC (4); FPS AP-120B (4); FPS-164 (4); IBM 3838 (4); Cray-1 (4); CDC Cyber-205 (4); Fujitsu VP-200 (4); CDC-NASF (4); Fujitsu FACOM-230 75 (4).
SIMD (word-slice processing)	Illiac-IV (6); PEPE (1); BSP (6)
SIMD (bit-slice processing)	STARAN (1); MPP (6); DAP (1).
MIMD (loosely coupled)	IBM 370.168 MP (9); Univac 1100.80 (9); Tandem 16 (9); IBM 3081.3084 (9); C.m* (9)
MIMD (tightly coupled)	Burroughs D-825 (9); C.mmp (9); Cray-2 (9); S-1 (9); Cray-X MP (9); Denelcor HEP (9)

SONUC:

Flynn'in sınıflandırması biraz eski modadır. Sık kullanılmasına rağmen genel tanımlar yapar ve sistemlerin

önemli detaylarını açığa çıkarmaz, gösteremez. Örneğin pekçok işlemci (uniprocessorler veya MIND sistemlerin içinde olanlar) aritmetik ve komut pipeline'larına veya her ikisine de sahiptirler. Flynn'ın sınıflandırması bu tip işlemcileri ayıramaz.

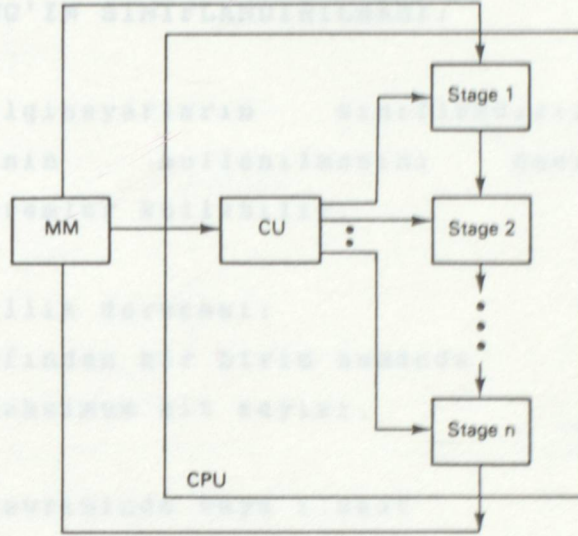


FIGURE 1.2. An n-stage pipeline.

Gerçekte 1980'lerin ortalarından sonra üretilen çoğu bilgisayar pipelining özelliğine sahiptir. Ayrıca eksperler arasında da bazı mimarileri hangi sınıfa koyacakları konusunda fikir ayrılıkları vardır. Örneğin Cray-1 pipelining özelliğine sahip bir vektör işlemcili bilgisayardır. Vektör işlemcili olduğu için Hockney ve Jesshop(1981) SIMD mimariye örnek gösterirler. Birden çok işlemcisi olmadığı için Hwang ve Briggs(1984) SISD sınıfına sokarlar. Flynn'ın ki ise çok geneldir. Multiprocessor'ler dışında tüm paralel bilgisayarları SIMD sınıfına sokar. Pipeline bilgisayarlar ile Processor Array'ler arasında fark gözetmez. Flynn'ın sınıflandırmasındaki bir başka belirsizlik de Data Stream kavramıdır. Pipelining özelliğine sahip vektör bilgisayarlar Hockney(1977) tarafından, vektör özelliğindeki tek bir veri dizisi işledikleri için SISD sınıfına dahil edilirler. Oysa Flynn(1972) bunları SIMD sınıfında hatta bazı araştırmacılar hem SIMD hem de MIMD sınıfında kabul ederler. Çünkü Pipelining özelliğindeki bir aritmetik

birime sahiptir. Bu da vektör özelliğindeki tek bir veri veya skaler birçok veri üzerinde işlem yapmaya eşdeğerdir.

bu işlemler için $\lambda = 100$ fayda için $\lambda = 1$ çıkar. Çoğaltılarak uygulanan programına bağlıdır.

2.3. FENG'IN SINIFLANDIRILMASI:

Feng paralel bilgisayarların sınıflandırılmasında paralellik derecesinin kullanılmasını önermiştir. Sınıflandırmada şu kavramlar kullanılır.

- P : Maksimum paralellik derecesi:
Bilgisayar tarafından bir birim zamanda işletilebilen maksimum bit sayısı.
- P_i : i. işlemcinin çevriminde veya i. saat çevriminde işletilebilen maksimum bit sayısı.
- T : İşlemci çevriminin adedidir.
 $i = 1, 2, \dots, T$
- P_a : Ortalama paralellik derecesi.

$$\sum_{i=1}^T P_i$$

$$P_a = \frac{\sum_{i=1}^T P_i}{T} \text{ dir.}$$

Genelde $P_i \leq P$ dir. Sonuç olarak T adet çevrim için λ , fayda oranı şu şekilde hesaplanır.

$$\eta = \lambda = \frac{P_a}{P} = \frac{\sum_{i=1}^n P_i}{T * P} \text{ dir.}$$

Eğer işlemciden tam olarak yararlanılıyorsa, veya paralellik tam anlamıyla ortaya çıkarılmışsa $P_i = P$ dir, tüm i 'ler için %100 fayda için $\Lambda = 1$ çıkar, çalıştırılan uygulama programına bağlıdır.

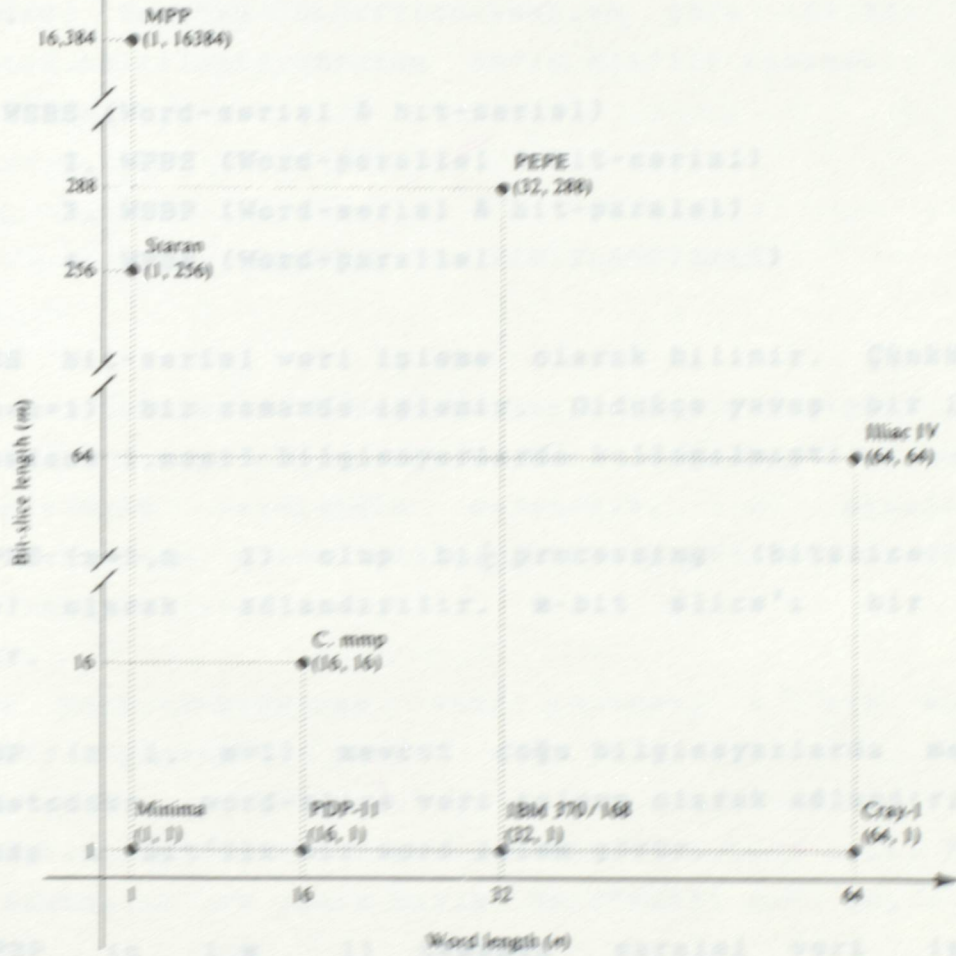


Figure 1.37 Feng's classification of computer systems in terms of parallelism exhibited by word length and bit-slice length.

Yatay eksen word genişliğini (n), dikey eksen bit-slice genişliğini (m) gösterir. Bunlar bit miktarlarını verir, "Bit slice" bitlerden oluşan bir katarıdır. $P(C)$; C bilgisayarlarının maximum paralellik derecesi word genişliği olan n ile bit-slice genişliği olan m 'in çarpımı ile bulunur.

$$P(C) = n.m$$

(n, m) çifti koordinat sisteminde bir nokta verir. $P(C)$ ise oluşan dikdörtgenin alanıdır.

Örneğin Carnegie Mellon C.mmp 16 bit word genişliği olan 16 işlemci. Bu diyagramda görüldüğü gibi 4 tip veri işleme metodu vardır.

Feng'in sınıflandırmasına göre $(16, 16)$ sayı çifti ile belirtilir. Sistem 16×16 bit'lik eşzamanlı işlem yapar.

1. WSBS (Word-serial & bit-serial)
2. WPBS (Word-parallel & bit-serial)
3. WSBP (Word-serial & bit-parallel)
4. WPBP (Word-parallel & bit-parallel)

1. WSBS bit-serial veri işleme olarak bilinir. Çünkü bir bit $(n=m=1)$ bir zamanda işlenir. Oldukça yavaş bir işlem olup sadece 1. nesil bilgisayarlarda kullanılmıştır.

2. WPBS $(n=1, m=1)$ olup bit-slice processing (bitslice veri işleme) olarak adlandırılır. m -bit slice'ı bir anda işlenir.

3. WSBP $(n=1, m=1)$ mevcut çoğu bilgisayarlarda mevcut olan metoddur. word-slice veri işleme olarak adlandırılır. Bir anda n bit'lik bir word işlem görür.

4. WPBP $(n=1, m=1)$ tamamen paralel veri işleme gerçekleştirilir. $n.m$ bitlik bir dizi bir anda işlenir.

Feng'in 4 modu arasında en hızlı olanıdır.

Table 1.4 Feng's computer systems classification

Mode	Computer model (manufacturer)	Degree of parallelism (n, m)
WSBS $n = 1$ $m = 1$	The "MINIMA" (unknown)	$(1, 1)$
WPBS $n = 1$ $m > 1$ (bit-slice processing)	STARAN (Goodyear Aerospace) MPP (Goodyear Aerospace) DAP (ICL, England)	$(1, 256)$ $(1, 16384)$ $(1, 4096)$
WSBP $n > 1$ $m = 1$ (word-slice processing)	IBM 370/168 UP CDC6600 Burrough 7700 VAX 11/780 (DEC)	$(64, 1)$ $(60, 1)$ $(48, 1)$ $(16/32, 1)$
WPBP $n > 1$ $m > 1$ (fully parallel processing)	Illiac IV (Burroughs) T1-ASC C.mmp (CMU) S-1 (LLNL)	$(64, 64)$ $(64, 32)$ $(16, 16)$ $(36, 16)$

Örneğin Carnegie Mellon C.mmp 16 bit word genişliği olan 16 işlemciye sahip bir çok (ve işlemcili) (multiprocessor) sistemdir. Feng'in sınıflandırmasına göre (16,16) sayı çifti ile belirlenir. Sistem 16*16 bit'lik eşzamanlı işlem yapılabilir. PCU sayısı (macropipelining) D' Pipeline

özellikleri sağlayacak şekilde birbirine bağlanan ALU sayısı, W' Tüm ALU'lardaki veya bir PE'deki pipeline 2.4.2. HANDLER 'in SINIFLANDIRMASI

Wolfgang Handler, bilgisayarların pipelining yapılıp yapılmadığını, pipeline safhalarının sayısını gözönünde bulundurularak paralellik derecesini ve pipelining derecesini ifade etmek için yeni bir sınıflandırma metodu geliştirmiştir (1977).

Handler paralel-pipeline PU veri işlemeyi 3 alt sistem düzeyinde düşünmüştür.

1. İşlemci kontrol birimi (processor control unit, PCU)
2. Aritmetik ve lojik birim (Aritmetik and Logic Unit ALU) ise paralel çalışabilir.
3. Bit düzeyinde devre (Bit_level circuit BLC) dir.

Bir PCU bir işlemciye veya bir CPU 'ya karşılık gelir. Bir ALU bir fonksiyon unitesine veya bir array processor'un bir işlem elemanına (PE) karşılık gelir ve BLC ise ALU içinde tek bir bit işlem yapmak için gerekli kombinasyonel lojik tasarıma karşı gelir.

Bir C bilgisayarı, 3 çift tamsayı ile karakterize edilir.

$$T(C) = K \times K', D \times D', W \times W'$$

Burada: TI-ASC Instrument'ın Advanced Scientific Computer'ın
(TI-ASC) bakıldığında 4 aritmetik pipeline'ın kontrol eden bir

K: Bilgisayarın içindeki işlemci sayısı (PCU), D: Bir PCU
'nın kontrolü altındaki ALU (veya PE) sayısı, W: ALU (veya
PE) 'nin 1 wordundeki bit sayısı, K': Pipeline özelliğini
sağlayacak şekilde birbirine $x \cdot 8) = 1,4,64 \times 8$ $K'=D'=W'=1$
oldu bağlanan PCU sayısı (macropipelining) D': Pipeline
özelliğini sağlayacak şekilde birbirine

bağlanan ALU sayısı, W': Tüm ALU'lardaki veya bir PE
deki pipeline safhalarının
sayısı,

Tek CPU'su vardır ve 12 pipeline özelliği ile ilgili
olarak tanımlanır. Bir çiftteki 2. eleman 1 ise yazardır
denilir. X operatörü bir bilgisayar sistemi içindeki farklı
tipteki işlemcilerin bağlantı tarifleri için kullanılır.
Şimdi birkaç örnekle bu tanımları ve formülü gerçekleyelim.

a) *CDC 6600 nasıl sınıflandırılır ?

*CDC 6600 'ün 1 tane CPU'su vardır. $K=1$ $K'=1$

c) *ALU 10 tane fonksiyon ünitesine sahiptir.

*Bu 10 tane fonksiyon hepsi pipeline olabilir.

*Word genişliği 60 bittir.

*10 adet çevre I/O işlemcisi bir diğeri ile veya CPU
ile paralel çalışabilir.

*Her I/O işlemcisi word genişliği 12 bit olan
bir ALU içerir.

$$T(\text{CDC 6600}) = T(\text{merkezi işlemci}) \times T(\text{G/Ç işlemcileri})$$

$$= 1, 1 \times 10, 60 \times 10, 1, 2$$

Table 1.5 Händler's computer system classification

Computer model $T(C)$	System specification† $\langle K \times K', D \times D', W \times W' \rangle$
$T(\text{TI-ASC})$	$\langle 1, 4, 64 \times 8 \rangle$
$T(\text{CDC-6600})$	$\langle 1, 1 \times 10, 60 \rangle \times \langle 10, 1, 12 \rangle$ central I/O processor processors
$T(\text{Illiac IV})$	$\langle 1, 64, 64 \rangle$
$T(\text{MPP})$	$\langle 1, 16384, 1 \rangle$
$T(\text{C.mmp})$	$\langle 16, 1, 16 \rangle + \langle 1 \times 16, 1, 16 \rangle + \langle 1, 16, 16 \rangle$
$T(\text{PEPC})$	$\langle 1 \times 3, 288, 32 \rangle$
$T(\text{IBM 360/91})$	$\langle 1, 3, 64 \times (3 \sim 5) \rangle$
$T(\text{Prime})$	$\langle 5, 1, 16 \rangle$
$T(\text{Cray-1})$	$\langle 1, 12 \times 8^{\dagger}, 64 \times (1 \sim 14) \rangle$
$T(\text{AP-120B})$	$\langle 1, 2, 38 \times (2 \sim 3) \rangle$

† K' , D' , and W' are omitted when equal to 1.

‡ For Cray-1, the pipeline chaining degree is a variable with a maximum value equal to 8.

b) Texas Instrument'ın Advanced Scientific Computer'ine (TI-ASC) bakalım 4 aritmetik pipeline'i kontrol eden bir controller'a sahiptir. Her pipeline 64 bit'lik word'e ve 8 sayfaya sahiptir.

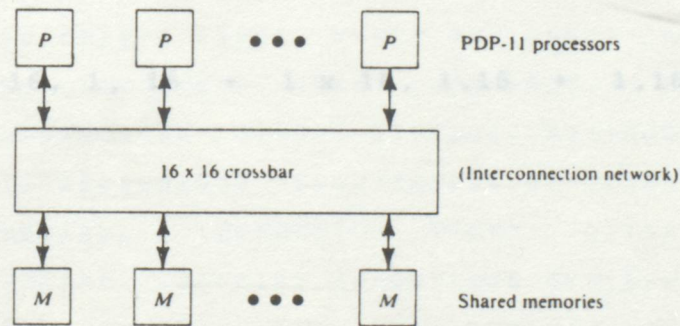
$T(ASC) = 1 \times 1, 4 \times 1, 64 \times 8 = 1, 4, 64 \times 8$ $K'=D'=W'=1$ olduğu için atladık. Çünkü 1 birimle 1 sayfada pipelining anlamsızdır.

c) CRAY_1

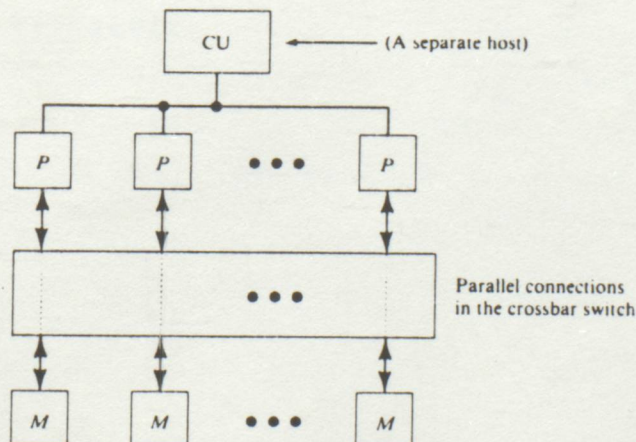
Tek CPU'su vardır ve 12 pipeline özelliği ile bağlı fonksiyon ünitesine sahiptir. Farklı fonksiyon üniteleri 1 ila 14 sayfaya sahiptir. Bir pipeline yapmak için max 8 fonksiyon ünitesi birbiriyle bağlanır. Cray-1'in 1 wordü 64 bittir.

$T(Cray-1) = 1, 12 \times 8, 64 \times (1 \text{ } 14)$

c) Cornell Melon Üniversitesi tarafından yapılan C.mmp birkaç farklı tipte çalışabilir (farklı işlem modunda)



(a) $T(16, 1, 16)$ for MIMD mode



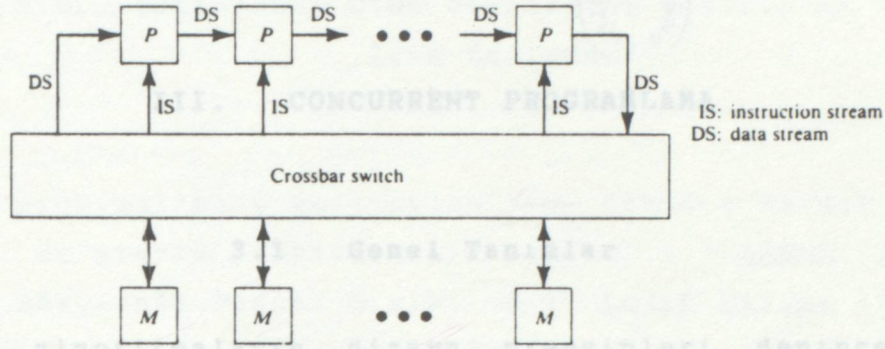
(c) $T(1 \times 16, 1, 16)$ for MISD mode

Figure 1.18 Operation modes in C.mmp system (all double-arrowed paths are for both IS and DS).

Sistem word genişliği 16 olan 16 adet PDP-11 minicomputer'den oluşur. Normal'de MIMD modda çalışır a şıkkındaki gibi. Teorik olarak SIMD modunda da çalışır, b şıkkında bir master tarafından senkronize edilen minicomputerler görülmektedir.

Son olarak sistem MISD modunda çalışması için yeniden organize edilebilir. c şıkkındaki gibi. Bu üç ayrı işlem moduna bağlı olarak C.mmp + operatörü kullanılarak eş gibi ifade edilir.

$$T(\text{C.mmp}) = 16, 1, 16 + 1 \times 16, 1, 16 + 1, 16, 16$$

III. CONCURRENT PROGRAMLAMA

3.1 Genel Tanımlar

Paralel algoritmaların dizayn prensipleri denince akla CONCURRENT PROGRAMLAMA denir. Concurrent programlama paralellik imkanı sağlayan senkronizasyon ve iletişim problemlerinin çözümü için kullanılan bir tekniktir. Paralel algoritmaların ardışıl algoritmalarından çok farklı olması concurrent programlama adı altında yeni bir dalın gelişmesine neden olmuştur. Paralel bir programı anlamak için önce hangi işlemlerin eşzamanlı yapılacağını ve bunların paylaşacakları objeleri belirlemek gerekir.

Paralel programların dizaynında iki tür yaklaşım (iki tür concurrency) vardır. Birincisi EXPLICIT CONCURRENCY dir. Bu metodda programcı kullandığı programlama dili sayesinde paralel işlem görecektir. İkincisi IMPLICIT CONCURRENCY dir. Bu metodda derleyici paralel yürütülecek işlemlere karar verir. Bu ikinci metodda DATA-FLOW bilgisayarlar için daha uygundur. Bu çalışmada yakından incelenecek olan birinci metoddur. Bir MIMD çoklu işlemcili sisteminde problemleri çözerken işlemlerin çalışma zamanları (EXECUTION TIME) birbiri üzerine biniyorsa (OVERLAP) işlemler concurrent tır. Kavram daha iyi şöyle açıklanabilir. Eğer bir işlemin bir operasyonu diğer işlemin bir son operasyonu bitmeden başlar ise bu iki işlem concurrent tır denir. JOIN ifadesinin yürütülmesi

başlatılır. Sonuçta en sonuncusu dışında tüm ifadeler JOIN sonlandırılmasını gerektirir. Bu komutlarla concurrent bir işlemin kontrolü aşağıdaki şekilde verilmektedir.

Bu konu 3.2'de Concurrent Olma Özelliğini Çoklu İşlem geçmede komutun bitmesine İçin Kullanma'dır. FORK ve JOIN özellikle ve dikkatli kullanılmalıdır. Ardışıl yürütülen işlemleri de concurrent olarak yapar. Ardışıl Paralel programlamada kullanılan pek çok dil vardır. Bunlar başlıca üç grupta toplanabilir. Bunlar 1.) ALGOL Tabanlı Diller Ada, Linda, Pascal-S v.b. 3.) Lojik Diller Paralel Lisp, Multilisp ve Concurrent Prolog v.b. 3.) Paralel Fonksiyonlu diller PARAF1'dir. Bu çalışmada Pascal'ın bir alt kümesi kullanılacaktır. İleride anlatılacak paralel algoritma örneklerine geçmeden önce bunların kodlanacağı dilin sentaksını tanıtilmasında yarar vardır. Concurrent olma özelliğini belirtmek için bir yol FORK ve JOIN diye iki komut kullanılır. FORK yeni bir işlem üretir, JOIN daha önce yaratılan işlemin son bulmasını bekler. Genelde FORK komutu üç tipte kullanılır.

FORK A : A adresindeki işleme ilk değer verir
ve o andaki işleme devam eder.

FORK A, J : FORK A ile aynı işlemi yapar ama bu
sırada sayıcıyı J adresine getirir.

FORK A, J, N : FORK A ile aynı işlemi yapar ama bu
sırada sayıcıyı J adresinden N
adresine getirir.

Tüm FORK ifedelerine karşı gelen JOIN ifadesi JOIN J'dir. Bu ifadenin J'de olan sayıcının bir azaltılmasına neden olur. Eğer sonuç 0 ise J+1. adresteki işlem ilk değer alır. Aksi halde JOIN ifadesinin yürütülmesi bırakılır. Sonuçta en sonuncusu dışında tüm ifadeler JOIN sonlandırıcısını yürütürler. Bu komutlarla concurrent üç işlemin kontrolü aşağıdaki şekilde verilmiştir. Her biri bağımsız olarak yürütülür. Her biri bağımsız olarak yürütülür. Her biri bağımsız olarak yürütülür.

Bu komutlarda bir kavşak noktası (Junction Point) den geçmede komutun bitmesine izin verilmemektir. FORK ve JOIN akıllıca ve dikkatli kullanılmazsa ardaşıl yapılması gereken işlemleri de concurrent olarak yapar. Ardaşıl programlar için GOTO ne ifade ediyorsa paralel programlar için FORK ve JOIN de aynı şeyi ifade eder. Özelliklerinden dolayı şartlı ifadelerde ve çevrimlerde paralellik sağlamak için kullanılır. FORK ve JOIN kavramının geliştirilmiş şekli COBEGIN-COEND yapısıdır.

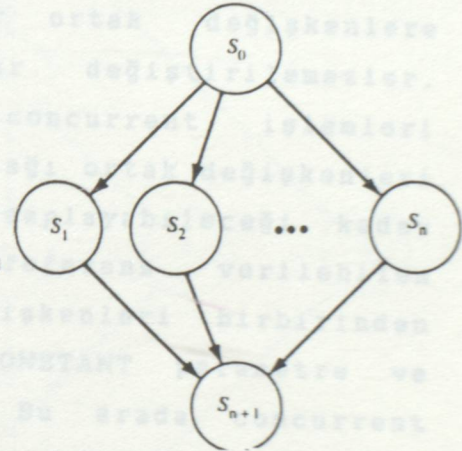
3.3 Concurrent Programlamada Notasyonlar

Aşağıda açıklanan COBEGIN- COEND yapısı Dijkstra tarafından ortaya atılmıştır.

```

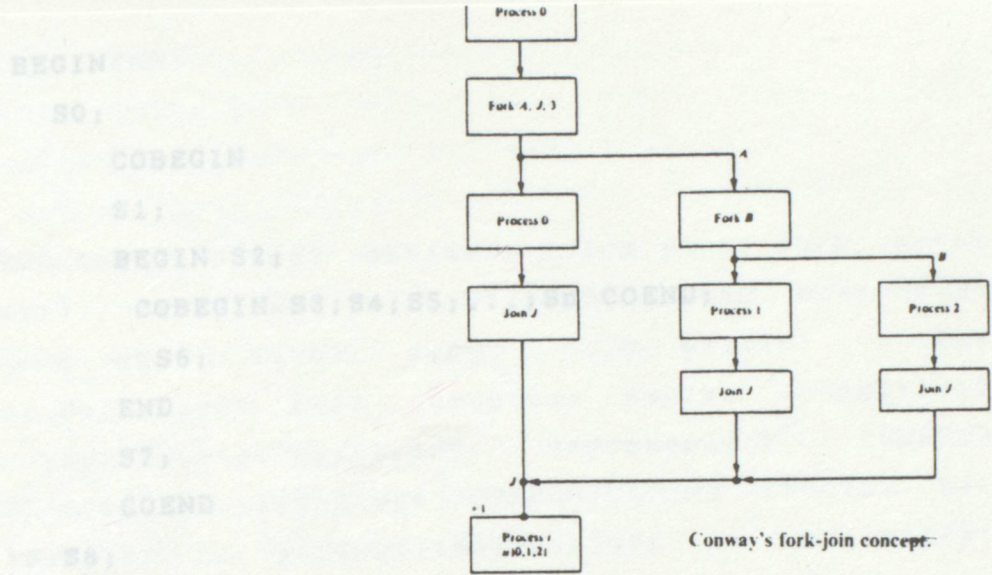
BEGIN
  S0
  COBEGIN S1;S2;S3;...;Sn COEND
  Sn+1
END

```



Precedence graph of the concurrent program.

COBEGIN S1;S2;S3;...;Sn işlemlerinin paralel olarak çalıştırılacağını belirtir. Paralel çalışması istenen işlemler COBEGIN-COEND parantezi içine yazılır. Paralel çalışma ancak S0 bitince başlar. Sn+1 in başlaması için de ancak tüm S1,S2,S3,...,Sn işlemlerinin bitmesine bağlıdır. Concurrent ifadenin bir giriş ve bir çıkışı olduğundan yapısal programlamaya çok uygundur. Bu programda tanımlanan S1,S2,S3,...,Sn ifadeleri birbirlerinden tamamen bağımsızdır. Her biri bağlantısız(disjoint) işlemler olarak yürütülürler. Bağlantısızlık her bir Si ifadesi tarafından



güncelleştirilen bir V_i değişkeninin bir başka S_j ($i \neq j$) ifadesi tarafından referans verilmesinin kesinlikle önlenmesidir. Bir başka deyişle bir işlem tarafından güncelleştirilen bir değişken kesinlikle bu işleme özeldir. Fakat bağlantısız işlemler ortak değişkenlere referans verebilirler ama bunlar değiştirilemezler. Program derleyicinin programdaki concurrent işlemleri gözden geçirerek ve bunların kullanacağı ortak değişkenleri bularak bağlantısızlık derecesini hesaplayabileceği kadar basit olmalıdır. Derleyici sadece referans verilebilen değişkenler ile değiştirilebilen değişkenleri birbirinden ayırabilmelidir. Bunlara sırasıyla CONSTANT parametre ve VARIABLE parametre adı verilir. Bu arada concurrent işlemler içiçe de olabilir. Aşağıda bunun bir örneği verilmiştir.

Başka bir örnek olarak $C=A.B$ şeklinde bir matris çarpımını gözönüne alalım. $A(n \times n)$, $B(n \times n)$, $C(n \times 1)$ boyutlu matrisler olsun. n sayısı çok büyük ise PARALEL FOR komutu ile çarpım yapılır. Bu işlem aşağıdadır.

```
PARFOR i=1 until p DO
```



```

BEGIN
  S0;
  COBEGIN
    S1;
    BEGIN S2;
      COBEGIN S3; S4; S5; ...; Sn COEND;
    S6;
  END
  S7;
COEND
S8;
END

```

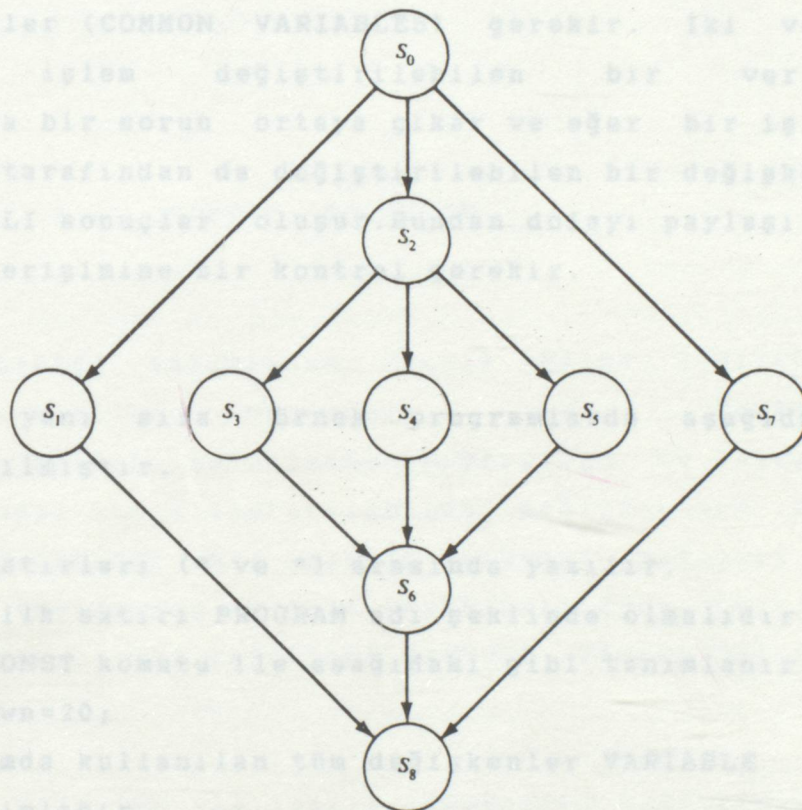


Figure 7.52 Precedence graph of nested concurrent processes.

```

BEGIN
  FOR j=(i-1)*s+1 until s*i DO
    COBEGIN
      PROCEDURE c(j)=0
      FUNCTION FOR k=1 until n DO
        c(j)=c(j)+a(j,k)*b(k);

```


7.3 Programda END ile tanımlanan kontrol yapıları

END lojik-ifade THEN komutlar

IF lojik-ifade THEN komutlar ELSE komutlar

FOR değişken:=ilk-değer TO son-değer DO komutlar

Paralel FOR komutu p adet bağımsız işlem yapar (n/p işlemi tam olmalı). $s=n/p$ ise en dıştaki BEGIN-END arasındaki ifadeler s adet farklı değer için paralel olarak işlenirler. Dolayısıyla c(i) grubunun hesabı concurrent olarak yapılmıştır. Paylaşılan değişkenlere (SHARED VARIABLES) erişilen işlemlere (COMMUNICATING PROCESS) adı verilir. İşlemler paylaşılan kaynakları kullanmayı bitirdikten sonra serviste gerekli işlemin takibi için ortak değişkenler (COMMON VARIABLES) gerekir. İki veya daha fazla işlem değiştirilebilen bir veriyi paylaştıklarında bir sorun ortaya çıkar ve eğer bir işlem diğer işlemler tarafından da değiştirilebilen bir değişkene erişiyorsa HATALI sonuçlar oluşur. Bundan dolayı paylaşılan değişkenelerin erişimine bir kontrol gerekir.

Paralel programlarda tanımlanan Kritik Bölge (CRITICAL

Bu komutların yanı sıra örnek programlarda aşağıdaki komutlar kullanılmıştır.

İşlemler tarafından değiştirilebilen değişkenlere bir

1.) Açıklama satırları (* ve *) arasında yazılır.

3.) Programın ilk satırı PROGRAM adı şeklinde olmalıdır.

3.) Sabitler CONST komutu ile aşağıdaki gibi tanımlanır.

CONST town=20;

4.) Ana programda kullanılan tüm değişkenler VARIABLE olarak tanımlanır

VAR i,j,temper : integer;

found : boolean;

1. KUTU VAR a:ARRAY(.ilk-indis..son-indis.) OF integer;

5.) Procedure ve Fonksiyonlar aşağıdaki gibi tanımlanır

PROCEDURE sort(low,high,integer);

FUNCTION last(index:integer):boolean;

6.) Programın tüm gövdesi Begin_End parantezi içindedir.

7.) Programda kullanılan kontrol yapıları

IF lojik-ifede THEN komutlar

IF lojik-ifede THEN komutlar ELSE komutlar

FOR değişken:=ilk-değer TO son-değer DO komutlar

WHILE lojik-ifede DO komutlar

REPEAT komutlar UNTIL lojik-ifede

REPEAT komutlar FOREVER

Pascal'da paralel program yazmada kullanılan önemli komutlardan bir de REPEAT...FOREVER Komutudur. Bu komut ardaşıl kullanılan REPEAT...UNTIL false komutu gibi çalışır.

3.4 KRİTİK BÖLGELER

Paralel programlarda tanımlanan Kritik Bölge (CRITICAL SECTION) kavramı sayesinde bu kontrol gerçekleştirilir. Kritik bölgelerde paylaşılan ve birçok işlemci tarafından güncelleştirilebilen değişkenlere bir anda bir anda ancak tek bir işlemin erişmesi sağlanır ve diğer işlemler tarafından erişim yasaklanır. Buna paralel veri işlemese KARŞILIKLI YASAKLAMA (MUTUAL EXCLUSION) adı verilir.

Genelde Kritik Bölgeler ile ilgili aşağıdaki tanımlamalar yapılır:

1. MUTUAL EXCLUSION : Bir anda bir Kritik bölgede en fazla bir işlem olabilir.
2. TERMINATION : Kritik bölge sonlu bir zaman dilimi içinde yürütülür.

3. JOB SCHEDULING : Kritik bölgeye girmeye teşebbüs eden bir işlem neticede bunu sonlu bir zaman dilimi içinde yapacaktır.

Bir grup paylaşılan değişkene karşılıklı yasaklamalı erişim (mutually exclusive access) çeşitli yapılar ile gerçekleştirilir. Bunlardan biri MUEXBEGIN_MUEXEND yapıları ile gerçekleştirilir. Her yapı tek başına bir 'v' değişkeninin bir işlemciye özel mi yoksa birçok işlemci tarafından paylaşılmakta mı olduğunu işlemciye belirtmeye yetmez. Bir derleyici paylaşıla bir değişken kullanan işlemleri farketmelidir. Aşağıda verilen notasyon bir grup paylaşılan veya ortak değişken tanımlar.

T : var v: shared T

Kritik bölge Csect v do s olarak tanımlanır. Bu tanımlamada s ifadesinin kullanılacağı, bu s komtunun 'v' değişkenine erişiminin özel (Exclusive) olduğunu belirtir. Kritik bölge de aynı 'v' değişkeni ile ilgilidir ve işlemlerin birini veya diğerini dışarıda tutarı açıkça kritik bölge ile paylaşılan değişkenin pekçok concurrent işlem tarafından kullanıldığını belirtir. Bu da bağılantızılık prensibine açıkça karşı gelmektir. Aynı zamanda derleyici paylaşılan bir değişkenin sadece kritik bölge içinde kullanıldığını kontrol edebilir ve karşılıklı yasaklamayı doğru olarak gerçekleştiren bir kod üretebilir. Aşağıdaki örnekteki gibi farklı değişkenlerle ilgili olan kritik bölgeler paralel işlenebilir. Bunun yanı sıra kritik bölgeler içiçe (nested) de olabilir. Bütün bu olanaklara rağmen genede bütününüyle bir başarısızlık tehlikesi vardır. Bu durum bir veya daha fazla işlemin hiç olmayacak bir işlemi beklemek için bloke edilmeleri durumunda ortaya çıkar. P1 ve P2 concurrent işlemlerinde, paralel programda eğer P1 Section v'ye, P2 Section w'ya, girerse bir kilitlenme söz konusudur.


```

Bir örnek olarak iki işlemci düşünelim. Birinci işlemci
verileri üreterek bunları ikincisine göndermektedir.
İkinci işlemci verileri kullanmaktadır. Verileri
gönderen işlemci için bir değişken var v . shared v;
gönderen işlemci için bir değişken var w . shared w;
kullanılmaktadır. Bu durumda
cobegin
  csect v do P;
  csect w do Q;
coend

```

```

var shared event;
cobegin
  cycle 'sender'
  csect v do
    begin
      signal(event);
      .....
    end
  csect w do S;
  .....
end
coend

cobegin
  P1: csect v do csect w do S1;
  P2: csect w do csect v do S2;
coend

```

3.5 MUTUAL EXCLUSION PROBLEMİ

P1 işlemi w kritik bölgesine girmek istediğinde, henüz P2 işlemi w kritik bölgesinde olduğundan, geciktirilecektir. P2 işlemi v kritik bölgesine girmek istediğinde, henüz P1 işlemi v kritik bölgesinde olduğundan, geciktirilecektir. Bu sonsuz bir döngü oluşturur. Sorun her iki işlemin kendi kritik bölgelerine ters sırada girmek istemelerinden oluşmuştur. Bu kilitlenme olasılık dahilindedir çünkü bir kaynak bir başka kaynağı tutmayı (allocate) bekleyen bir işlem tarafından terkedilemez varsayımı yapılmıştır. Bu kilitlenme (COMMON MEMORY) tipidir. Bu metod sıkı-sıkı paralel

Bir örnek olarak iki işlemci düşünelim. Birincisi bazı verileri üreterek bunları ikincisine göndermektedir. İkincisi ise bunları almakta ve kullanmaktadır. Verilerin gönderilenden daha hızlı alınamayacağı bellidir. Bu düzeni sağlamak için alıcının bazı gecikmeler oluşturması gerekir. İşte senkronizasyonda böyle concurrent işlemler arasında zamanlamayı sağlayan genel bir terimdir. Aşağıda

erişimi kısıtlıdır (access restricted) procedural bir tutulmuşunda da kullanılır. Bu alt programlara erişim bir işlem var e: shared event; etmektedir. Mutual Exclusion Karşılıklı yasaklama anlamında gelir ve concurrent programların cycle'u 'sender' tarafından biridir. Çünkü pek çok senkronizasyon beginlemesi başlangıcıdır.

P1 işlem..... signal(e);..... P2 işleminin A2 aktivitesi ve end. A1 in işlemi A2 nin işlemesinin üstüne binmeye cycle 'receiver' başlamalıdır. Eğer P1 ve P2 eşzamanlı olarak beginleri aktivitetlerini A1 ve A' yi işletmeye kalkarsa bunlar..... wait(e);..... classını garantilenmiş gerekir. Kaybeden de diğer işlem ise bloke edilir ve işi başlatır coend en kendi A Aktivitesini bitirdikten çalışmaya başlanmalıdır. Karşılıklı yasaklamanın en iyi örneklerinden biri gerçek sistemlerdeki bu düzeni sağlayan program verilmiştir. paylaşımdır. Ortada eşitlik bir çözüm vardır. Bir zamanda sadece bir iş(job) çalışmalıdır. Bu ise concurrent programlamanın PARALLELİK PRENSİBİ'ni

3.5 MUTUAL EXCLUSION PROBLEMİ

(Karşılıklı Yasaklama) Karşılıklı yasaklama ve zaman zaman ortaya çıkan iletişim ile

Daha önce concurrent programlama tanımında ardışıl programlamada ardışıl olarak yapılan işlemlerin paralel algoritmalarda birbirinden ayrıldığını belirtmiştik. Paralel programlardaki işlemler tamamen bağımsız değildirler. Zaten böyle olası tartışılacak bir konu olmazdı. Bağımsız olan birimler birbirleriyle senkronizasyon ve veri değişimi ile iletişim kurmalıdırlar. Tartışılacak ilk iletişim tipi Ortak Bellek (COMMON MEMORY) tipidir. Bu metod sözde-paralel

(pseudo parallel) bilgisayarlarda (cdc Cyber 1 CPU ve 10 Peripheral Processor PP) da kullanılır. Bir CPU ve 10 çevre işlemci aynı zamanda ayrı ayrı programları yürütürler. Senkronizasyon PP'lerin CPU belleğini okuma/yazma işlemi kontrolü ile sağlanır. Ortal bellek bu çalışma içinde basitçe tüm işlemcilerin erişebileceği global değişkenlerle belirtilmiştir. Bu bellek aynı zamanda erişimi kısıtlayıcı (access-restricted) procedurelerin tutulmasında da kullanılır. Bu alt programlara erişim bir işlemciyi tutmak(allocate) etmektir. Mutual Exclusion Karşılıklı yasaklama anlamına gelir ve concurrent programların en büyük sorunlarından biridir. Çünkü pek çok senkronizasyon probleminin başlangıcıdır.

P1 işleminin A1 aktivitesi ve P2 işleminin A2 aktivitesi verilsin. A1 in işlemesi A2 nin işlemesinin üstüne binmiyorsa birbirinden ayrılmalıdır. Eğer P1 ve P2 eşzamanlı olarak kendi aktivitelerini A1 ve A' yi işletmeye kalkarsa bunlardan birinin başarılı olmasını garantilememiz gerekir. Kaybedilen diğer işlem ise bloke edilme ve işe başlayan işlem kendi A Aktivitesini bitirmeden çalışmaya başlamamalıdır. Karşılıklı yasaklamanın en iyi örneklerinden biri gerçek sistemlerdeki kaynakların(teyp, printer, disk v.s) paylaşımıdır. Ortada aşikar bir çözüm vardır. Bir zamanda sadece bir iş(job) çalışmalıdır. Bu ise concurrent programlamanın PARALELLİK PRENSİBİ'ni yok eder. Anlamli concurrency eğer işlemciler gevşek bağlı ise mümkündür. Bu gevşek bağlantı kendini kısa ve zaman zaman ortaya çıkan komunikasyon ile gösterir. Özetle karşılıklı ayırma problemi aşağıdaki kavramlarla ifade edilir.

1. REMAINDER

2. PRE-PROTOCOL

3. KRİTİK BÖLGE(CRITICAL SECTION)

4. POST-PROTOCOL

REMAINDER'ın belirli işlemleri gerçekleştirdiği kabul edilecektir. İşlem zaman zaman remainder'ın tamamlanmasından sonra kısa bir Kritik Bölge'ye girmeye ihtiyaç duyar. Bu Kritik Bölge'den önce işlem belli komut kümelerinden oluşan emirleri yerine getirecektir ki bunlara protokoller denir. Protokollerin görevi bir kritik bölgenin bittiği, işlendiği ve böylece tüm diğer kritik bölgelerin işlenebileceğine güvence vermektedir. Concurrency den yararlanmak için kritik bölgeler ana programa göre kısa olmalıdır. Protokoller concurrency için başa çıkılamayan sorunları çözümlerler. Protokoller de kısa olmalıdır. Concurrent işlemler arasındaki bağlantının gevşek olmasının bir başka nedeni ise işlemlerde meydana gelebilecek hatalar (Bug'lar) dır. İşlemlerin birinde oluşan bir hatanın tüm sistemi bloke etmesi istenmez. Bu durumda problemlili işlem sistemden izole edilir. Eğer işlem kritik bölge veya protokol dışında anormal olarak son bulursa diğer işlemler de etkilenmeyecektir. Bu amaçla protokoller kritik bölgenin bir kısmı olarak kabul edilirler. Kritik bölge işlemler arasında asıl iletişimin gerçekleştirildiği yerdir.

3.4 Problemin Tanımlanması

Burada P1 ve P2 olarak adlandırılmış iki işlem için karşılıklı yasaklama problemi için bir çözüm araştırılacaktır. P1 ve P2 işlemleri bir programda sonsuz bir çevrim içinde çalışmaktadırlar. Programda iki bölge vardır. Kritik Bölgeler 'crit1' ve 'crit2' olarak Remainder'lar 'rem1' ve 'rem2' olarak tanımlanmıştır.

İLK ADIM

Bir eskimo klubesi ve bir protokol düşünelim. Iglooda bir karatahta olsun. Giriş tüneli ve iglooo o kadar küçüktür



ki verilen bir zamanda içinde ancak bir eskimo olabilir. Tahtada da kritik bölgeye girme sırası kimin ise onun numarası yazılı olur. Iglooonun küçüklüğü mecazi olarak bellek hakemine karşılık gelir. Kritik bölgeye girmek isteyen işlem iglooya girer, tahtaya bakar eğer tahtada kendi numarası yazılı ise rahatca işlemlerini yapmak üzere kritik bölgeye gider. Eğer tahtada diğer işlemin numarası yazılı ise iglooyu terkeder ve dışarıda diğer işlemin ismini bitirmesini bekler. İdeal olarak, şanssız işlem bu sırada iglooyu devamlı kontrol etmek zorundadır çünkü bir programlama dili ile bekleme esnasında iken işlem sırasının kendine geldiğinin belirtilmesine imkan yoktur. Bu da hiç bir iş yapmamak demektir. Peryodik olarak iglooonun içine girmeli ve tahtayı kontrol etmelidir. Buna BUSY WAIT (MEŞGUL BEKLEME) denir. Bu ismin verilmesinin nedeni bir iş yapılmamasına rağmen bekleme sırasında enerjinin amaçsız bir iş için harcanıyor olmasıdır. Bir işlem gittiği kritik bölgede yaptığı iş bitince geri döner ve iglooya girerek tahtaya diğer işlemin numarasını yazar. Iglooda bir anda sadece ve sadece bir işlem olacağı için tahta ya 1 yada 0 değerini alır. Yukarıdaki program bu adımların gerçekleştirilmesini göstermektedir. Bu çözüm Karşılıklı Yasaklama'nın gereklerini yerine getirmektedir. Bir "Pi" işlemi sadece TURN değişkeni (TURN=i) değerini alırsa kendi kritik bölgesine girmekte ve bir anda ancak tek bir işlem kritik bölgede olmaktadır. Kritik bölgede olan program işini bitirip TURN değişkenini değiştirinceye kadar aynı değerinde kaldığı için diğer işlem kritik bölgeye giremez. Bir anlaşmazlık sözkonusu değildir. Çünkü Turn sırayla hem 1 hem

tahtaya 2 yazacak ve P2 kritik bölgeye girerken P1'in
varındayken beklerken sorunda kalmaktadır.

```

program firstattempt;
var    turn: integer;
procedure p1;
begin
  repeat
    while turn=2 do (* nothing *);
  crit1;
  turn := 2;
  rem1
  forever
end;
procedure p2;
begin
  repeat
    while turn=1 do (* nothing *);
  crit2;
  turn := 1;
  rem2
  forever
end;
begin (* main program *)
  turn := 1;
  cobegin
    p1; p2
  coend
end

```

de 2 değerini almakta ve bir anda bir işlem yürütülmektedir. İşlemin aynı anda tereddüte düşmesi imkansızdır.

Benzer olarak eğer her programın her bir ifadesi sonlu bir zaman alıyorsa., işlemlerden bir önceki kritik bölgedeki işini mutlaka bitirecek ve diğeri buraya girebilecektir. Yukarıdaki düzen doğru bir concurrent programlama sağlamakla beraber paralel algoritmaların tasarım gereksinimlerinden birini yerine getiremez. İşlemler birbirine gevşekçe bağlanmamıştır. Kritik bölgeye girmek için şart bir işlemten diğeri geçmektir. dezavantaja sahiptir. Eğer P1' in günde 100 defa kritik bölgeye girmesi gerekiyor, buna karşı P2' in günde 1 defa kritik bölgeye girmesi gerekiyorsa P1, P2 nin bir günlük işini yapmaya zorlanacaktır. P1 kendi kritik bölgesini bitirdiğinde

tahtaya 2 yazacak ve P2 kritik bölgeye girmeye karar verinceye kadar beklemek zorunda kalacaktır.

İkinci dezanantaj daha ciddidir. P1 işleminin P2' nin kritik bölgeden çıkmasını ve tahtadaki değeri 2'den 1'e değiştirilmesini beklediğini düşünelim. Eğer P2 iglooya dönerken bir kutup ayısı tafarından (herhangi bir engel) engellenirse P2 sonbulmakla kalmayacak p1 de umutsuzca tahtanın P2 tarafından değiştirilmesini bekleyecek ve durum kilitlenecektir. Eğer P2 kendi rem2 sinde yani kritik bölgesi dışında işini bitirirse durum doğrudur. Bu da bir işlem işini kendi kritik m bölgesi dışında bitirebilir fikrini doğrular.

Kontrolün böyle açıkça geçirilmesi tekniğine COROUTINE'leradı verilir. RESUME(P2) şeklinde bir komut yürüterek P1 kendi işleminin P2 için askıya alındığını belirtir. P2 RESUME(P1) komut yürütene kadar işine devam eder. Coroutine'ler yararlı bir programlama tekniği olmakla beraber Coroutine'ler sistemi tek bir entegre sistem olarak tasarlanmamalıdır ve bu concurrent programlama yerine geçemez.

```

program secondattempt;
var   c1, c2: integer;
procedure p1;
begin
  repeat
    while c2=0 do;
      c1 := 0;
      crit1;
      c1 := 1;
      rem1
    forever
  end;
  procedure p2;
  begin
    repeat
      while c1=0 do;
        c2 := 0;
        crit2;
        c2 := 1;
        rem2
      forever
    end;
  begin (* main program *)
    c1 := 1;
    c2 := 1;
    cobegin
      p1; p2
    coend
  end
end

```


İKİNCİ ADIM

Bir önceki adımdaki ikinci dezavantaja bu adımda çözüm bulunmuştur. Burada her işlemin kritik bölgeye gitmek için kendi anahtarı vardır. Bir tanesi herhangi bir dış engel tarafından engellense bile diğer işlem kendi anahtarını kullanarak kritik bölgeye girebilecektir. Burada bir işlemle birlikte bir global değişken (Igloo) tanımlanmıştır. Birinci adımda TURN değişkeni üzerinde OKUMA/YAZMA işlemi yapılmakta idi. Burada ise çözümün canlandırılması daha kolaydır. Çünkü her işlem diğer işlem ile tanımlanmış değişkeni okur ama buna değer yazamaz. Eğer P1 i kritik bölgeye girmek istiyor ise P2 igloosuna periyodik olarak girer ve P tahtaya bakar. Bu tahta yani C2=1 oluncaya kadar buna devam eder. C2=1 demek P2 o anda kritik bölgede değil demektir. Bu gerçeği bulunca P1 kendi kritik bölgesini zamanında kendi tahtasına (C1) 0 yazarak girebilir. P1 işini bitirince C1 değerini 1 yapar böylece P2'ye kritik bölgenin

ÜÇÜNCÜ ADIM

İkinci adımın aksayan yönlerini analiz ederek üçüncü

adımda bu sorunlara çözüm getirilecektir. P1 bir kez

aragatırlarsa P2'nin kendi kritik bölgesinde

olmadığı, P2'nin doğru kritik bölgeye sokma görevini

üstlendiği görürüz. Burada P1 WHILE ifadesiyle

geçmişte olduğu gibi P2'nin kendi bölgesinde

yaşamadığını kontrol eder. Eğer P2'nin kendi bölgesinde

yaşamadığını görürse P1 kendi bölgesine girer. Burada

WHILE ifadesiyle P1 kendi bölgesine girer. Burada

WHILE ifadesiyle P1 kendi bölgesine girer. Burada

WHILE ifadesiyle P1 kendi bölgesine girer. Burada

WHILE ifadesiyle P1 kendi bölgesine girer. Burada

WHILE ifadesiyle P1 kendi bölgesine girer. Burada

WHILE ifadesiyle P1 kendi bölgesine girer. Burada

WHILE ifadesiyle P1 kendi bölgesine girer. Burada

WHILE ifadesiyle P1 kendi bölgesine girer. Burada

boş olduğunu anlatmış olur. Yukarıdaki programda



karşılıklı yasaklamaya güvenilir getirilmekle beraber aşağıdaki değerlere dikkat edilirse P1 ve P2 nin aynı anda (eş zamanlı) kritik bölgede olduğu ve dolayısıyla programın

<u>DEĞERLER</u>	<u>C1</u>	<u>C2</u>
İlk Değer	1	1
P1 C2 yi test eder	1	1
P2 C1 i test eder	1	1
P1 C1 i set eder	1	1
P2 C2 yi set eder	0	1
P1 crit1'e gider	0	0
P2 crit2'ye gider	0	0

hala yanlış olduğu görülür.

ÜÇÜNCÜ ADIM

İkinci adımın aksayan yönlerini analiz ederek üçüncü adımda bu sorunlara çözüm getirilecektir. P1 bir kez araştırılırsa P2'nin kendi kritik bölgesinde olmadığı, P2'nin aonu doğru kritik bölgeye sokma görevini üstlendiğini görürüz. Bunun sonunda P1 WHILE ifadesini geçtiği anda kendi kritik bölgesindedir düşüncesini yalanlar. Çünkü WHILE ifadesi ile atama ifadesi arasında rastgele uygun bir bekleme süresi olabilir. Aşağıda görülen program atama ifadesini geliştirerek bu sorunu düzeltmiştir. Böylece C1=0 önce C''yi kontrol etse bile P1'in kendi kritik bölgesinde olduğunu belirtir. Sonuçta WHILE ifadesi başarı ile geçildiği anda P1 kendi kritik bölgesindedir. Yukarıda bu durum için verilen program görülmektedir. Fakat bir şanssızlık bu program kolayca aşağıda belirtildiği gibi çözülemeyen anlaşmazlıklara neden olabilir. Değişkenlerin sürekli kontrol edilmesi dışarıdan farkedilmeden devamlı olabilir ve bir gelişme elde edilmeden program ümitsizce çalışır. Bu program

3.4 THIRD ATTEMPT

```

program thirdattempt;
var   c1, c2: integer;
procedure p1;
begin
  repeat
    c1 := 0;
    while c2=0 do
      crit1;
      c1 := 1;
      rem1
    forever
  end;
procedure p2;
begin
  repeat
    c2 := 0;
    while c1=0 do
      crit2;
      c2 := 1;
      rem2
    forever
  end;
begin (* main program *)
  c1 := 1;
  c2 := 1;
  cobegin
    p1; p2
  coend
end.

```

DEĞERLERC1C2

(ADIM)

İlk Değer

1

1

P1 C1 i set eder

0

1

P2 C2 yi set eder

0

0

P1 C2 yi test eder

0

0

P2 C1 i test eder

0

0

çözülemeyen anlaşmazlıklardan dolayı kabul edilemez

olmasına rağmen yinede KARŞILIKLI YASAKLAM PROBLEMİ' nin gerçeklerini göstermek açısından öğreticidir.

3.5 FOURTH ATTEMPT

```

program fourthattempt;
var c1, c2: integer;
procedure p1;
begin
  repeat
    c1 := 0;
    while c2 = 0 do
      begin
        c1 := 1;
        (* do nothing for a few moments *)
        c1 := 0;
      end;
    crit1;
    c1 := 1;
    rem1
  forever
end;
procedure p2;
begin
  repeat
    c2 := 0;
    while c1 = 0 do
      begin
        c2 := 1;
        (* do nothing for a few moments *)
        c2 := 0;
      end;
    crit2;
    c2 := 1;
    rem2
  forever
end;
begin (* main program *)
  c1 := 1;
  c2 := 1;
  cobegin
    p1; p2
  coend
end.

```

DÖRDÜNCÜ ADIM

Bir önceki örnekte P₁ C₁'i 0 yaptığında kritik bölgeye girmek istediğini belirtir ve aynı zamanda bunun kendi hakkı olduğu konusunda ısrar eder. C₁'i C₂'yi test etmeden set etmek Karşılıklı Yasaklama'nın bozulmasını önler fakat eğer P₂ birşeyler üretmeye hazır değilse bu durumda P₁'in işlem yapması sağlanmalıdır. Aşağıda görülen programda bu inatçı davranış düzeltilmiştir.

Programda bir işlem geçic olarak kritik bölgeye girme niyetinden vazgeçer ve diğer işleme kritik bölgeye girmesi için bir şans tanır. P1 iglooya girer ve tahtaya 0 yazar. Eğer P1 P2'nin igloosunu kontrol sonuna 0 bulursa kahramanca kendi igloosuna 0'ı silmek üzere geri döner. Igloonun çevresinde biraz gezindikten sonra C1=0 yapar ve tekrar dener. Burada sadece değişken bölümlenmeye (interleaving) izin verildiği için aynı değişkene yapılan aynı iki atamanın aynı anlamda olmadığına dikkat etmek gerekir. Eğer her iki işlem de işlemeye devam eder ve hiçbirisi kritik bölgeye girmezse durum aşağıdaki gibi olur.

<u>DEĞERLER</u>	<u>C1</u>	<u>C2</u>
İlk Değer	1	1
P1 C1 i set eder	0	1
P2 C2 yi set eder	0	0
P1 C2 yi test eder	0	0
P2 C1 i test eder	0	0
P1 C1 i set eder	1	0
P2 C2 yi set eder	1	1
P1 C1 i set eder	0	1
P2 C2 yi set eder	0	0

Burada ana hedef eğer işlemlerin ikisi de hiçbir zaman bir kritik bölgeye girmezse (Sanki belirsiz mükemmel bir senkronizasyon varmış gibi) çevrimlerin iterasyonlarına bir değer veremediğimizden böyle bir sistemin performansı garanti edilemez.

DEKKER ALGORİTMASI

Dekker algoritması birinci ve dördüncü adımların

3.6 DEKKER'S ALGORITHM

```

program Dekker;
var
    turn: integer;
    c1, c2: integer;
procedure p1;
begin
    repeat
        c1 := 0;
        while c2 = 0 do
            if turn = 2 then
                begin
                    c1 := 1;
                    while turn = 2 do;
                        c1 := 0
                    end;
                end;
            crit1;
            turn := 2;
            c1 := 1;
            rem1
        forever
    end;
procedure p2;
begin
    repeat
        c2 := 0;
        while c1 = 0 do
            if turn = 1 then
                begin
                    c2 := 1;
                    while turn = 1 do;
                        c2 := 0
                    end;
                end;
            crit2;
            turn := 1;
            c2 := 1;
            rem2
        forever
    end;
end;

```

begin (* main program *)

c₁ := 1;

c₂ := 1;

turn := 1;

cobegin

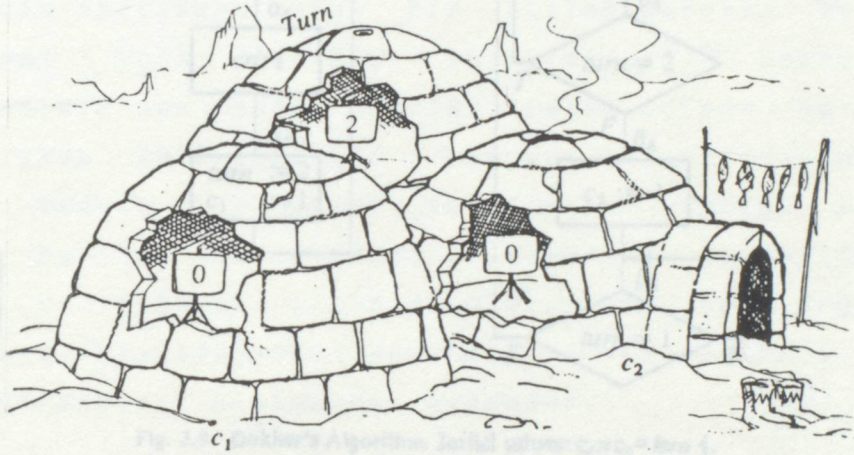
p₁; p₂

coend

end.

akıllıca P₁ oluşturulmuş bir kombinasyondur. İlk çözüm hatırlanacak olursa kritik bölgeye girme hakkını açıkça işlemler arasında geçirdiğimizi görürüz. Fakat şanssızlık eseri kritik bölge için kullanılan anahtar eğer işlemlerden biri karışon bulursa kurtarılamayacak gibi şekilde kaybolabilir. Dördüncü adımdaki çözümde farklı anahtarlar

kullanmanın bir işlemden diğerine sonsuz bir gecikme olasılığı yaratacağı bulunmuştu. Aşağıda görülen Dekker'ın Algoritması bir önceki çözüme dayanır fakat bu çözümdeki kilitlenmeyi kritik çözüme girmeye ısrar etme hakkını açık olarak geçirerek çözer. Her işlemin ayrı bir ıgloosu vardır. Bu nedenle bir işlem kötü bir dış koşul tarafından sonlandırılrsa bile diğeri işlemeye devam eder. (Hiç bir kritik işlemin kendi kritik bölgesinde ve aynı zamanda



protokolde sonlanmadığı kabul edilmiştir. Şimdi bir HAKEM IGLOO vardır. Bu hakem ıgloo nun içinde TURN adı verilen bir tahta ile örneklenebilen bir değişken bulunur. Eğer $P1 \ C1=0$ yapar ve sonra $P2'$ nin de bir 0 yazdığını görürse danışmak üzere hakeme gider. Eğer hakemin yahtasında 1 yazıyorsa $P1$ ısrar etme hakkının kendinde olduğunu anlar ve $P1$, periyodik olarak $P2'$ nin ıgloosunu kontrol eder. Tabi ki $P2$ de erteleme ve gecikme sırasının kendinde olduğunu görerek kahramanca $C2=1$ yapar. Zaten daha önce $c2$ $P1$ tarafından dikkate alınmıştır. Bu sırada $P2$, $P1$ in işini bitirmesini bekler. $P1$ işini bitirmesi üzerine sadece kritik bölgeyi serbest bırakır bırakmaz aynı zamanda $TURN=2$ yapar. Böylece hem $P2$ yi iç çevrimden çıkarır hem de ısrar etme hakkını $P2$ ye devreder.

Karşılıklı yasaklama üçüncü adımdaki gibi aynen sağlanır çünkü $TURN$ değişkenin kritik bölgeye girmeye karar

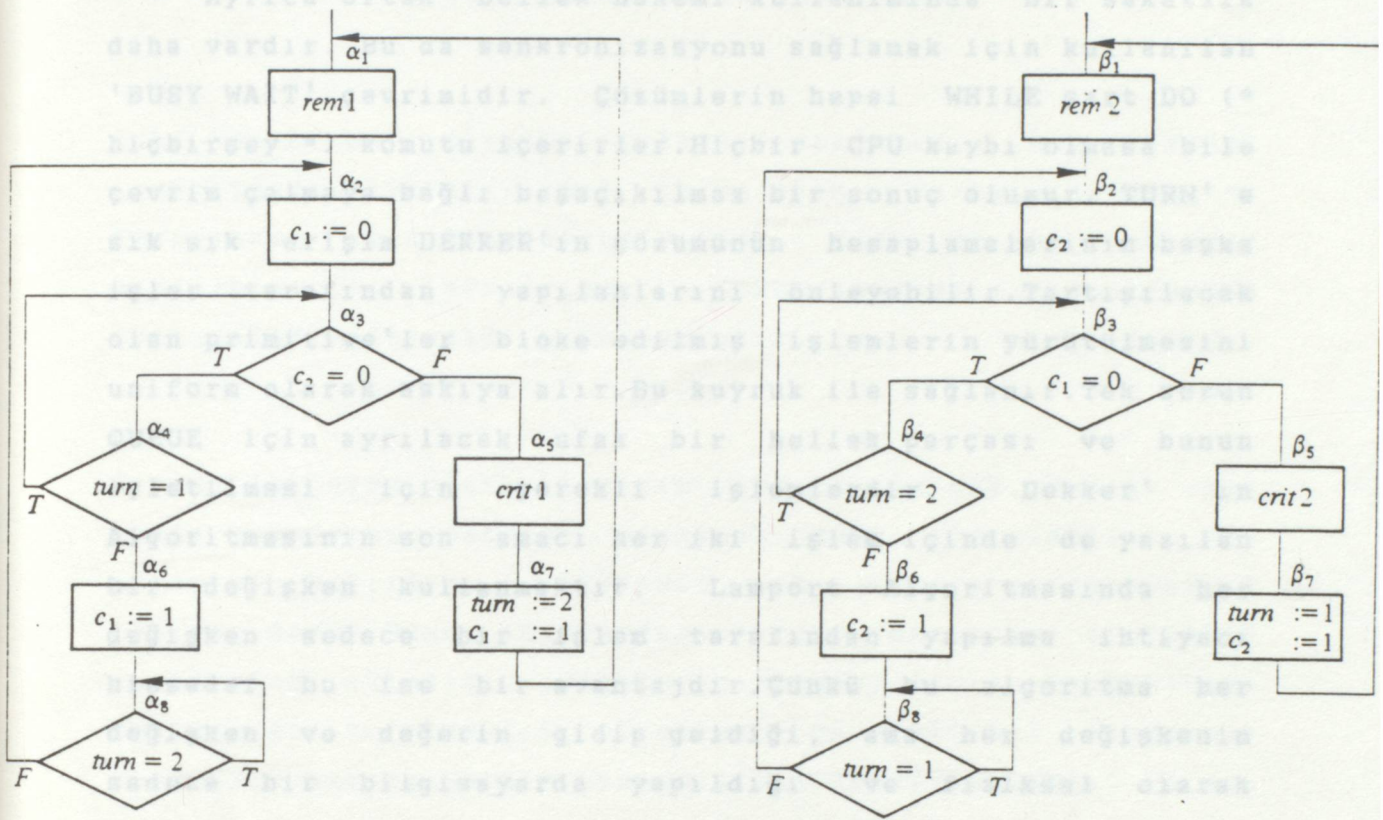


Fig. 3.9 Dekker's Algorithm. Initial values: $c_1 = c_2 = \text{turn} = 1$.

vermede etkisi yoktur.

SONUC

Concurrent programlamada Karşılıklı Yasaklama (MUTUAL EXCLUSION) problemi basit problemlerdendir. Doğru bir çözüm elde ederken ana zorluk bu derece kolay bir problemin bile ortak bellek hakeminden daha güçlü programlama imkanlarına ihtiyaç duymasıdır. Özellikle 'n' adet işlem için MUTUAL EXCLUSION Problemi öyle zordurki sadece akademik bir çalışma için düşünülür.

Ayrıca ortak bellek hakemi kullanımında bir sakatlık daha vardır. Bu da senkronizasyonu sağlamak için kullanılan 'BUSY WAIT' çevrimidir. Çözümlerin hepsi WHILE şart DO (* hiçbirşey *) komutu içerirler. Hiçbir CPU kaybı olmasa bile çevrim çalmaya bağlı başa çakılmaz bir sonuç oluşur. TURN' e sık sık erişim DEKKER'in çözümünün hesaplamalarının başka işler tarafından yapılanlarını önleyebilir. Tartışılacak olan primitive'ler bloke edilmiş işlemlerin yürütülmesini uniform olarak askıya alır. Bu kuyruk ile sağlanır. Tek sorun QUEUE için ayrılacak ufak bir bellek parçası ve bunun işletilmesi için gerekli işlemlerdir. Dekker'ın Algoritmasının son amacı her iki işlem içinde de yazılan bir değişken kullanmaktır. Lamport Algoritmasında her değişken sadece bir işlem tarafından yapılma ihtiyacı hisseder bu ise bir avantajdır. Çünkü bu algoritma her değişken ve değerın gidip geldiği, ama her değişkenin sadece bir bilgisayarda yapıldığı ve fiziksel olarak saklandığı dağıtık sistemlere uygundur.

MUTUAL EXCLUSION

is a solution to the mutual exclusion problem using semaphores.

```

program mutualexclusion;
var      s: (* binary *) semaphore;
procedure p1;
begin
  repeat
    wait(s);
    crit1;
    signal(s);
  rem1
  forever
end;
procedure p2;
begin
  repeat
    wait(s);
    crit2;
    signal(s);
  rem2
  forever
end;
begin (* main program *)
  s := 1;
  cobegin
    p1; p2
  coend
end.

```


3.6 SENKRONİZASYON METODLARI (Message Passing)

Daha önce bu çalışma içinde, sık sık, bir paralel algorithmada ayrıştırılmış işlemlerin farklı işlemciler tarafından yürütölmelerine rağmen, genelde tam bağımsız olmadıkları, aralarında haberleşerek senkronize çalışmaları gerektiği açıklanmıştır. Durum senkronizasyonu

(Condition Synchronization) dur.

İşlemciler arasındaki iletişim paylaşılan değişkenler veya mesaj gönderme (message passing) yolu ile olabilir. Bu mimariye bağlıdır. İletişim sık sık senkronizasyona ihtiyaç duyar. Bunun iki faydası vardır. Birincisi olayların sıralanması ikincisi işlemlerin müdahalesinin kontrolünün sağlanmasıdır. Örneğin pipeline paralel bir algoritma düşönelim bir işlem bir tampon alana değer yazmakta, ikinci işlemci ise gidip oradan okumaktadır. Aşıkarak bellidir ki birinci işlem tampon bölge dolu ise buradaki değeri ezerek yazma yapamaz. İkinci işlem ise bu bölgeye bir değeri konmadan okuma yapamaz. Bu olayların olmasını engelleyen işte bu tip, birinci tip bir senkronizasyondur.

3.1-Busy-Waiting

İkinci tip senkronizasyon için n adet tam sayısının paralel toplanmasını düşönelim. N adet işlemci ortak bir toplam değışkenine kendi sayısını eklemeyi isteyecektir. Bu tip senkronizasyon bir işlemci sayısını eklerken diğeri işlemcinin müdahalesini engeller. Bu sırada LOCK kavramı kullanılır.

Burada 2 senkronizasyon metodu incelenecektir. göre çalışan bir bölge olsun. S'a giris bir değışken (gate) ile kontrol

A-) Paylaşılan Değişkenler ile Senkronizasyon için gerekli kontrolölen geçmek zorundadır. Bu bölgeye giris protoköle

B-) Mesaj Yollama ile Senkronizasyon (Message Passing) olarak tanımlanır. Gerektiği zaman bunlar işlemcinin kritik bölgedeki komutları yürütmesini engelleyebilirler. Buna göre A-) paylaşılan Değişkenler ile Senkronizasyon bu protokollerin etkisini bastırabilir.

Paylaşılan değişkenler ile yapılan senkronizasyonun iki kullanım alanından biri karşılıklı yasaklama (Mutual Exclusion) olayı olduğu ise Durum senkronizasyonu (Condition Synchronization) dur.

Örneğin, yani daha önce bu bölgeye bir başka işlemci girmiş ve burada çıkmamış ise,

Karşılıklı Yasaklama olayı başladığında, fiziksel veya görüntü bir kaynağın bölünemez, paylaşılamaz duruma gelmesi garantilenmiş olur. Bir başka durum ise şöyle oluşur. Birbirleriyle yardımlaşarak çalışan bir kaç işlemci paylaşılan bir değişkeni, uygun olmayan bir zamanda kullanmaya kalkıştırlarsa, devam eden yürütme, değişken uygun duruma gelinceye kadar geciktirilir. Buna 'Durum Senkronizasyonu' denir.

İşte bu tekniğe (Busy Waiting) Meşgul Bekleme denir. Bu işleme LOCK ve UNLOCK komutları ile gerçekleştirilir.

Paylaşılan değişkenlerle senkronizasyon 3 metodla yapılır.

a.1-) Busy-waiting

a.2-) Semaphore'lar

a.3-) Monitor'lar

a.1-) Senkronizasyonda Busy-Waiting Metodu ile kullanılan komutlar

3.5.2'de Kritik bölgelerde LOCK ADI'nda detaylı olarak anlatılmıştır. gate=0 ise Kritik bölge kullanılabılır, gate=1 ise meşgul demektir.

S kritik bölgesi, karşılıklı yasaklama esasına göre çalışan bir bölge olsun. S'e giriş bir değişken (gate) ile kontrol edilir. Bu bölgeye girmek isteyen bir işlemci gerekli kontrolden geçmek zorundadır. Bu bölgeye giriş protokolü

MUEXBEGIN (gate), çıkış protokülü ise MUEXEND (gate) olarak tanımlanır. Gerektiği zaman bunlar işlemcinin kritik bölgedeki komutları yürütmesini engelleyebilirler. Buna ters olarak kritik bölgedeki 'Csect gate do S' ifadesi bu protokollerin etkisini bastırabilir.

MUEXBEGIN ve MUEXEND yapılarının gerçekleşmesinde ciddi problemlerle karşılaşılır. MUEXBEGIN komutu kritik bölgenin o anki durumuna bakar; eğer meşgul ise, yani daha önce bu bölgeye bir başka işlemci girmiş ve henüz çıkmamış ise, işlemci bölgeye giremez ve beklemeye başlar. Bu bir belirtece kritik bölgenin dolu olduğunu gösteren bir değer atayarak yapılır. Kritik bölgede olan işlemci bölgeden çıkarken MUEXEND komutunu yürüterek, kritik bölgeyi serbest bıraktığını belirtmeli ve belirtecin bu değerini düzeltmelidir. Böylece kritik bölge (CS) tekrar kullanıma açılır. Dışardan bölgeye girmeyi isteyen işlemci devamlı olarak, bu belirteci ard arda test edecek ve bekleyecektir. İşte bu tekniğe (Busy Waiting) Meşgul Bekleme denir. Bu işlem LOCK ve UNLOCK komutları ile gerçekleştirilebilir.

LOCK ve UNLOCK KOMUTU:

LOCK (gate)

Kritik bölge işlemlerini yap

UNLOCK (gate)

Bu olay 3.5.2'de kritik bölgelerde İLK ADIM'da detaylı olarak anlatılmıştır. gate=0 ise kritik bölge kullanılabilir, gate=1 ise meşgul demektir.

LOCK(X) işlemi aşağıdaki gibi gerçekleştirilir.


```

var X: shared integer;

LOCK(X): begin
    var y: integer;
    y=x;
    while y=1 do y=x; (* kapı açılana kadar bekle *)
    x=1;               (* bölgeyi meşgul yap *)
end;

var x: shared integer;
LOCK(X): begin

```

UNLOCK(X) işlemi ise şu işlemleri yapar.

```

repeat(y=TEST-AND-SET(X)) until y=0;

UNLOCK(X) : X=0      (* kapıyı aç *)

```

Kritik bölge konusunda da anlatıldığı gibi bu metodun sakıncaları vardır. Senkronizasyon için daha güvenilir bir metod bulmak gerekir.

kullanılır. paylaşılan bir değişken üzerinde, işlem yapan bir başka komut **COMPARE-AND-SWAP** (karşılaştır ve yer değiştir) Bu komut **a_addr** adresindeki **r_old** ve **r_new** değişkenleri ile yapılır. **r_old** ve **r_new** işlemleri saklayıcılarıdır.

TEST-AND-SET KOMUTU:

İşlemci kritik bölgeye girmek istediğinde tek bir komutla kontrol değişkenini test eder ve hemen ardından bölgenin dolu olduğuna dair değişkeni değiştirir (set) ise ve bu iki işlem bir bütün ve araya hiçbir işlemin ve kesmenin girmesine izin verilmeden yapılırsa problem çözülür. Read, Modify, Write işlemleri, bu durumda, tek bir bellek çevriminde yapılmaktadır. Bunu gerçekleştiren TEST-AND-SET komutudur ve aşağıdaki gibi kullanılır.

```

(* a_addr: r_new; x=1 *)
(* else *)
(* r_old: r_new; x=0 *)

```

```

var x: shared integer;
TEST-AND-SET(X) : begin

```


Kritik bölge: `var y: integer;` bu bölgeye girişi engellemek için `primitive y=x;` tanımlanabilir. Bunlar `wake-up(p)` ve `block(p)` `if y=0 then x=1` `end.` `TEST-AND-SET(X)` komutuna uygun olarak `LOCK(X)` komutunu da yeniden yazmak gerekir.

`WAKE-UP(P)`: Eğer P işlemci mantıksal olarak bloke edilmiş ise, durumunu aktif yapar. Aksi halde `WWS (Wake-up Waiting Switch)` `var x: shared integer;` ini artırılmak için set eder.

`LOCK(X) : begin`
`var y: integer;`
`repeat (y=TEST-AND-SET(X)) until y=0;` ise reset konumuna endirir ve işlemi yürütmeye devam eder, aksi halde p işlemi bloke eder.

COMPARE AND SWAP KOMUTU:

İşlemleri bloke etmek veya bu durumdan kurtarmak devamlı ve Senkronizasyon için kullanılıp, paylaşılan bir değişken üzerinde, işlem yapan bir başka komut da `COMPARE-AND-SWAP` (karşılaştır ve yer değiştir). Bu komut `m_addr` adresindeki paylaşılan değişkenin durumu kontrol edilir ve değeri değiştirilir. `r_old` ve `r_new` işlemci saklayıcılarıdır.

`FETCH-AND-ADD KOMUTU` `(CA S r_old, r_new, m_addr)` komutu aşağıdaki şekilde tanımlanır.

Çalışma şekli: Önce bir kere birden fazla işlemcinin

`var m_addr: shared address;` yapması gerekebilir. `Fetch-and-`
`var r_old, r_new: registers;` ve `lerden` `gerçek` `ri` ve
`var z: CAS bayrağı;`

`CAS: if r_old=m_addr then`
`FETCH (m_addr=r_new; z=1)` dir. X, paylaşılan tam sayı

değişkeni ise bir tam sayı ifadesidir. X'in değerini Y ile `gö` `(r_old=m_addr; z=0)` ve `tek` ve `belirli` bir komutta

X'in eski değerini verir ve X'e `y-e` toplamını atar. İşlem başarılı olarak bir kez işlemci tarafından yapılır.

Compare-and-swap işlemeye başladımı kesilemez. Z adlı bir işlemci bayrağı kullanılır.

Kritik bölge meşgul iken bu bölgeye girişi engelleyen iki primitive daha tanımlanabilir. Bunlar wake-up(p) ve block(p) dir. Bu iki işlemci tarafından ortak kullanılan P ve V adlı iki işlem kuyruğudur. Bunlara da senkronizasyon WAKE-UP(P):Eğer P işlemci mantıksal olarak bloke edilmiş ise, durumunu aktif yapar. Aksi halde WWS (Wake-up waiting switch)'i wake-up'ı çağıracağını hatırlamak için set eder. İşlemlerin numaralarını tutarlar.

BLOCK(P):Eğer p işleminin WWS'i set konumunda ise reset konumuna getirir ve işlemi yürütmeye devam eder, aksi halde p işlemini bloke eder.

Semaphore sadece 1 veya 0 oluyorsa, "binary semaphore", integer sayı 4 olabiliyorsa "counting Semaphore" adını işlemleri bloke etmek veya bu durumdan kurtarmak devamlı ve çok fazla yapılması gereken bir iş olup, külfetlidir.Yeni bir primitiv tanımlamak gerekir. Bu FETCH-AND-ADD komutudur.

3-3. Monitör'ler ile Senkronizasyon

FETCH - AND - ADD KOMUTU

Şimdiye kadar incelenen komutların hepsi seri olarak çalışmakta idi. Oysa bir anda birden fazla işlemcinin READ/MOD/FY/WRITE işlemi yapması gerekebilir. Fetch-and-Add komutu diğer primitive lerden gerçekt kılı ve paraleldir. Genel formatı,

FETCH-AND-ADD(X,e) dir. X, paylaşılan tamsayı değişkeni, e ise bir tamsayı ifadesidir. X 'in değerini Y ile gösterirsek, bu primitive tek ve bölünemez bir komutta X 'in eski değerini verir ve X 'e y+e toplamını atar. İşlem eşzamanlı olarak birkaç işlemci tarafından yapılırsa X tüm bu değerlerin toplamı kadar arttırılır.

a-2. Semaphore (Bayraktar) ile Senkronizasyon: CCS'lerin gelişmiş halidir.

Dijkstra karşılıklı yasaklama problemini efektif olarak çözen, pekçok işlemci tarafından ortak kullanılabilen P ve V adlı iki işlem keşfedilmiştir. Bunlara da senkronizasyon primitive'i denir ve kesilmeden çalışırlar. Semaphore adı verilen özel, paylaşılan değişkenler üzerinde işlem yaparlar. Semaphore kritik bölgeye girmek isteyen işlemlerin numaralarını tutarlar.

Var s: semaphore şeklinde tanımlanırlar.

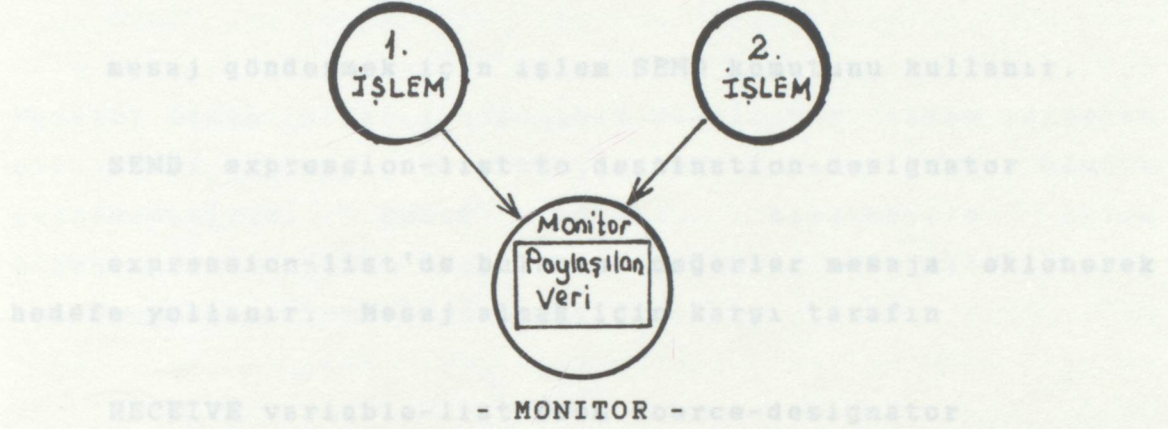
Semaphore sadece 1 veya 0 oluyorsa, 'binary semaphore', integer sayı d olabiliyorsa 'counting Semaphore' adını alır. Semaphoreların kullanışı karşılıklı yasaklama bölümünde gösterilmiştir.

a-3. Monitor'lar ile Senkronizasyon

Semaphore'ların dejavantajı yapısal olmayan işlemler olmalarıdır. Bu da hatalara neden olur. Örneğin her kritik bölgenin aynı semaphore üzerinde bir P işlemi ile başlamak ve bir V işlemi ile bitmesi gereklidir. Eğer bunlardan biri unutulursa veya farklı semaphorelara erişirlerse veya iki kritik bölge aynı semaphore'u referans şekilde, karşılıklı yasaklamış ise, karşılıklı yasaklamanın güvenilirliği olduğu söylenemez. Monitör, karşılıklı yasaklamanın yapısal gerçekleştirilme yoludur.

Ayrıca P ve V işlemlerinin zorluklarına karşılık şartlı kritik bölgeler (Conditional Critical Sections) CCS, keşfedilmiştir. Bu işlem ortak değişkende belli bir değer oluşuncaya bekler. CCS'ler concurrent işlemlere de izin verirler. Concurrent işler arasındaki iletişim aslında V, V, ..., V gibi değişkenlerin bileşimi olan V paylaşılan

değişkeni üzerinden yapılır. Monitor'lar CCS'lerin gelişmiş halidir. İşlemci bir mesaj alıyorsa, bazı değerler alıyor demektir. Mesaj yollama bir senkronizasyon formudur. Çünkü bir mesaj ancak, alındıktan sonra yollanabilir.



Bir monitor, kaynakların durumunu saklayan bir grup sabit değişken ve bu kaynaklar üzerinde işlem gerçekleştiren procedurelerle, sabit değişkenlerin ilk değerlerini içerir. Bu değişkenlerin değerleri monitor procedurelerinin çalışması sonunda oluşurlar ve sadece monitor'un içinden erişilebilirler.

Monitor'un kritik bölge ile ilgili değişkenleri saklaması ile bu değişkenler artık global olmaktan yani paylaşılır olmaktan çıkıp monitor'un gövdesinde saklanan lokal değişkenler olurlar. Aynı monitor içinde bulunan procedureler arasında karşılıklı yasaklama garanti edilmiştir. Monitor'ler ile durum senkronizasyonu işletim sisteminin kullanıldığı semaphore'lar ile olur. Concurrent Pascal ve Modula monitor olanağını destekleyen programlama dilleridir.

B) Mesaj Yollama ile Senkronizasyon

Mesaj yollama bir iletişim biçimidir. Çünkü bir işlemci, bir başka işlemciden mesaj olursa, bazı değerler alıyor demektir. Mesaj yollama bir senkronizasyon formudur. Çünkü bir mesaj ancak, alındıktan sonra yollanabilir.

mesaj göndermek için işlem SEND komutunu kullanır.

Paralel olan bir bilgisayar bir işlem yaparken bir SEND expression-list to destination-designator komutu gerçekleştirir. Komut kunesi, bilgisayara işlem sırasında expression-list'de bulunan değerler mesaja eklenerek hedefe yollanır. Mesaj almak için karşı tarafın

RECEIVE variable-list from source-designator

4.1 Bir Seri Bilgisayar Modeli

komutunu yürütmesi gerekir. Alınan mesajdaki değerler variable-list'deki değişkenlere atanır.

'Random Access Machine' (RAM) tek adresli bilgisayara bir örnektir. girişini teybe (tek alabilen ve çıkışını yalnızca teybe yapabilen, bellek ve bir programdan oluşur.

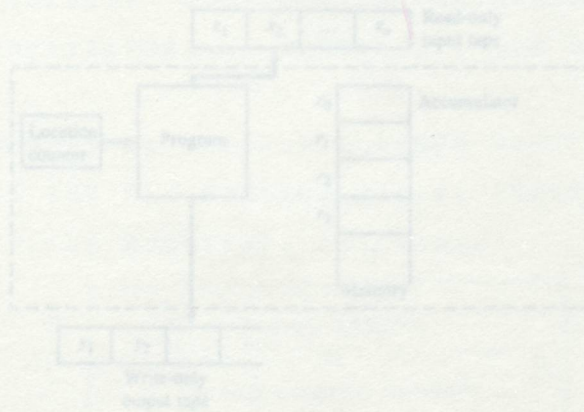


Figure 3-1 The random access machine (RAM) model of serial computation. (Aho, Hopcroft, and Ullman (1974).)

Program bellekte saklanması ve değiştirilmesi. Giriş teybi bir dışı tabanaya içerir. Her adımda, bir giriş değeri okunur. İşaretçi bir sonraki değere gider, çıkışta ise değer teybe yazılır. Çıkış işaretçisi bir sonraki boş değere gider. Belleğin r0, r1, r2, ... gibi sınırlı sayıda

IV. PARALEL VERİ İSLEMEDE KULLANILAN TEMEL BİLGİSAYAR MODELLERİ

Paralel olsun ardışıl olsun, bir bilgisayar işlem yaparken bir veri kümesi üzerinde, istenen bir küme komutu gerçekleştirir. Komut kümesi, bilgisayara işlem sırasında, her adımda yapacaklarını tarif eder. Veri kümesi ise sonuçta bu komutlardan etkilenir.

4.1 Bir Seri Bilgisayar Modeli:

'Random Access Machine' (RAM) tek adresli bilgisayara bir örnektir. girişini teyp 'ten alabilen ve çıkışını yalnızca teybe yazabilen, bellek ve bir programdan oluşur.

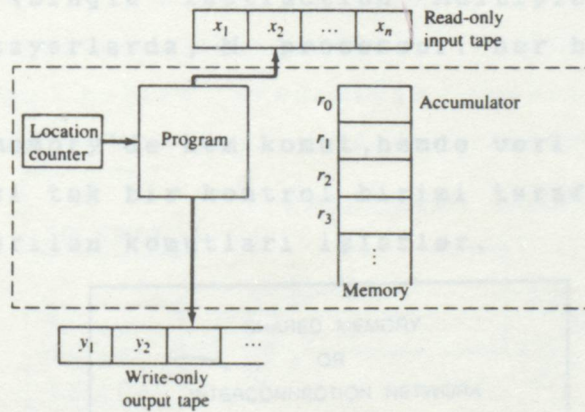


Figure 2-1 The random access machine (RAM) model of serial computation. (Aho, Hopcroft, and Ullman [1974].)

Program bellekte saklanmaz ve düzeltilemez. Giriş teybi bir dizi tamsayı içerir. Her seferinde, bir giriş değeri okunur; İşaretçi bir sonraki değere gider, çıkışta ise değer teybe yazılınca, çıkış işaretçisi bir sonraki boş değere gider. Belleğin $r_0, r_1, r_2, \dots$ gibi sınırsız sayıda



saklayıcıdan ibaret olduğu kabul edilir. Her saklayıcı tek bir değer tutabilir, r0 saklayıcısı akümülatör olarak kabul edilmiş olup, tüm hesaplamalar burada yapılır. Ram'in load, store, read, write, add, subtract, test, jump, halt gibi komutları olmalıdır, yapıdan da anlaşıldığı gibi bir anda bu tipte bir işlem yapılabilmekte, bir sonraki işleme geçmek için bir önceki komutun işinin tamamen bitmesi gerekmektedir.

Paralel bilgisayarlar da sınıflandırma daha önce incelenmiş ve Flynn'ın 4 tipi açıklanmıştı. Bunlar SISD, SIMD, MISD MIMD tipi paralel bilgisayarlardı. Bu bölümde veride tasarlanacak algoritmalarda sık kullanılacak, SIMD bilgisayarda, işlemcilerin ve belleğin durumu ayrıntılı olarak incelenecektir.

SHARED-MEMORY (SIMD) Model Bilgisayarlar

SIMD BİLGİSAYAR MODELLERİ

SIMD (Single Instruction Multiple Data) tipi paralel bilgisayarlar da, N processor.-her birinin local Memory'si var. bu bellek yardımıyla veriler saklanabilir.

-bu memory'de hem komut, hemde veri saklanabilir.

-hepsi tek bir kontrol birimi tarafından gönderilen komutları işlerler.

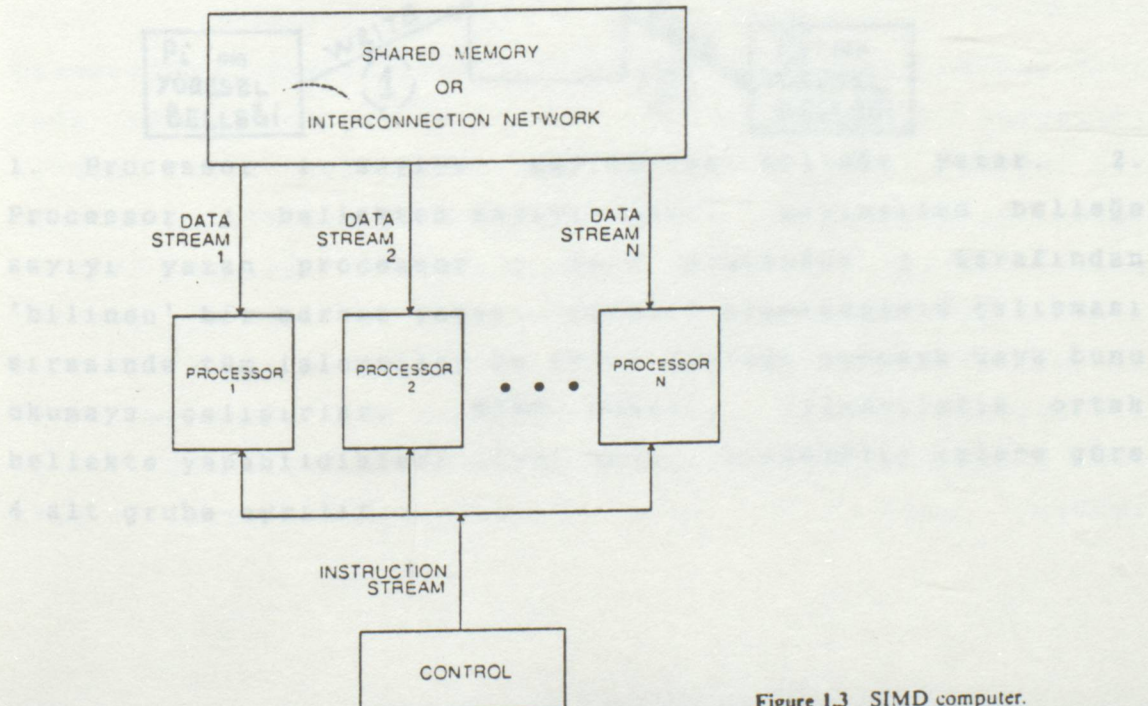


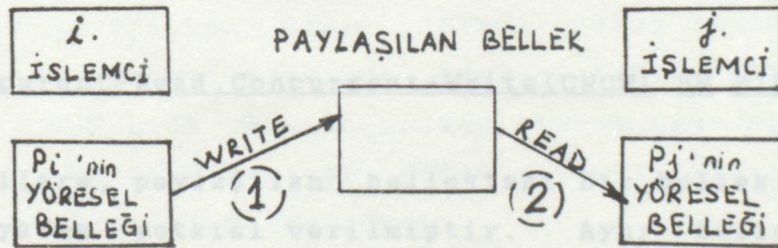
Figure 1.3 SIMD computer.

Kontrol birimi tarafından gönderilen komutun sadece işlemcilerden bir kaç tarafından işletilmesi istenebilir. Bu durumda komut hangi işlemcilerin 'aktif' hangilerinin 'pasif' durumda olacağına dair gerekli kodu'da içerir durumda işlemcilere gönderilir. Bu konfigürasyonda 'global bir saat' mekanizmasına gerek vardır. SIMD tipinde organize edilmiş paralel bilgisayarlar ile çözülmek istenen problemlerde programın çalışması sırasında işlemcilerin verileri veya bir adım önce hesaplanan ara değerleri aralarında değiştirmeleri, bir alışveriş, gerekir. Bu işlem iki yolla yapılır.

- 1- Shared Memory.üzerinde.
- 2- Interconnection Network kullanarak.

SHARED-MEMORY(SIMD) Model Bilgisayarlar

Bu literatürde paralel Random-Access Machine (PRAM) olarak bilinir. Tüm işlemciler bu ortak belleği kullanabilirler.İki işlemci iletişim kurmak istediklerinde bunu bu bellek yardımıyla yaparlar. Örnek: i.işlemci j.işlemciye bir sayı göndermek istiyor.



1. Processor i sayıyı paylaşılan belleğe yazar.
2. Processor j bellekten sayıyı okur. paylaşılan belleğe sayıyı yazan processor i bunu processor j tarafından 'bilinen' bir adrese yazar. paralel algoritmanın çalışması sırasında tüm işlemciler bu ortak belleğe yazmaya veya bunu okumaya çalışırlar. SIMD Modeli, işlemcilerin ortak bellekte yapabildikleri (aynı anda, eşzamanlı) işlere göre 4 alt gruba ayrılır.

SIMD BİLGİSAYAR MODELLERİ

4.2 Paylaşılan Bellekli SIMD Bilgisayar Tipleri

1)-Exclusive-Read,Exclusive-Write(EREW) SM SIMD Bilgisayar:

Bellek bölgesine okuma veya yazma amacıyla yapılan erişim, istekte bulunan işlemciye özeldir. Başka bir deyişle, iki işlemcinin aynı anda aynı bellek gözündeki veriyi okuyamaz, aynı bellek gözüne bir değer yazamazlar.

2)-Conceurent-Read,Exclusive-Write(CREW) SM SIMD Bilgisayar

Birden fazla işlemci aynı bellek bölgesindeki bilgiyi okuyabilir ama aynı anda aynı göze bilgi yazamazlar.

3)-Exclusive-Read,Concurrent-Write(ERCW) SM SIMD Bilgisayar

Bir bellek gözüne birden fazla işlemci aynı anda yazma yapabilir. Ama bu bölgeyi birden fazla işlemci aynı anda okuyamaz.

4)-Concurent-Read,Concurrent-Write(CRCW) SM SIMD Bilgisayar

İşlemcilere, paylaşılan bellekteki bir bellek gözünü aynı anda yazma yetkisi verilmiştir. Aynı bellek bölgesini birden fazla işlemcinin eş zamanlı okuması büyük bir problem yaratmaz. SIMD bilgisayarda her işlemcinin kendi yöresel belleği olduğuna göre; okunmalı istenen bellek gözünün içeriği tek tek buralara kopyalanır. Her işlemci bilginin kendi belleğindeki kopyasını kullanır. Eş zamanlı yazma durumunda ise problem büyüktür. Birçok işlemci bir bellek gözündeki değeri değiştirmek için savaşıacak, ama hangisi başarılı olacaktır? Olsa bile onun yazmayı

başardığı veri diğer işlemcilerin işine yaramayacaktır? Bu problemi çözmek için birkaç politika izlenebilir. Bunlar

1 - 'En küçük numaralı' işlemcinin isteği kabul

edilir, diğer işlemcilerin yazma isteği

reddedilir.

2 - Yazmak istedikleri değer 'birbirine eşit'

olan işlemcilere izin verilir, diğer

işlemcilerin isteği reddedilir.

3 - Tüm işlemcilerin yazmak istedikleri

değerlerin 'TOPLAMI' istenen bellek

gözüne yazılır.

İleride dikkat edilirse matris çarpımı algoritmalarında 3. çözüm kullanılmıştır.

4.3 İç bağlantı Ağılı SIMD Bilgisayar Tipleri : (Interconnection - Network SIMD)

Paylaşılan Bellekli SIMD Bilgisayarda bellek bloklara ayrılır ve bu bloklara erişimde eşzamanlılık kaldırılırsa model daha fizibel olur. Şimdi ortak belleğin M adet gözünün N işlemci arasında paylaşıldığını düşünelim. Her işlemci M/N adet bellek gözüne sahip olur. Her işlemcide diğer tüm işlemcilere iki yönlü hatlarla bağlanırsa, Durum aşağıdaki gibi olur. ($N=5$) için

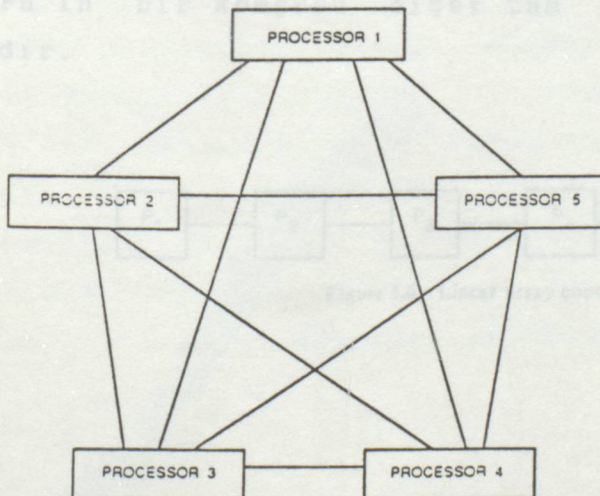


Figure 1.5 Fully interconnected set of processors.

Bu duruma tamamen bağılı (fully-connected) durum denir. ve $N(N-1)/2$ hat olduğu İşlemin herhangi bir adımında P_i işlemci P_j işlemciden bir bilgi alarak bunu P_k işlemciye gönderebilir. Sonuç olarak her işlemci :

1)- Kendi dışında kalan $(N-1)$ işlemcinin hangisi ile iletişim kuracağını bulabilmek için $\log(N-1)$ bit'lik bir adres çözen devreye ihtiyaç duyar. Bunun maliyeti $f(N-1)$ ile ifade edilir.

2)- Bir başka işlemci tarafından sağlanan $\log(M/N)$ bit'lik adresi çözecek $f(M/N)$ maliyeti devreye ihtiyacı vardır. Bu model bloklanmış paylaşılan bellekli modelden daha efektiftir. Fully-interconnected network N adet işlemcili ise maliyeti $N(N-1)/2$ dir. Bu ise işlemci sayısının artmasıyla maliyetin aşırı artması demektir. Ayrıca işlemci sayısı fazla ise model fiziksel olarak da gerçekleştirilemez; çünkü işlemcilerin boyutlarından dolayı bir işlemciye bağlanabilecek hat sayısı sınırlıdır. Sonuç olarak daha az bağlantılı yapılar kullanmak gerekir. Şimdi bunları görelim:

1) Lineer Dizi:

N işlemcinin en basit bağlama yoludur. İşlemciler tek boyutlu bir dizi şeklinde bağlanırlar. $N=6$ için bağlantı aşağıda gösterilmiştir. Bağlantı yolları iki yönlüdür. P_1 ve P_n 'in bir komşusu diğer tüm işlemcilerin iki komşusu vardır.

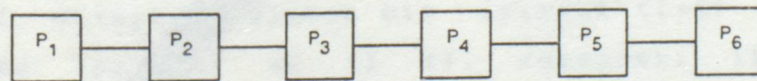


Figure 1.6 Linear array connection.

2) İki Boyutlu dizi: (Mesh Connection)

N adet işlemci $M = \sqrt{m}$ olmak üzere $m \times n$ boyutlu bir matris şeklinde bağlanırlar. j .sattır k sütundaki işlemci $P(j,k)$ ile gösterilir.

$$0 \leq j \leq m-1 \text{ ve } 0 \leq k \leq m-1 \text{ dir.}$$

$P(j,k)$ komşuları olan $P(j+1,k)$, $P(j-1,k)$, $P(j,k+1)$, $P(j,k-1)$ 'e iki yönlü yollarla bağlıdır. Sınırlardaki işlemcilerin komşu sayısı 4'ten az sayı 3'tür. $N=4$ için mesh tipi bağlantı şeması aşağıda verilmiştir.

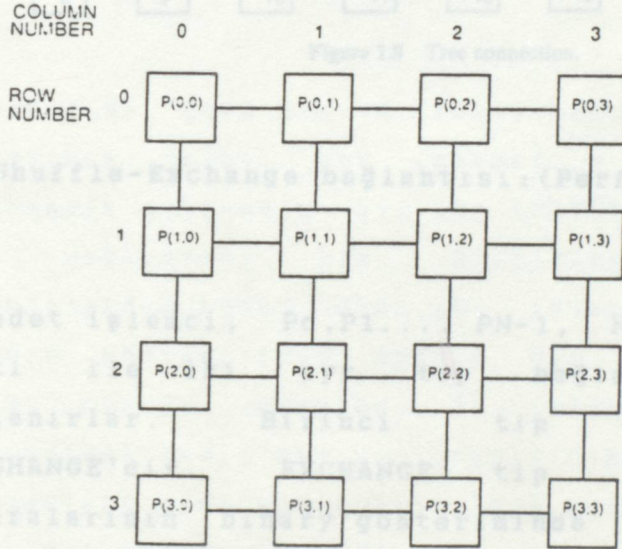


Figure 1.7 Two-dimensional array (or mesh) connection.

3) Ağaç Bağlantısı (Tree Connection)

Bu durumda tam bir ikili ağaç (binary tree) oluşturacak şekilde düzenlenmişlerdir. Ağaçın düzeyi 'd' olup 0'dan (d-1)'ekadar numaralanır. N işlemci sayısı olmak üzere $N = 2^d - 1$ dir. Ağaçın her bir düğüm noktası bir işlemciye karşı gelir. i . düzeyde bulunan bir işlemci $(i+1)$ düzeyindeki ebeveynine (parent) ve $(i-1)$. düzeydeki iki çocuğuna (children) iki yönlü yol ile bağlıdır. Ana kökteki (root) işlemcinin (0. düzeyde) ebeveyni yoktur. Benzer olarak

yapraklardaki (n -1.düzeyde işlemcilerin de çocukları yoktur. N = 15 işlemcinin d = 4 düzeyli bir ağaca yerleştirilmesi aşağıdaki gibidir.

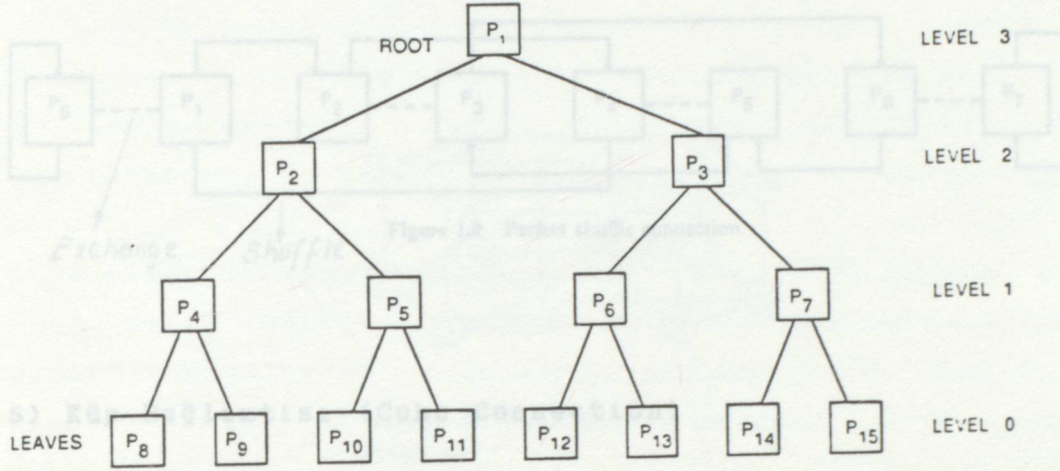


Figure 1.8 Tree connection.

4)-Shuffle-Exchange bağlantısı:(Perfect-Shuffle Connection)

N adet işlemci, $P_0, P_1, \dots, P_{N-1}$, N ikinin kuvveti olmak şartı ile iki ayr tip bağlantı ile birbirlerine bağlanırlar. Birinci tip 'SHUFFLE' ikinci tip 'EXCHANGE'dir. EXCHANGE tip bağlantılar, işlemci numaralarının binary gösteriminde en anlamlı biti (Most significant bit) birbirinden farklı işlemci çiftlerin birbirine bağlanması için kullanılır. Bu bağlantılar iki yönlüdür. Bir başka deyişle EXCHANGE Bağlantılar her çift numaralı işlemciyi kendisinden hemen sonra gelen tek numaralı işlemciye kendisinden hemen sonra gelen tek numaralı işlemciye bağlar. 'SHUFFLE' bağlantılar ise şu formülle tanımlanır. P_i işlemciden P_j işlemciye bir Shuffle bağlantı varsa.

$$j = \begin{cases} 2i & 0 \leq i \leq N/2 - 1 \\ 2i + 1 - N & N/2 \leq i \leq N - 1 \end{cases}$$

Hesaplanan j değeri P_j işlemciyi belirler. SHUFFLE bağlantılar tek yönlüdür. Aşağıda $N=8$ için Perfect-Shuffle görülmektedir.

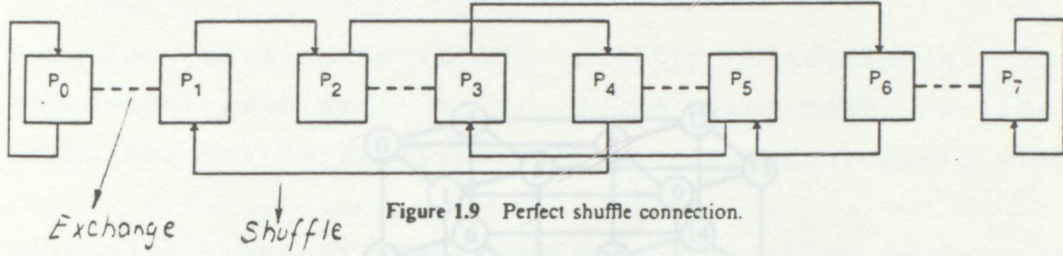


Figure 1.9 Perfect shuffle connection.

5) Küp Bağlantısı (Cube Connection)

$N = \exp(2, q)$ $q \geq 1$ ise ve $P_0, \dots, P_{(N-1)}$ olmak üzere N adet işlemcimiz varsa; her işlemci q adet komşu işlemciye bağlanarak q -boyutlu bir küp (HYPERCUBE) elde edilebilir. Küp bağlantısı ile birbirlerine bağlanmış olan işlemcilerden komşu olan iki işlemcinin numaraları ikili düzende gösterilirse sadece bir bitlerinin birbirinden farklı olduğu görülür. P_i işlemcisinin komşuları P_j işlemcileri ise j sayısı i sayısının 1 bitinin tümleyeninin alınmasıyla elde edilebilir. $q=3$ için P_0, P_1, P_7 nin ikili ifadeleri sırasıyla 000, 001, 111 dir ve bu iki işlemci komşudur. Aşağıdaki şekilde $q=3$ için bu durum gösterilmektedir.

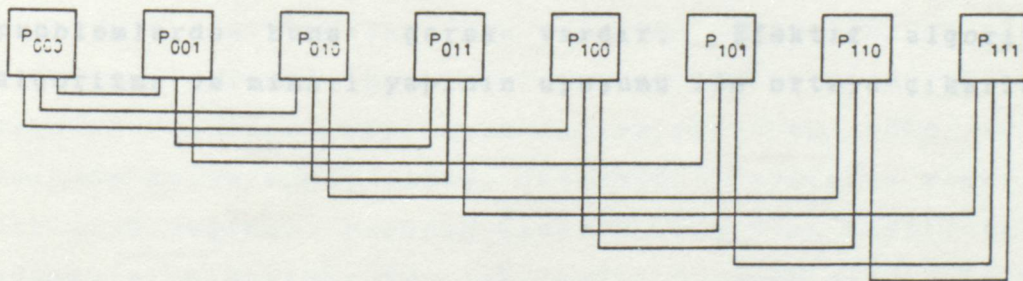


Figure 1.10 Cube connection.

Aşağıdaki şekilde ise $q=4$ için 16 işlemcinin durumu gösterilmektedir.

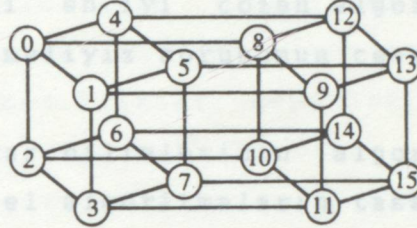
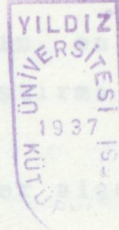


Figure 2-7 Four-dimensional (16-node) hypercube.

Bu bölümde genel amaçlı mimariler anlatılmıştır. Belirli ve özeline bir problem için özel amaçlı paralel bilgisayar tasarımları da yapılmaktadır.

Sonuç olarak şu söylenebilir. Belli bir algoritmanın tasarımı sırasında, kullanılan mimari yapıyı, işlemci sayısını ve bunların nasıl haberleştiklerini bilmek gerekir. Genellikle tasarımcının önünde iki yol vardır. Ya zaten belli olan donanımına uygun bir algoritma tasarlamak için uğraşır. Ya da baştan problemi çözmek için gerekli algoritmayı tasarlarlarken, mimari yapının da nasıl olması gerektiğine de karar verir. Özellikle büyük çaplı problemlerde buna gerek vardır. Efektif algoritmalar algoritma ve mimari yapının uyumu ile ortaya çıkarlar.



ALGORİTMALARIN ANALİZİ

Pek çok problem için çok çeşitli algoritmalar vardır. Önümüzdeki problemi en iyi çözen algoritmayı seçmek için nasıl bir yol izlemeliyiz sorusunun cevabını nasıl veririz?

Bu soruya bilgisayar bilimlerinin algoritma analizi konusu cevap arar. Paralel algoritmaların tasarımını etkileyen en önemli etmenlerden biri olan, bilgisayarın mimarisidir. Mimari yapı en önce maksimum kaç adet işlemci kullanacağımızı belirler. İşlemci sayısı doğrudan paralellik düzeyini, ayrıştırmayı (Decomposition) etkiler. Daha sonra kullanılabilecek işlemcilerin birbirleri ile nasıl iletişim kurdukları, gereksinme duyduklarında ortak bilgileri nerelerden aldıkları ve oluşturdukları bilgileri nereye yazdıkları, hep algoritmaların tasarımını etkileyen unsurlardır.

Genellikle üzerinde uğraştığımız problemin doğal bir boyutu, büyüklüğü vardır. Bu genellikle $\ll N \gg$ ile ifade edilir; tipik olarak işleme girecek verilerin adedidir. N , bir polinomun derecesi, sıralanacak bir file'in veya dizinin boyutu, bir grafın düğüm sayısı olabilir. Örneğin bir sayı dizisinde dışarıdan verilen bir değer aranıyorsa ve dizi 20 elemanlı ise $\ll N = 20 \gg$ dir.

Bir algoritma incelenirken genelde "'Ortalama durum'" ve "'En kötü durum'" üzerinde durulur. En kötü durum girişte verilen verinin yapılacak işlem için en kötü ve ters durumda düzenlenmiş olması durumunda karşımıza çıkar. Sayı dizisi örneğinde program dizinin başından başlayarak her elemanın dışarıdan verilen sayının eşit olup olmadığını test ediyorsa aradığınız elemanın dizinin 20. elemanı

olması durumu bu algoritma için en kötü durumdur. Çünkü bu sayıyı bulmak için 20 karşılaştırma yapmak gerekir.

Bir programda belirli işlemler algoritmanın karmaşıklığını etkilerler. Bunların başında karşılaştırma işlemleri (IF komutları) gelir. Bunun yanında fonksiyonlar ile yapılan hesaplamalar da fazla zaman alan işlemler arasındadır. Üst hesaplama, trigonometrik fonksiyonları kullanma v.b. Algoritmaların karmaşıklıkları hesaplanırken bu ve benzeri işlemlerin kaçar defa yapıldıkları eğer bir çevirim içindeyseler çevirimlerin sınırları gözönünde bulundurularak algoritmanın karmaşıklığı hesaplanır.

Algoritmaların Sınıflandırılması:

1 :

Çoğu programdaki çoğu işlem sadece bir veya birkaç defa yürütülür. Eğer bir programdaki tüm işlemler bu özellikte iseler bu programın çalışması belirli sabit bir zaman alır. Algoritma tasarımı bu tip algoritmalar yazmak için çalışıp didiniriz.

log N :

Bir programın çalışma zamanı logaritmik ise N sayısı büyüdükçe program hafifçe yavaşlamaya başlar. Bu çalışma zamanı, genellikle büyük bir problemi, belli sabit bir sayıda daha küçük boyutlu problemlere bölerek alt problemler cinsinden ifade ettiğimiz durumlarda oluşur. Çalışma zamanı ilk durumdaki büyük sabit sayıdan daha küçük olur. Logaritma bu sabiti küçültür, ama çok fazla da değil. $N=1000$ için logaritmanın tabanı 10 ise $\log N=3$ tür.

$N=1.000.000$ ise $\log N=6$ dır; yani problemin boyu 1000 kat artmasına rağmen çalışma zamanı sadece 2 katına çıkmıştır. N iki katına çıkarsa $\log N$ sadece belli bir sa dar artar ve iki katına çıkmaz. Çalışma zamanı ancak problemin boyutu N^2 olduğunda iki katına çıkar.

N :

Çalışma zamanının lineer olması durumunda genellikle her giriş elemanı ile az N miktarda işlemin yapıldığı algoritmalarda rastlanır. N iki katına çıktığı zaman çalışma zamanı da iki misli artar. N adet giriş işleyen (veya N adet sonuç veren) algoritmalar için optimal bir durumdur.

$N \cdot \log N$:

Bu çalışma zamanı büyük boyutta olan bir problem olduğu için daha küçük alt programlara bölünür, bunlar birbirinden bağımsız olarak çözülür, sonuçlar birleştirenince oluşur. $N=1.000.000$ ise çalışma zamanı uygun bir taban için diyelim ki 20 milyondur. N iki katına çıkarsa çalışma zamanı iki katından fazladır (ama çok değil).

N^2 :

Çalışma zamanı N 'in karesi olan programlar bağıl olarak küçük miktarda veri ile işlem yapıyor olmalıdırlar. Çünkü örneğin $N=1000$ ise N iki misli arttığında çalışma zamanı bir öncekinin 4 katı artar. Örneğin iç içe iki çevirim içinde N adet işlem yapılıyorsa bu gerçekleşir.

ALGORİTHALARIN HESAPLANAN $N \cdot N^{**2}$: PLEKSLİCİ:

Bir önceki duruma benzer. $N=100$ iken çalışma zamanı 1.000.000 dur. N iki katı artarsa çalışma zamanı 8 katı artar.

$\exp(2,n)$: Bu metod işi çeker. Çünkü kullanıcılara algoritmanın performansını

Çalışma zamanı üstel olan algoritmalara pratikte ender olarak rastlanır. Örneğin $N=20$ ise çalışma zamanı yaklaşık 1.000.000 dur. N iki katına çıkarsa çalışma zamanı bir önceki değerin karesine erişir.

Doğal olarak her algoritmanın çalışma zamanını bu formlara tamamen sokma imkansızdır. Bazı algoritmalarda yukarıda tanımlanan bu terimlerin yanın küçük sabit terimler de eklenebilir. Genelde de logaritma tabanı olarak veya doğal logaritma seçilir. Şimdiye kadar açıklanan çalışma zamanı formülleri, bir fikir edinilmesi için, N sayısının belli

Ref12. Sh 71 figure 6.1

Algoritma - 1/10/61

değerleri için aşağıda bir tablo halinde verilmiştir.

ALGORİTMALARIN HESAPLANABİLİR KOMPLEKSİLİĞİ:

Algoritmaların kompleksliğini incelerken izlenecek bir metod da, sabit sayıların etkisini atarak, çalışma zamanının giriş büyüklüğü olan $\langle\langle N \rangle\rangle$ sayısına (veya diğer bazı değişkenlere) fonksiyonel bağımlılığını bulmak için, en kötü durum performansı hesaplamaktır. Bu metod ilgi çekicidir. Çünkü kullanıcıya algoritmanın performansını matematiksel olarak kanıtlama olanağı verir. Örneğin Merge Sort Algoritması'nın $N \cdot \log N$ zamanda çalışması garanti edilmiş olur.

Bu işlemde çoğu zaman sabit terimler atlanır. Bu durumda nisbi bir hesaplama yapıldığı için programın bir mikro bilgisayarda veya süper bilgisayarda çalışıyor olması hiç bir şey farketmez. Bu nisbi performansı göstermek için $\langle\langle O \rangle\rangle$ (büyük O) harfi kullanılır. **PARALEL ALGORİTMALARIN ANALİZİ:** Çalışma Zamanı $t(n)$: Paralel algoritma tasarımının ana amacı ilgilenilen problemi daha hızlı çözmek, dolayısıyla daha hızlı çalışan programlar yazmak olduğuna göre, paralel algoritma tasarımında en önemli ölçü $\langle\langle\text{Çalışma Zamanı}\rangle\rangle$ dır. Bu, programın çalışmaya başlamasından bitimine kadar geçen zaman olarak tanımlanır. Eğer çeşitli işlemciler bir işlemi, işe aynı anda başlayıp bitirerek, hesaplamalarını eşzamanlı olarak yapmazlarsa çalışma zamanı ile işini en son bitiren işlemcinin işini bitirme zamanı arasındaki zaman dilimidir.

İşlem Adımlarının Sayılması:

Bir algoritma ardışıl olsun paralel olsun bir bilgisayar üzerinde bire bir çalıştırılmadan önce ne kadar zaman gerektiği hesaplanabilir. Bu genel olarak en kötü durumda bazı temel işlemlerin, işlem adımlarının sayılması ile

bulunur. Bu adımların sayısı sonuçta problemin giriş büyüklüğü olan $\langle N \rangle$ sayısının bir fonksiyonu olarak elde edilir. Toplama, çıkarma, çarpma, bölme, yer değiştirme, karşılaştırma bu temel işlemlerdendir. Bunların herbiri sabit bir zamanda gerçekleşir.

Paralel bir algoritmanın çalışma zamanı genelde iki tür işlemin adımlarının sayılmasıyla bulunur. Bunlardan birinci tip 'Hesaplama Adımları'; ikinci tip ise 'Atamalar' dır.

'Hesaplama Adımları' işlemcide bir veri üzerinde yapılan aritmetik veya lojik işlemlerden oluşur. Bir 'Atama' adımı ise veri işlemciler arasında ortak bellek (Shared memory) veya iletişim ağı (communication network) üzerinden transfer edilir. Büyüklüğü N olan bir paralel algoritmada çalışma zamanı $t(N)$ olarak gösterilir.

Genel olarak 'Hesaplama Adımları' ve 'Atama Adımları' nın harcadıkları zaman birbirinden farklıdır. Örnek olarak bir işlemcinin iki sayıyı toplaması için geçen zaman, bu sayının ikinci bir işlemciye gönderilmesi için geçen zamandan farklıdır. Atama adımlarının aldıkları zaman paralel bilgisayarın mimarisine ve işlemcilerin birbirinden olan uzaklıklarına bağlıdır.

ALT SINIRLAR VE ÜST SINIRLARı

Bir problem hakkında yeni bir algoritma tasarımı yapıldığında bu yeni algoritmanın şimdiye kadar problemi çözen en hızlı algoritma olup olmadığı ve eğer değilse bilinen diğer algoritmalarla nasıl mukayese edileceğidir.

Birinci sorunun cevabını almak için algoritma tarafından yürütülen adımların sayısı ile en kötü durumda bu problemi

çözmek için gerekli adım sayısının bilinen alt sınırı karşılaştırılır. Örnek $n \times n$ boyutlu iki matrisin çarpımını bulmak istiyorsak sonuç matris $n \times n$ boyutlu olacağından herhangi bir matris çarpımı algoritmasında en az $n \times n$ adet adıma gerek olacaktır. Alt sınırlar bu örnekte olduğu gibi önemsiz denecek kadar az veya belirlidir. Bu örnekte kolaylık olarak adımlar sayılarak bulunmuştur. Daha zor bir örnek inceleyelim. Random sırada verilmiş N adet sayının küçükten büyüğe sıralanması isteniyorsa giriş $n!$ mümkün permütasyon biçiminde olabilir ve $\log(n!)$ (yani $n \cdot \log(n)$ mertebesinde) bit kullanılarak sayılar birbirinden ayrılabilir. Bundan dolayı sıralı diziyi elde etmek için en kötü durumda 'en azından' $n \cdot \log(n)$ mertebesinde adım gerekir.

Eğer bir algoritmanın en kötü durumda çalıştığı zaman yürüttüğü adım sayısı alt sınıra eşit (veya aynı mertebeden) ise bu algoritma mümkün olan en hızlı algoritmadır ve OPTİMAL dir denir. Aksi halde bilinen algoritmalarından daha hızlı bir algoritma keşfedebilir veya ant sınır daha da iyileştirilebilir demektir. Her ne olursa olsun bilinen tüm algoritmalarından daha hızlı bir algoritma bulunmuş ise en kötü durum için gerekli adım sayısı üzerine yeni bir üst sınırın oluştuğu söylenir. Bundan dolayı yazılan yeni bir algoritmanın çalışma zamanı daha önce bilinen en hızlı algoritmanın oluşturduğu üst sınırla karşılaştırılır.

Örneğin şu anda bilinen hiç bir algoritma $n \times n$ boyutlu iki matris çarpımını $n \times n$ adımda yapmaz. Sonuç matrisin her elemanını bulmak için n adet çarpma yapmak gerekir. Sonuç matris $n \times n$ elemanlı olacağına göre $n \cdot (n \cdot n)$ kadar yani toplam n^3 'ün kübü kadar işlem yapmak gerekir. Bu arada çarpımı bulan $\exp(2, x)$ $x < 2.5$ olan bir algoritma da mevcut olup, bu üst sınırı oluşturur. Bunun tersi olarak sıralama algoritmaları en fazla $n \cdot \log(n)$ adım gerektirirler. Bu

sebepten optimaldirler.

$g(n)$ fonksiyonu tüm $n \geq (0)$ sayıları için $g(n) \geq c \cdot f(n)$ şartını sağlayan c ve $n(0)$ pozitif sabitleri var ise bu $g(n)$ fonksiyonu "'en azından $f(n)$. mertebededir'" denir ve $\omega(f(n))$ ile gösterilir.

$g(n)$ fonksiyonu tüm $n \geq n(0)$ sayıları için $g(n) \leq c \cdot f(n)$ şartını sağlayan c ve $n(0)$ pozitif sabitleri var ise bu $g(n)$ fonksiyonu "'en fazla $f(n)$. mertebededir'" denir ve $O(f(n))$ ile gösterilir.

Bu notasyon alt ve üst sınırları belirtirken baskın olan (ağırlıklı) terimi üzerinde konsantre olmamızı ve çarpım durumunda bulunabilecek herhangi bir sabiti görmezlikten gelmemizi sağlar.

Matris çarpımı algoritması için alt sınır $\omega(n^2)$, üst sınır ise $O(n^3)$ dur. Sıralama algoritmalarında alt sınır $\omega(n \log(n))$, üst sınır ise $O(n^2 \log(n))$ dir.

Buraya kadar alt ve üst sınırları ardışıl algoritmalar için inceledik. Açık olarak aynı genel kurallar paralel algoritmalar için de uygulanabilir. Sadece iki nokta gözönünde bulundurulmalıdır. Bunlar:

- 1- Kullanılan paralel hesaplamanın tipi.
- 2- Gerekli işlemcilerin sayısı.

Örnek olarak $n \times n$ boyutlu "'Mesh'" tipi bağlantılı SIMD tipi bir bilgisayar için n^2 adet sayının toplamının bulunması problemini ele alalım. Her işlemcinin ilk değeri bu sayılardan biridir. $P(n-1, n-1)$ işlemciye gidecek bir yol bulmalıdır. Bu en azından $2 \cdot (n-1)$ atama işlemi gerektirir. Sonuç olarak bu toplama algoritmasının alt sınırı $\omega(n^2)$ dir.

VI.

PARALEL ALGORİTMALAR

6.1 Genel Özellikleri

Çoklu işlemciler ile tasarlanmış bir paralel algoritmanın esası verilen problemi çözen, işbirliği içinde çalışan bir küme (k) adet birbirinden ayırık (concurrent) işlemden oluşur. Bir iş sisteminde bir işlemin diğer işlemlerle iletişim kurdukları noktalar olabilir. Bu noktalara 'Etkileşim noktaları (Interaction Point) denir. Etkileşim noktası paralel işlemleri safhalara (stage) ayırır. Bundan dolayı bir işlem bir sonraki safhada yapılan işlemlerden bağımsız ise diğer safhalar ile iletişim kurabilir. İşlemler arasındaki etkileşimden dolayı bazı işlemler belirli zamanlarla sınırlandırılabilir. Bu açıklamalardan yararlanarak aşağıdaki tanımları yapabiliriz.

MACRO PIPELINING

Bu yöntem eğer hesaplamalar safhalara bölünebiliyorsa uygulanabilir. Bir veya pek çok bölümün çıkışları bir başka bölümün girişidir. Aşağıdaki Şekil de bu görülmektedir.

6.3 SENKRON PARALEL ALGORİTMALAR

(Synchronised Parallel Algorithms)

Bir paralel algoritmada işlemlerin bazıları diğer bazı işlemlerinden beklemek zorundaysalar bu tipte paralel algoritmalara Senkronize Paralel Algoritmalar denir. Bir işlemin işleme zamanı (Execution Time) değişken olduğu için, giriş verisine ve sistem kesmelerine bağımlı olarak

YILDIZ
ÜNİVERSİTESİ
1937
KÜLTÜR

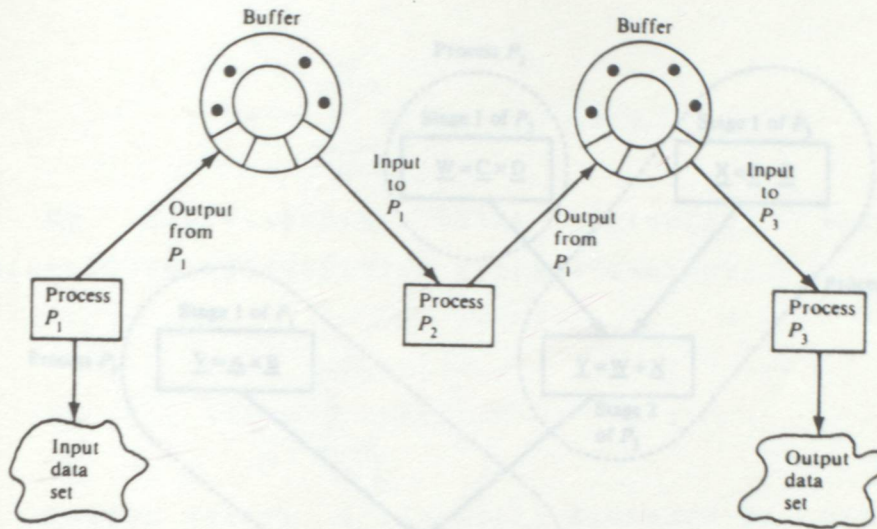


Figure 8.26 Program flow in macropipelines.

tüm işlemler aralarındaki en yavaş işlemi beklemek için senkronize edilmelidirler. Bu en kötü durumda hesaplama hızını düşürür. Bu ise senkronize algoritmaların temel zayıflığıdır ve algoritmanın beklenenden daha kötü çalışmasına az bir hız artışına ve kötü işlemci kullanımına neden olur. Senkronize paralel algoritmaların bu dezavantajlarına kari Senkronize olmayan (Asenkron) paralel algoritmalar geliştirilmiştir. Senkronize paralel bir algortmada mevcut bir işlemin bir safhası bir başka işlem kendi safhalarından belirli birinin işini bitirmeden aktif olamaz. Senkronizasyon ileride de BÖLÜM 2.6 de anlatılacağı gibi Semaphore'lar, monitor'lar ve kritik bölgeler ile sağlanır.

Örnek olarak $Z = A * B + (C + D) * (I + G)$ matris işlemini yapan Senkronize Paralel Algoritma tasarlayalım. P_1, P_2, P_3 yaratılacak işlemler olsun. Açıkça görülmektedir ki P_3 işleminin 2. safhası P_2 işleminin bitmesinin kontrolündedir. Benzer olarak P_1 in de ikinci safhası P_3 ün 2. safhası tamamlanmadan ilk değer alamaz. Böylece ilk P_1, P_2, P_3 kümesinden oluşan işlem bir senkronize paralel algortmadır. n adet işlem den oluşan senkronize bir

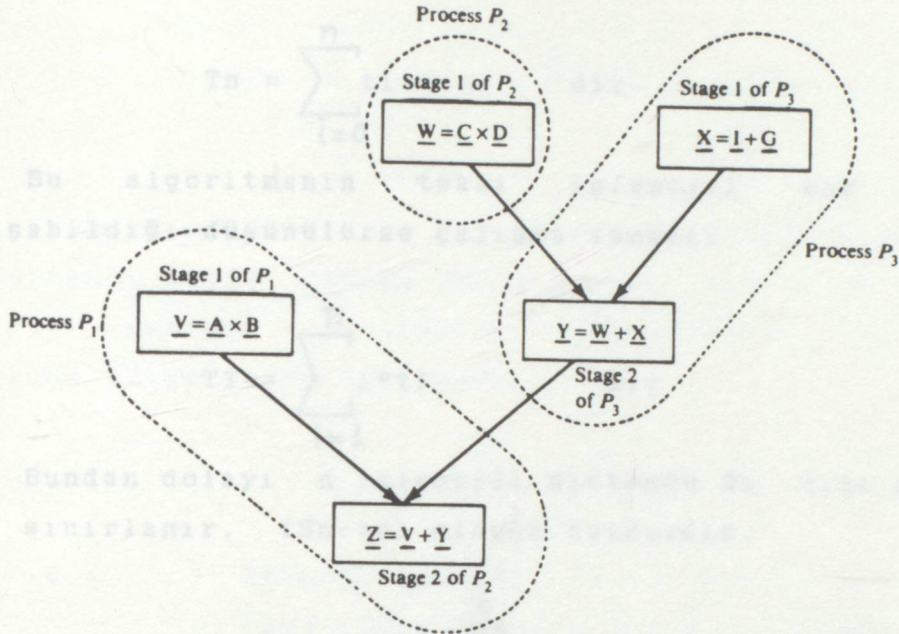


Figure 8.27 Example of a synchronized algorithm with synchronizing stages.

Example 8.9

var W, Y: shared real; var S_w, S_y : semaphore; initial $S_w = C_y = 0$;
cobegin

Process P_1 : begin
 $V \leftarrow A \times B$; // stage 1 of P_1 //
 $P(S_y)$;
 $Z \leftarrow V + Y$; // stage 2 of P_2 //
end

Process P_2 : begin
 $W \leftarrow C \times D$; // stage 1 of P_2 //
 $V(S_w)$;
end

Process P_3 : begin
 $X \leftarrow I + G$; // stage 1 of P_3 //
 $P(S_w)$;
 $Y \leftarrow W + X$; // stage 2 of P_3 //
 $V(S_y)$;
end

coend

Paralel algoritmanın homogen bir çokluışlemcili sistemde T_n zamanını aldığını farzedelim. Algoritmanın çalışması sırasında i adet işlemin çalışması için geçen toplam zaman

ile belirtilirse n-i adet işlem bloke edilir. Bundan dolayı

$$T_n = \sum_{i=0}^n t_i \quad \text{dir}$$

Bu algoritmanın tekli işlemcili bir sistemde çalışabildiği düşünülürse çalışma zamanı:

$$T_1 = \sum_{i=1}^n i * t_i \quad \text{dir}$$

Bundan dolayı n işlemcili sistemde S_n hızı aşağıdaki gibi sınırlanır. ($S_n \leq n$) olduğu aşıkardır.

6.4 SENKRON İTERATİF ALGORİTMALAR

$$S_n = \frac{T_1}{T_n} = \frac{\sum_{i=1}^n i * t_i}{\sum_{i=1}^n t_i} \quad \text{dir.}$$

Performanstaki düşüşleri belirlerken paralel algoritmanın sınıfına dikkat edilmelidir. SIMD Algoritmalarında işlemlerin genellikle aynı safhaları senkronize edilmiştir. Bir küme işlemi ve bu işlemlerin n adet safhasının senkronize edilmek istendiğini düşünelim; i. safha tarafından harcanan random süre t_i dir. Tüm safhalar tamamen aynı olduklarından bunların ortalama değeri 't' olsun. n adet safhanın senkronizasyonu bu n safha tamamlanmadan herhangi bir işlemin aktif olamayacağını belirtir. T' ortalama değer olmak üzere herhangi bir işlemin senkronize safhasının alması gereken zaman T' den ziyade T dir.

$$T = \max (t_i) \quad \text{dir.}$$

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y)$$

Genelde $T' > t$ dir. ve n adet benzer safha için

$$T'/t = \text{LAMBDA} \lambda(n) \text{ penalty faktörüdür.}$$

Lambda n büyük ise senkronize paralel algoritmanın performansı büyük ölçüde düşer. Sn hız sınırı ve Lambda n penalty faktörü senkronize algoritmanın performansı hakkında bilgi veren iki büyüklüktür.

6.4

SENKRON İTERATİF ALGORİTMALAR

(Synchronized Iterative Algorithms)

Pratikte pek çok problem iteratif algoritmalar ile çözülebilir. Örneğin f fonksiyonunun kökleri iteratif bir metod olan Newton metodu ile hesaplanabilir.

$$x_{i+1} = x_i - f'(x_i)^{-1} f(x_i)$$

Yukarıdaki formül ile aşağıda belirtilen lineer denklem sistemi elde edilir. Burada X_i ve b n boyutundaki vektörler A ($n \times n$) matristir. Bu lineer denklem sistemi iteratif bir paralel algoritma ile çözülebilir.

$$X_{i+1} = AX_i + b$$

İteratif bir paralel algoritmaların kullanıldığı bir başka örnek ise Eliptik Diferansiyel Denklem çözümüdür. Bu bir sınır değer problemi olarak ele alınır. Problem D domeninde aşağıdaki formülü sağlayan $u(x,y)$ fonksiyonunu bulmaktır.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y)$$

Yukarıdaki formülden yararlanarak D kapalı bölgesinde sınır değer problemini çözmek için D bölgesinin üzerine mXn boyutlu bir kafes sistemi konarak bölge örneklenir. Bu kafes içinde kalan iki nokta arası yatay veya düşey çizgi üzerindeki uzaklığa kafes genişliği denir ve h harfi ile gösterilir. Eğer h yeter derecede küçük ise aşağıdaki

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u(x+h, y) + u(x-h, y) - 2u(x, y)}{h^2}$$

$$\frac{\partial^2 u}{\partial y^2} \approx \frac{u(x, y+h) + u(x, y-h) - 2u(x, y)}{h^2}$$

yaklaşık ifadeyi elde ederiz. Bu ifadeyi açık yazarsak:

$$4u(x, y) - u(x+h, y) - u(x-h, y) - u(x, y+h) - u(x, y-h) = -h^2 f(x, y)$$

Sadece kafes sistemi içinde kalan noktalar düşünülerek aşağıdaki ifade elde edilir. ($i=1, 2, 3, \dots, m$ ve

$$u_{ij} = \frac{1}{4} (b_{ij} + u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1})$$

$j=1, 2, 3, \dots, n$) ve Sonuç iteratif olarak farklı metodlarla elde edilebilir ama bu çalışmada İki metod tartışılacaktır: Birinci metodda senkronize paralel algoritma kullanılarak Kısmi Türevli Diferansiyel Denklem Çözümü incelenecek daha sonra asenkron paralel algoritma incelenecektir. Her iki durumda da yukarıda tanımladığımız kafesin sürekli $(u_{i,j})$ altkümesi kullanılacaktır. Ayırıştırma metodlarından biri her altkümeyi kafesin ard arda gelen satırlarının birkaçından oluşur hale getirmektir. Böylece kafe kafes satır esaslı (ROW MAJOR) bir formatta

saklandığında elemanların sürekli olan bir kümesi bir işlemci için atanır.

Senkronize algoritma $(r-1)$. iterasyondan yararlanarak (r) . iterasyon değerini bölgesel olarak aşağıdaki formülden yararlanarak hesaplamaktan ibarettir.

$$(r). \text{ iterasyon değeri} = U_{ij}^{(r)}$$

$$(r-1). \text{ iterasyon değeri} = U_{ij}^{(r-1)}$$

$$U_{ij}^{(r)} = \frac{1}{4} \left(b_{ij} + U_{i+1,j}^{(r-1)} + U_{i-1,j}^{(r-1)} + U_{i,j+1}^{(r-1)} + U_{i,j-1}^{(r-1)} \right)$$

Bu hesaplamalar sırasında tüm işlemler local olarak kendi iterasyonları sırasındaki hesaplamaları tamamladıkları zaman diğer tüm işlemlerdeki değişkenlerde güncelleştirilir. Bu işlem $(r+1)$. iterasyonun ilk değerini almasından (senkronizasyon işleminden) sonra olur. Bu işlem tüm iterasyonlar kabul edilebilir bir toleransla birleşecekleri zamana kadar devam eder. Problemin çözüm zamanı hem algoritmanın hızına hemde ard arda gelen iterasyonların birleşme hızına bağlıdır.

Genelde iteratif bir algoritma iteratif bir fonksiyonla tariflenebilir.

$$x_{i+1} = \phi(x_i, x_{i-1}, x_{i-2}, \dots, x_{i-d+1})$$

Yukarıdaki iteratif formülle tarif edilebilecek problemlerin sayısı pek çoktur. Bunlar arasında Diferansiyel Denklemler çözmek için metodlar, Lineer ve Nonlineer Denklemler

Sistemleri çözümü, fonksiyonların ekstreme noktalarını bulan algoritmalar vardır. İteratif algoritmaların çoklu işlemcili sistemlerde uygulanması sırasında ayrıştırma için kullanılan bir yaklaşım 'İterasyon fonksiyonu içindeki paralellikten yararlanmaktır'. F_i fonksiyonunun tüm kısımları arasında birbiri ile aynı büyüklükte en azından aynı komplekslikte bir küme paralel iş (TASK) elde edilmeye çalışılır. Böylece bu işlerin yürütme zamanı bağımsız ve eşit olarak dağılmış (identically distributed) olur. Örneğin lineer denklem sistemi (LSE) çözümünde

$$X_{i+1} = \Phi(X_i, X_{i-1}, \dots, X_{i-d+1}) \quad \text{denklemi}$$

$$X_{i+1} = AX_i + b \quad \text{'ye indirgenmiş olur.}$$

Eğer sistem n boyutuna sahipse biz p adet işlemciyi işimizde kullanabiliriz. Bunlardan her biri n/p (tam bölünür) adet bilinmeyen iterasyonu güncelleştirir. İteratif algoritmaların bu şekilde ayrıştırılması işlemine Dikey Ayrıştırma (VERTICAL DECOMPOSITION) adı verilir. Bu durumda eğer A matrisi sparse değilse bütün işler eşit stochastic değerlere sahiptir. Bazı durumlarda X 'in iterasyonları veya B matrisinin elemanları 'Null' olduğundan her Task'ın işlem görme zamanı RANDOM değişkendir. Örneğin aşağıdaki formülün iterasyonlarında f ve f' 'nin x deki değerinin hesabı paralel yapılabilir. Ve aşağıdaki ifadede $X(i+1)$ vektörünün tüm bileşenleri eşzamanlı olarak hesaplanabilir.

$$X(i+1) = A * X(i) + B$$

Yukarıdaki ifedeyi çoklu işlemcili bir sistemde gerçekleştirmenin bir başka yolu da belirsizlik ve kararsızlıkları işlemin hızı yararına kullanmaktır. Bu durumda birden fazla işlem aynı sonucu hesaplamak için paralel olarak çalışır. Donuca ilk erişen işlemin ortalama

zamandan daha az zaman aldığı kabul edilir.

Bundan sonraki açıklamalarda monoton olarak artan sürekli bir $f(x)$ fonksiyonunun sıfırlarını bulma problemi 'Senkronize Algoritma' olarak tartışılacaktır. Bitiş noktasında $f(x)$ in ters işaret aldığı düşünülmüştür. Bu noktalar l ve u dur. Öyleki aralık $ABS(u-l)$ olarak

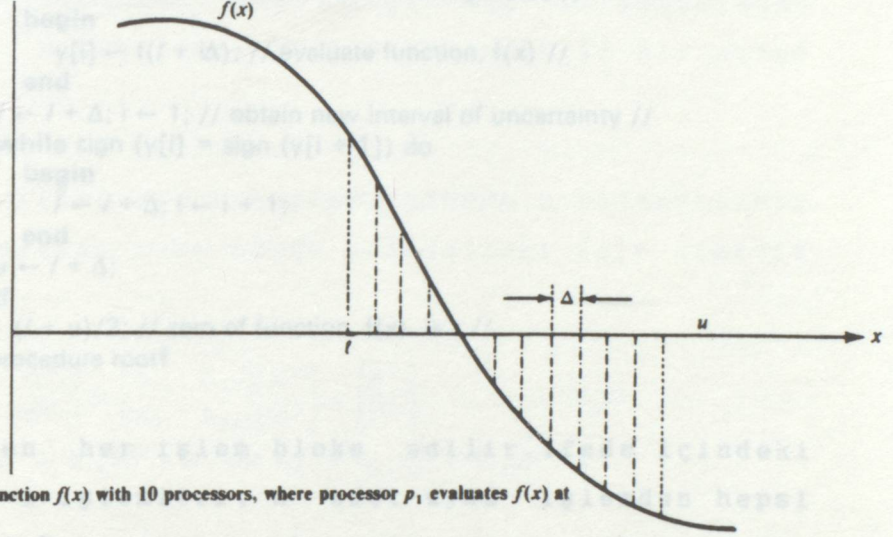


Figure 8.28 Finding the zero of a function $f(x)$ with 10 processors, where processor p_i evaluates $f(x)$ at $x = l + i\Delta$. $\Delta = |u - l|/(n + 1)$.

belirlenmiştir. İşlem $ABS(u-l) < \epsilon$ olduğu zaman bitecektir ki bu izin verilebilir bir hatadır. Şu da belirtilmelidir ki burada verilen algoritma kolayca ayrık (discrete) f ile uyusacak şekilde düzenlenebilir. Ve böylece istenen bir verinin sıralı bir dizide aranmasında da kullanılabilir. Açıklanacak algoritma n adet uydu işlemci ve bir tane ana işlemci oluşur. Ana işlem $ABS(u-l)$ aralığını $(n+1)$ alt aralığa böler ve bunların her birini $DELTA = ABS(u-l)/(n+1)$ dir. Her uydu işlem i fonksiyonunun $X_{i+1} = l + i*DELTA$ deki değerini hesaplanabilen bu işlemin girişte bahsedilen safhalardan (STAGE) birine karşılık gelir. Tüm bu işlemler tamamlandığında ana işlem hesaplanmış fonksiyon değerini işareti değiştirmek üzere karşılaştırmayı ve belirli bir alt aralıktaki (subinterval) kökün varlığını belirler. Yukarıdaki program bu algoritmayı içermektedir. Bu algoritmanın kilit noktası uydu işlemciler arasında senkronizasyonun olmasıdır. Fonksiyonun değerini

Example 10

```

real procedure rootf(f, l, u, n)
begin
  function f;
  var  $\Delta$ , l, u, y[1:n]: shared real;
  var i: shared integer;
  while  $|u - l| > \epsilon$  do
    begin
       $\Delta \leftarrow |u - l| / (n + 1)$ ; // compute subinterval //
      parfor i=1 until n do // create n slave process //
        begin
           $y[i] \leftarrow f(l + i\Delta)$ ; // evaluate function, f(x) //
        end
      l  $\leftarrow l + \Delta$ ; i  $\leftarrow 1$ ; // obtain new interval of uncertainty //
      while sign(y[i]) = sign(y[i + 1]) do
        begin
          l  $\leftarrow l + \Delta$ ; i  $\leftarrow i + 1$ ;
        end
      u  $\leftarrow l + \Delta$ ;
    end
  z  $\leftarrow (l + u) / 2$ ; // zero of function, f(x), is z //
end procedure rootf

```

hesaplamayı bitiren her işlem bloke edilir. İfade içindeki $l = l + \Delta$ ve $i = i + 1$ işlemleri, n adet uydu işlem den hepsi fonksiyonun değerini hesaplamayı bitirmeden yürütülemezler. Her uydu işlem bloke edilmiş durumdan bir sonraki iterasyonun başlaması ile uyandırılır ve tüm uydu işlemler eşzamanlı olarak uygun, seçilebilir duruma gelirler ve işin yürütülmesini yeniden ele alırlar, yani işler devam eder. Paralel çözümün esası bu senkronizasyon politikasına ihtiyaç duyar. Verilen aralıkta bu noktada f değerini hesaplamak için gerekli zamanın yani t'nin t' ortalama değerine sahip bir random değişken olduğunu düşünelim. Aynı zamanda yeni belirsizlik aralığını hesaplamak ve durma kriterini test etmek için geçen zamanın da C' ortalama değerine sahip C random değişken olduğunu kabul edelim. Bu örnekte $t' > C_2$ olduğu ve böylece C' nün analiz sırasında ihmal edilebileceğini varsaydık. Aynı zamanda ilk değer alınması (synchronization primitive) için geçen zamanda ihmal edilebilir. Algoritmanın bağlı performansının hesaplanmasında şu belirtilmelidir ki tekişlemcili bir sistemde algoritma en fazla $(\log \text{ABS}(u-l))$ kadar fonksiyonun

hesabını gerektirir. Bu yüzden beklenen çalışma zamanı $(\text{Log ABS}(u-1)) * t$ kadardır.

Senkronize paralel algoritmalarda açıkça görülür ki iterasyon bir sonraki belirsizlik aralığının boyunca $n+1$ faktörü ile azaltılır. n adet uydu işlemci kullanıldığında algoritmanın çalışma süresi $(\text{Log } n+1 \text{ ABS}(u-1) * \text{lamba} n * t)$ kadardır. Hız n tabanına göre logaritma mertebesinde olduğu için büyük n değerleri için daha efektif bir metod kullanılmalıdır.

Senkronize paralel algoritmalar $\text{LAMBDA } n$ katsayısının büyük olduğu durumlarda yani büyük n değerleri için efektif değildir.

$$\begin{bmatrix} x_{i+1}^{(1)} \\ x_{i+1}^{(2)} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_i^{(1)} \\ x_i^{(2)} \end{bmatrix} + \begin{bmatrix} b^{(1)} \\ b^{(2)} \end{bmatrix} \quad (8.32)$$

where $x_{i+1}^{(1)} = A_{11}x_i^{(1)} + A_{12}x_i^{(2)} + b^{(1)}$ and $x_{i+1}^{(2)} = A_{21}x_i^{(1)} + A_{22}x_i^{(2)} + b^{(2)}$.

Bu örnek iteratif olan senkronize bir paralel algoritmadır. Burada iterasyon fonksiyonu ayrıştırılmış ve her iterasyon adımı farklı bir işlemci tarafından yürütülecek düzen kurulmuştur. Tüm bu iterasyonların sonunda işlemler senkronize edilir. Lineer Denklem Sistemi'nin çözümünü iki concurrent işlem ile düşünersek aşağıdaki eşitlik kullanılır.

Her X_i vektörünü ayrıştırmanın en doğal tekniği vektörü $n/2$ boyunda iki bölüme X_i ve X_i ye ayırmaktır. Ve bunlar aşağıdaki gibi iki paralel işlemle güncelleştirilirler. (n in 2 ye tam bölünebileceği düşünülür) Bu bir iterasyon adımıdır. Her işlem bileşenlerden yarısını güncelleştirir. Bir sonraki işlem ancak her iki yarı kendi işlemlerini güncelleştirmey,

bitirdikten sonra başlar.

bloke edilebilir. Çözüm ise her işlemcinin kendi bellek bölgesi (CRITICAL SECTION) ile çalışmasıdır. Bu şekilde her işlemcinin kendi bellek bölgesini kullanması sağlanabilir. Bu şekilde bir bloklu çalıştırma yapılabilir.

6.5 Asenkron Paralel Algoritmalar

(Asynchronous Parallel Algorithms)

Asenkron paralel algoritmada işlemler genellikle bir diğerini beklemek zorunda değildir. İletişim paylaşılan bellekte saklanan genel (Global) değişkenlerin dinamik olarak güncelleştirilmesi ile sağlanır. Bu tür algoritmalarda aynı bellek bölgesine birden fazla concurrent işlem tarafından erişilmek istendiğinde bazı sorunları ortaya çıkacağı bellidir. Bu sorunlar ileride anlatılacak metodlar ile çözümlenecektir. Bu metodlar bile değişkenlere erişim sırasında bazı gecikmelerin

Paralel algoritmaların tasarımındaki bir başka yaklaşım ise MACROPIPELINING dir.

Bu örnek basit bir pipeline derleyicidir. Değişik işlemler lexical analiz, syntax analizi, semantik analizi, optimizasyon ve kod üretme işlemi farklı işlemler olarak kabul edilir. Kaynak program giriş olarak alınır sonra yukarıda belirtilen her safhanın çıkışı diğerinin girişidir. Oluşturulan kod optimizasyon aşamasından geçirildikten sonra çıkış olarak alınır. Pipelining sonunda oluşan sonuç heterojen iken analitik olarak ayrıştırma (partitioning) sonunda oluşan sonuç homojendir.

İşlemin belli bir safhasının işlemesi için gerekli zaman random bir zaman random bir değişkendir. İniş çıkışlar işlemcilerin hızlarının ve her safhanın girişlerinin

değişken olmasına neden olur. Bir işlem bir safhanın sonunda bloke edilebilir çünkü ya senkronize algoritmda bir giriş bekleniyordur ya da asenkron bir algoritmda kritik bir bölgeye (CRITICAL SECTION) girilmiştir. Bir işlemin bloke edilme zamanı o işlemim bloke edildiği toplam zamandır. Eğer çokluişlemcili sistem heterojen ise yani işlem daha hızlı bir işlemciye gönderildiğinde çalışma zamanı daha küçük olacaktır.

Çokluişlemcili bir sistemde çalıştırılacak bir algoritma PARALELLİK PRENSİBİ'nden yararlanmak için bir küme işleme ayrıştırılmalıdır. Bu işlem için doğal olarak iki metod vardır. Statik ve dinamik ayrıştırma.

STATİK AYRISTIRMA (Static Decomposition)

İşlem kümesi ve bunlar arasındaki önemli ilişkiler işlemler yürütülmeye başlanmadan önce bilinir. Statik ayrıştırma algoritmaları işlemler arasında çok az bir iletişim olanağı ve işlemlerin adedinin azalmasını sağlar. Buna rağmen adapte edilebilme özellikleri düşüktür.

DİNAMİK AYRISTIRMA (Dynamic Decomposition)

İşlem kümesi yürütme sırasında değişir. Dinamik ayrıştırma algoritmaları işlem grafının çalışma zamanı içinde efektif olarak varyasyonlara adapte edilebilirler. Fakat tasarımları ile başa çıkmak güç olduğu gibi işlemler aras iletişim de pahalıdır. Her asenkron paralel algoritmda tüm işlemcilerin erişebileceği bir grup GLOBAL DEĞİŞKEN bulunur. Bir işlemin bir safhası tamamlandığında işlem bu değişkenlerden birini okur, değerlendirir ve henüz bitmiş olan safhadan elde edilmiş sonuçlarla birlikte global değişken kümesini düzenler. Bir

sonraki safhayı aktif duruma geçirir ve kendi işini bitirir.

Pekçok durumda mantıksal bütünlüğü sağlamak için global değişkenler üzerinde yapılan işlemler KRİTİK BÖLGE'de yapılır. Bu işlemler arası iletişimi sağlar. Bu tip algoritmalarsa Senkron algoritmalarda olduğu gibi sıkı bir bağımlılık yoktur.

ASENKRON PARALEL ALGORİTMALAR

Asenkron Paralel Algoritmaların ana karakteristiği işlem hiç bir zaman bir giriş için beklemes. Fakat bilginin o andaki global değişkenlerde olup olmamasına bağımlı olarak işlemlere devam eder. Buna rağmen şu da belirtilmelidir ki işlemlerin Kritik Bölgeye girerek zaman zaman bloke edilme olasılıkları vardır. Aşağıda asenkron iteratif bir algoritma olarak Newton Metodu ile kök bulma verilmiştir. Burada V_1, V_2, V_3 olan üç global değişken yaratılır. Bunlar sırasıyla $f(x), f'(x), x$ değerlerini taşırlar. Aşağıda verilen denklemde $(i+1)$. iterasyondan sonra

$$x_{i+1} = x_i - f'(x_i)^{-1} f(x_i) \quad \text{denkleminde} \quad (8-25)$$

$f(x), f'(x)$ ve x sırasıyla $f(x), f'(x)$ ve x olarak değiştirilirler. $f'(x)$ hesaplaması daha zor olacağından uygun bir asenkron iteratif algoritma bulunur. Algoritma P1 ve P2 gibi iki işlemten oluşur. Bunlar şu şekilde tanımlanabilirler. P2 işlemi V_1 değişkenini update ederken P1 işlemi V_2 değişkenini update eder. Aşağıda bu P1 ve P2 işlemlerinin kodlanması gösterilmektedir. Bu programdan da görüldüğü gibi bir işlem bir değişkeni update etmeyi bitirir bitirmez herhangi bir gecikme olmaksızın, konuyla ilgili değişkenlerin değerlerini kullanarak bir sonraki update işlemlerini gerçekleştirir. İterasyonlar sıralı olarak etiketlenirler. Bunlar P1'in 2. adımında işlem

Example 8.12

```

function f, f';
var  $v_1, v_2, v_3$ : shared real;
cobegin
  Process  $P_1$ : begin
    while <termination criteria S not satisfied> do
      begin
         $v_1 \leftarrow f(v_3)$ ; // step 1 of  $P_1$  //
         $v_3 \leftarrow v_3 - v_2^{-1} v_1$ ; // step 2 of  $P_1$  //
      end
    end  $P_1$ 
  Process  $P_2$ : begin
    while <termination criteria S not satisfied> do
       $v_2 \leftarrow f'(v_3)$ ; // step 1 of  $P_2$  //
    end  $P_2$ 
coend

```

görürler. Bunun yanı sıra bu algoritmalar daha önce tanımlandığı (8.25) gibi bir rekürans bağıntısı

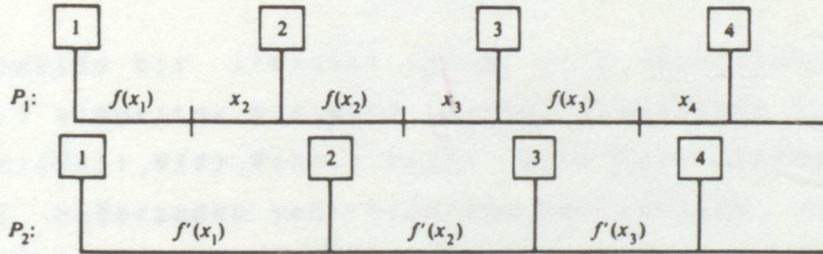


Figure 8.29 Time diagram for an asynchronous parallel algorithm.

oluşturamazlar. Örneğin ilk değerler: olarak belirli ise aşağıda verilen zamanlama diyagramı ile işlem sırasındaki hesaplamaların periyodu daha rahat anlaşılacaktır. Zamanlama diyagramında, sınırdeki i sayısı o işlemin i . iterasyonunun başlangıç noktasını gösterir. Bu durum için Yukarıdaki P_1 ve P_2 ile verilen concurrent programdan genellikle işlemin çalışması tarafından izlenen rekürans bağıntısı şeklinde olur. Bundan dolayı asenkron algoritma tarafından oluşturulan iterasyonlar ardışıl algoritma tarafından oluşturulan iterasyonlardan veya senkronize

$$V_1 = f(x_0) \quad V_2 = f'(x_0) \quad \text{ve} \quad V_3 = x_1 \quad \text{ise}$$

$$x_2 = x_1 - f'(x_0)^{-1} f(x_0)$$

$$x_3 = x_2 - f'(x_1)^{-1} f(x_2)$$

$$\vdots$$

$$x_{i+1} = x_i - f'(x_j)^{-1} f(x_i) \quad j \leq i$$

iteratif algoritmalar tarafından oluşturulan iterasyonlardan farklıdır. (x_i) lerin sırası için bir teori türetmek zordur. Bunun sebebi ise bir işlemin hızındaki belirsizliktir. Daha da ötesi iterasyonlar asenkron bir iteratif algoritma tarafından oluşturuldukları için genelde deterministik bir rekürans bağıntısı bulunamaz.

Genelde bir iteratif işlem için tasarlanan asenkron iteratif algoritma bir küme global değişkenin tanımlanması gerekir. $(V(1), V(2), V(3) \dots V(n))$ gibi. Her iterasyon adımı eski V_i değerinden yeni değerler hesaplanmak olarak kabul edilebilir. Genelde V_i lerin öyle seçilmesi gerekir ki her V_i nin update edilmesi istenen işin bir iterasyonda istenen anlamlı bir bölümünü oluştursun. Örneğin aşağıda verilen matris iterasyonunun hesaplanmak istendiğini farz edelim.

$$X(i+1) = A * X(i) + b$$

Bu sırada V_i ler vektör iterasyonu içinde eşit büyüklüğe sahip bileşenler olarak alınır. V_i leri asenkron olarak update eden işlemler aşağıdaki gibi verilebilir. $X(i)$ ve b de ' k ' adet eleman olduğunu farzedelim. Global değişkenler kümesi $(V(1), V(2), V(3) \dots V(n))$ p adet alt kümeye bölünebilir. ' k '. işlem $(V(k-1)*s+1, V(k*s))$ alt

kümesini update eder. Burada $V(j)$ X_i vektörünün j . bileşeninin o andaki değerini ifade eder. Aşağıda verilen örnek lineer denklem sistemini çözen algoritmanın gerektirdiği global değişkenler kümesinin tanımladığı bir işlemdir.

İşlem bir küme global değişkene gereksinme duyar. Bu

Example 8.13

```

var v[1:n]: shared real;
parfor k = 1 until p do
  for i = (k - 1)s + 1 until ks do
    begin
      var acc: real;
      var A[(k - 1)s + 1: ks, 1: n]: real;
      var b[(k - 1)s + 1: ks]: real;
      acc ← 0.0;
      for j = 1 until n do
        acc ← acc + A[i, j] * v[j];
      v[i] ← acc + b[i];
    end
  end
end

```

algoritmanın programı aşağıda verilmiştir. Bu asenkron iteratif algoritmada F_i iterasyon fonksiyonu içinde bir paralellik gerekir. Aynı fonksiyonda paralellik kullanmayarak işlem hızını arttırmak için bir asenkron paralel algoritma kullanmak da mümkündür. Bu algoritmalara BASİT ASENKRON İTERATİF ALGORİTMALAR denir. Ve ardışıl algoritmalar gibi her zaman aynı işlem sırasını izlerler. Aslında bu yöntem hızı işi paylaşarak değil, yürütme zamanındaki belirsizliklerden yararlanarak arttırılır. Aşağıda 'k' adet aynı işlemden $P_1, P_2, \dots, P_k$ oluşan bir basit asenkron paralel algoritma verilmiştir. Her bir hesaplamada işlem kendinden hemen önce hesaplanan iterasyon değerini kullanarak işlem yapar. Aşağıda verilen programda if ifadesi bir kritik bölge olarak kullanılmıştır. Bu tip algoritmaların en büyük avantajları

etkilemlerine sahip değildir. HEDUSA ve STAROS bu sınırları için tasarlanmıştır. Bu işlemler sistemidir. İşlemler sistemlerinin algoritmasının performansına etkisine dikkat edilmelidir. Buna neden olan Process Scheduling ve diğer diğer işlemlerin işleme girme sırasıyla sistemden gelen sorulardır. Ayrıca her bir işlemci 8-8 arasında olduğu zaman performans bir tape değeri erişir. Bunun nedeni bilgisayarın modüllerinden birinde bulunan paylaşılan verinin çok fazla referans göstermesidir.

uygulanabilir olmalarıdır. Daha çok işlem ayrıştırmalarının zor olduğu yerlerde kullanılırlar ama yine de hızları sınırlıdır. Diferansiyel Hesap (PDR)

Concurrent Quick Sort Algoritması

Bir dizinin elemanlarının sıralanmasında paralel bir algoritma kullanılması Quick Sort üzerinde tartışılacaktır. PE (Processing Elements) sıralanacak elemanlar olarak tanımlanır. Paralel algoritma değişken sayıda dinamik olarak yaratılan işlemin PE'lere verilmesinden ibarettir. İşlemciler elemanların bulunduğu dizi veya yığını paylaşırlar. Yığın henüz sıralanmamış olan sürekli altdizileri içerir. Yığına özel bir metod ile ortaklaşa erişilir. Yığının ilk değeri diziyi tanımlayan bir tanımlayıcıdır. Her yeni geçişte işlem paylaşılan yığından yeni bir altküme ve tanımlayıcı getirmeye çalışır. Eğer başarılı olursa işlem kümeyi daha küçük iki alt kümeye böler. Bu kümelerden birincisi tahmini bir ortak değerden daha küçük ikincisi ise daha büyük elemanlardan oluşur. Bu ortak değer kümenin ilk ve son elemanının ortalaması olarak

özelliklerine sahip değildir. MEDUSA ve STAROS Cm mimarisi için tasarlanmış iki ayrı işletim sistemidir. İşletim sistemlerinin algoritmanın performansına etkisine dikkat edilmelidir. Buna neden olan Pcess Scheduling ve diğer küçük işlemlerin işleme girmesiyle üstesinden gelinen sorunlardır. Ayırıştırma derecesi 6-8 arasında olduğu zaman performans bir tepe değere erişir. Bunun nedeni bilgisayarın modüllerinden birinde bulunan paylaşılan verinin çok fazla referans göstermesidir.

Asenkron Kısmi Türevli Diferansiyel Denklem (PDE)

Kısmi Türevli Diferansiyel Denklemi çözmek için tamamen paralel bir algoritma tasarlanabilir. Bu metodda her işlem her noktanın değerini update ederken paylaşılan bir alanda bulunan veriler arasından akabindeki komşusunun andaki değerini kullanır. Bu çalışma boşluğunu azaltır çünkü buffera ihtiyaç kalmaz. Oysa senkronize paralel algoritmalarda buffera ihtiyaç vardır. Eşzamanlı kullanıma karşı gelmesi gereken tek değişken henüz işini bitirmemiş işlemlerin kaydını tutan işlemdir. Bu değişkene her iterasyonda bir defa erişilir. Her işlem içinde yürütülen iterasyonların sayısını tutan bir sayıcı olmalıdır. Her işlem aynı işleri yaptıklarından bu sayıcılar işlemin bağıl hızını ölçmede mükemmel bir metoddur.

Asenkron paralel algoritma ile PDE' nin çözümü 150X150 eleman için integer sayılar ile Cm mimarisinde denenmiş ve aşağıda verilen farklı ayırıştırma dercelerinde grafikteki hızlar elde edilmiştir.

hesaplanmıştır. Bu bölümlere ve ayrıştırmadan sonra yeni alt kümelerden daha küçük olanına ait ayığına atılır. Daha sonra bunu daha büyük elemanların oluşturduğu kümenin tanımlayıcısının bir sonraki göze atılması izler. Bu procedure her alt kümedeki elemanların sayısı daha önce seçilmiş bir eşik değerden büyük olmayıncaya kadar devam eder. Bu alt küme sonra tek bir işlem içinde 'INSERTION SORT' metodu kullanılarak sıralanabilir. Bu paralel

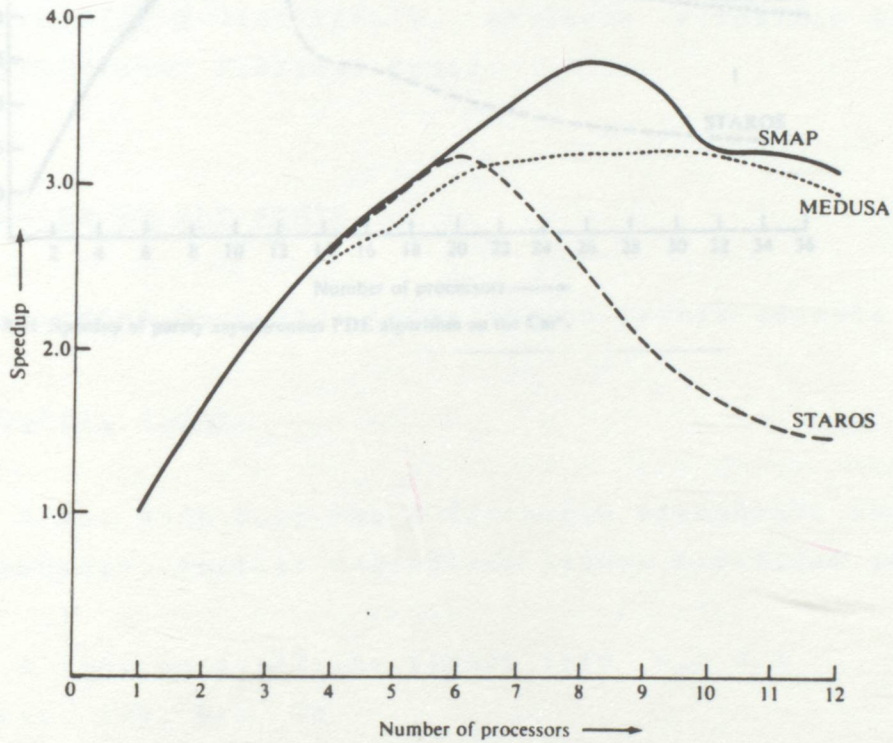


Figure 8.30 Speedup of parallel quicksort algorithm on Cm*.

algoritma Cm de 20480 elamanlı bir dizide denenmiştir. Yukarıdaki grafik üç farklı işletim sistemi ortamında algoritmanın hızını göstermektedir. Hız yapılan işlemlerin bir fonksiyonu olarak tanımlanmıştır. İşlemlerin eşik değeri 10 dur. Algoritmanın hızı tekişlemin hızına bağlıdır. Karşılaştırılan işletim sistemleri STAROS, MEDUSA ve SMAP dir.

Smep basit bir monitör programıdır ve işletim sistemi

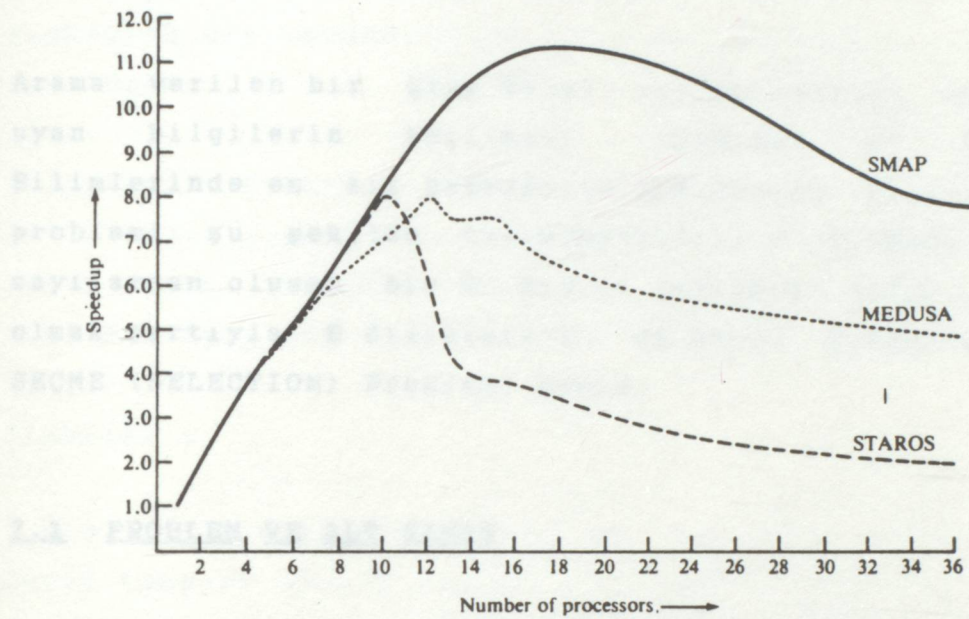
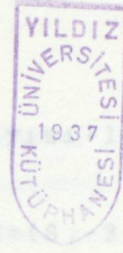


Figure 8.31 Speedup of purely asynchronous PDE algorithm on the Cm*.



SEÇME (SELECTION) PROBLEMİ

Arama verilen bir grup bilgi içinde istenen özelliklere uyan bilgilerin seçilmesi olayıdır ve Bilgisayar Bilimlerinde en sık başvurulmuş konulardan biridir. Arama problemi şu şekilde tanımlanabilir. N elemanlı ve tam sayılardan oluşan bir S dizisi verilmiş olsun, $(1 \leq k \leq N)$ olmak şartıyla S dizisinin k . en küçük elemanının bulmaya SEÇME (SELECTION) Problemi denir.

7.1 PROBLEM VE ALT SINIR

Konuya girmeden önce bazı tanımlamalar yapmak gerekir.

Lineer sıralı olmak:

$a, b, c \in A$ ve $A = (a, b, c)$ ise A dizisinin elemanları ancak ve ancak aşağıdaki şartlar sağlanırsa lineer sıralıdır denir.

- i) a, b gibi herhangi iki eleman için $a, b \in A$
 $a < b$, $a = b$, $b < a$ ve

- ii) a, b, c gibi herhangi üç eleman için $a, b, c \in A$

$a < b$ ve $b < c$ ise $a < c$ ise

RANK:

$S = (s_1, s_2, s_3, \dots, s_n)$ şeklinde lineer sıralı olarak verilen bir dizide $s(i)$ elemanının rankı



$\text{rank}(s(i)) = s(i)$ den küçük eleman sayısı + 1 olmak bir problem olmayacaktı. Fakat bir burada bulunan olarak tanımlanır. Örnek olarak S dizisi $S=(8,-3,2,-5,6,0)$ iken sıralanırsa $S=(-5,-3,0,2,6,8)$ dizisi elde edilir. $\text{rank}(0)=3$ dir çünkü -5 ve -3 olmak üzere 0 dan küçük iki sayı vardır. bir dizinin k. minimum sayısını bulmak için her elemanı en az bir defa kontrol etmek, yani en az 'n' Selection Problemi Rank kavramı kullanılırsa $S=(s_1,s_2,s_3,\dots,s_n)$ lineer sıralı dizisinin $1 \leq k \leq n$ olmak üzere rankı k olan elemanın bulunması şekline dönüşür. r reel bir sayı ise selliği etkilir.

$\text{trunc}(r)$:

r nin alt sınırı (taban), r ye eşit veya ondan küçük en büyük tamsayı olarak tanımlanır.

$\text{round}(r)$:

r nin üst sınırı (tavan), r ye eşit veya ondan büyük en küçük tamsayı olarak tanımlanır. Örnek olarak $\text{Trunc}(3.9)=3$; $\text{ROUND}(3.1)=4$; $\text{Trunc}(3.0)=\text{Round}(3.0)=3$ dür. Yani alt ve üst sınırlar sonuçta tamsayılardır.

Algoritma:

KARMASIKLIK:

Ardaşıl seçme algoritması verilen N adet elemana k. Selection probleminde k nin üç önemli değeri vardır. $k=1$, $k=n/2$ ve $k=n$. k olarak kullanılmaktadır. k sayısı ise KAMA $k=1$ için $s(k)$ S dizisinin MINIMUM elemanı $k=n$ için $s(k)$ S dizisinin MAXIMUM elemanı $k=n/2$ için $s(k)$ S dizisinin MEDYANI dir.

Ardaşıl olarak düşünülürse, içgüdüsel olarak $k=1$ ve $k=n$ için çözümün $k=n/2$ için çözümden daha kolay olduğu farkedilir. Aşıkarak başlangıçta verilen S dizisi

lineer sıralı olarak verilseydi k . elemanı bulmak hiç problem olmayacaktı. Fakat biz burada bununla ilgilenmiyoruz. Önce S dizisini sıralamak sonra da k . elemanı almak bizim aradığımız çözüm değildir. Problemden önemli bir gerçek vardır. ' k ' değerine bağlı olmaksızın, ' n ' elemanlı bir dizinin k . minimum sayısını bulmak için her elemanı en az bir defa kontrol etmek, yani en az ' n ' adet karşılaştırma yapmak lazımdır. Dolayısıyla bu, problemi çözmek için gerekli minimum ardışıl adım sayısını, Alt Sınırı verir. Selection probleminin alt sınırı; $OMEGA(n)$ dir. Bu da maliyeti etkiler.

Her alt diziyi SHELL SORT kullanılarak sıralanır ve her alt dizinin medyanı bulunur. Bu aşamada her alt dizinin bir tane medyanı bulunur. 7.2 SEÇME İÇİN ARDAŞIL BİR ALGORİTMA

Selection problemi için ardışıl algoritma ile ilgilenmemizin nedeni ileride EREW SM SIMD tipi paralel bilgisayar için tasarlayacağımız algoritmanın bu ardışıl algoritmaya dayanacak olmasıdır. Paralel algoritma ardışıl procedürü rekürsif olarak çağırarak kullanacaktır. Algoritmanın tasarımı Divide&Conquer Metodundan yararlanılır.

Algoritma:

Ardışıl seçme algoritması verilen N adet elemanın k . minimum sayısını bulmak için tasarlanmıştır. Programda eleman sayısı NN olarak kullanılmaktadır. K sayısı ise KANA seçişkeni ile ifade edilmiştir. Q sayısı seçimi ileride analiz safhasında açıklanacak olup bir alt dizinin tutabileceği en fazla eleman sayısını gösterir.

1. ADIM:

İşleme giren dizinin eleman sayısı Q değerinden küçük ise

ardaşıl seçme algoritmasının esas adımlarının işe başlamasına gerek yoktur, çünkü dizi çok küçüktür. Küçük olan bu dizi doğrudan sıralanır ve k . elemanı aradığımız elemandır. Sıralama altprogramı olarak SHELL SORT metodu kullanılmıştır. $N \geq q$ ise ana diziyi alt dizilere bölme işlemi başlar. Verilen dizi $N = !S!$ iken N/Q adet alt diziye bölünür. Bu durumda gerçekten de her dizi en fazla Q eleman tutar.

2. ADIM:

Her alt dizi SHELL SORT kullanılarak sıralanır ve her alt dizinin medyanı bulunur. Bu aşamada her alt dizinin bir tane medyanı olduğuna göre $!S!/Q$ adet medyan oluşur ve bunlar M medyanlar dizisinde saklanırlar.

3. ADIM:

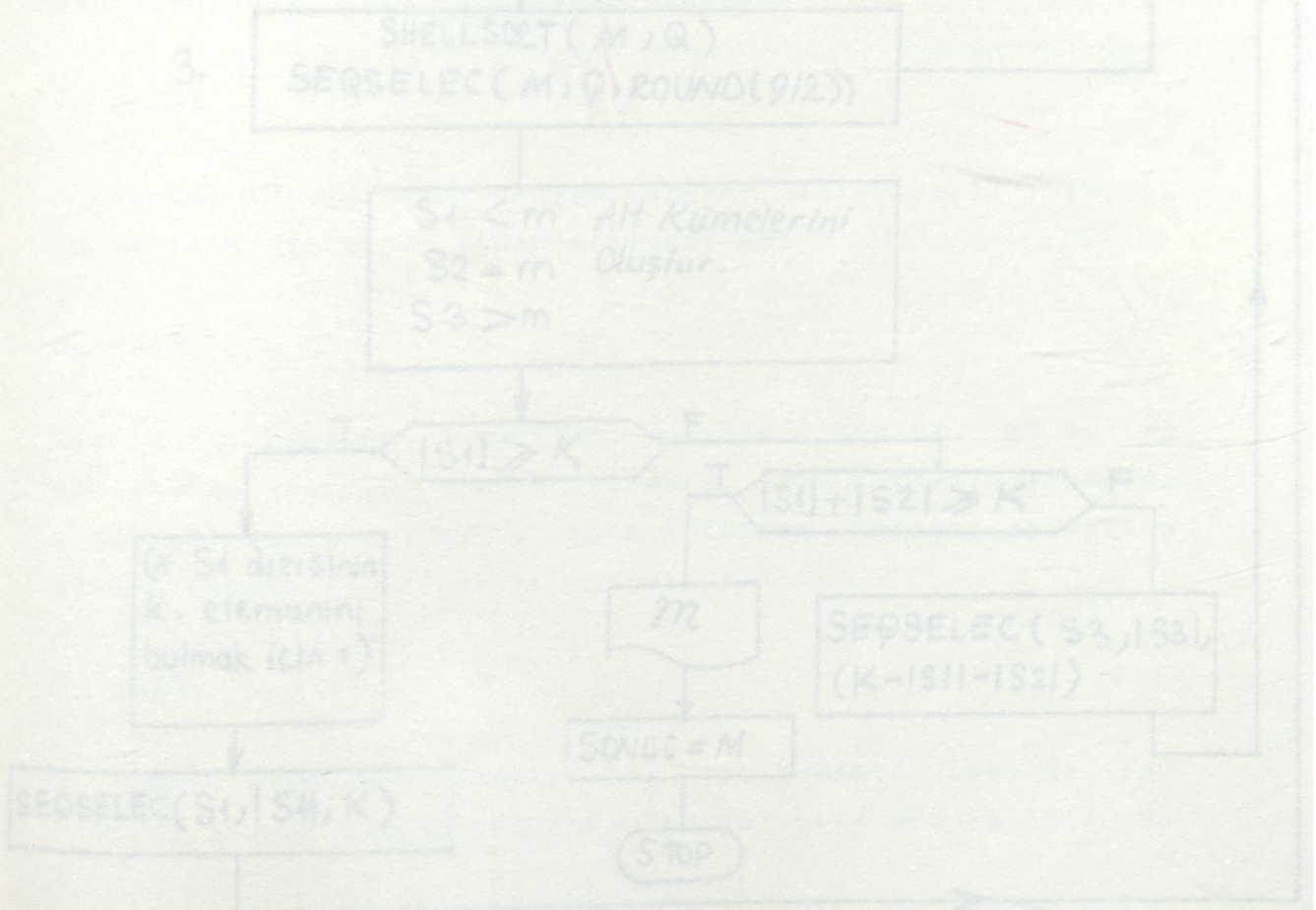
2. Adımda bulunan $!S!/Q$ adet medyanın da medyanı bulunur. Bu değer bulunmasında SEQSELEC programından rekürsif olarak faydalanılır. Bu dizinin medyanı aslında sıralı dizideki ortadaki elemandır ve $k = (\text{medyanlar dizisinin eleman sayısı})/2$ olarak verilerek medyanların medyanı bulunabilir.

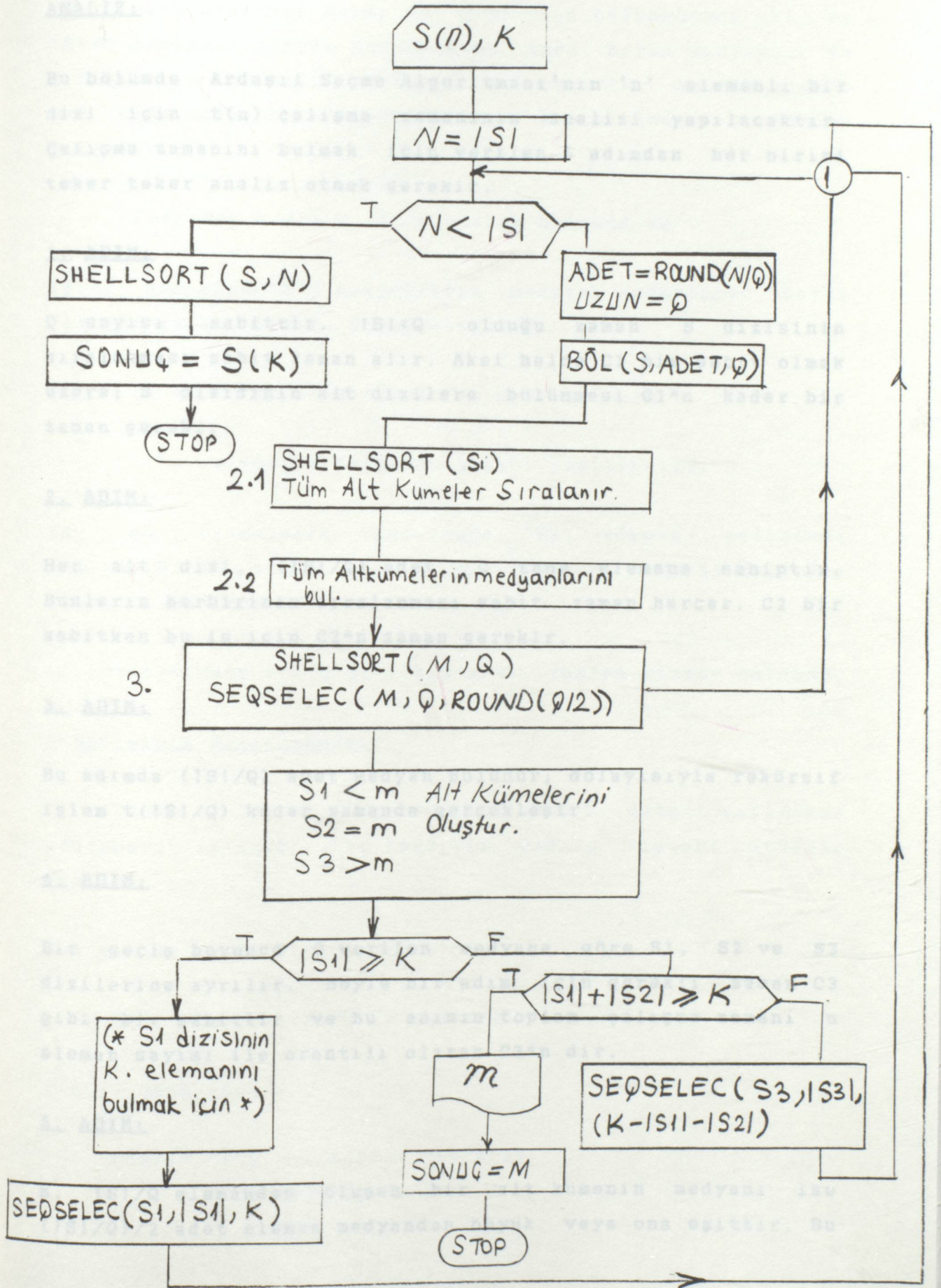
4. ADIM:

Başlangıçta verilen S dizisi medyanların medyanı (MEDMED) değişkenine göre 3 parçaya ayrılır. $S1$ dizisi, S dizisinin MEDMED değişkeninden küçük elemanlardan oluşur. $S2$ dizisi, S dizisinde MEDMED değişkenine eşit olan elemanlardan oluşur. $S3$ dizisi, S dizisinde MEDMED değişkeninden büyük elemanlardan oluşur. $S1$, $S2$ ve $S3$ dizilerinin eleman sayıları sırasıyla KUCUK, ESIT ve BUYUK değişkenlerinde saklanmaktadır.

5. ADIM:

Bu adım algoritmanın en önemli adımıdır. Yapılan kontroller sonunda $|S1| \geq k$ ise aranan elemanın $S1$ dizisi sınırları içinde olduğu farkedilir. Ardaşıl Seçme procedürü bu dizi için $SEQSELEC(S1, |S1|, k)$ olarak çağırılır. $|S1| + |S2| \geq k$ ise aranan eleman medyanın kendisidir, $S2$ dizisi içindedir. Program medyanların medyanı değerini yazarak sona erer. Bu iki şart ta doğru değilse elemanın $S3$ dizisi sınırları içinde olduğu bulunur. Ardaşıl Seçme procedürü bu dizi için $SEQSELEC(S3, |S3|, (k - |S1| - |S2|))$ olarak çağırılır. Bu şekilde procedürün rekürsif çağırıldığında hangi alt dizinin kaçınıcı küçük elemanın bulması gerektiğine karar verildikten sonra, 1. adımdan itibaren gerçekleşen işlemler yani dizi ve k değeri için bir kez daha yapılır. Eleman bulununcaya kadar işleme devam edilir. Şekil 7.1' de ARDAŞIL SEÇME (SEQUENTIAL SELECT) Algoritmasının akış diyagramı görülmektedir.





ANALİZ: Her elemanın kendi de Q parçaya bölüneceğine göre ve $IS!/Q$ birincil medyan bulunduğuna göre, birer medyan ve Bu bölümde Ardaşıl Seçme Algoritması'nın 'n' elemanlı bir dizi için $t(n)$ çalışma zamanının analizi yapılacaktır. Çalışma zamanını bulmak için verilen 5 adımdan her birini teker teker analiz etmek gerekir.

1. ADIM:

$IS!/4$ elemanın '2' medyanlarından büyük Q sayısı sabittir. $IS! < Q$ olduğu zaman S dizisinin sıralanması sabit zaman alır. Aksi halde $C1$ bir sabit olmak üzere; S dizisinin alt dizilere bölünmesi $C1*n$ kadar bir zaman gerekir.

2. ADIM:

Her alt S dizi, $(IS!/Q)$ adet, Q tane elemana sahiptir. Bunların herbirinin sıralanması sabit zaman harcar. $C2$ bir sabitken bu iş için $C2*n$ zaman gerekir.

3. ADIM:

Bu adımda $(IS!/Q)$ adet medyan bulunur; dolayısıyla rekürsif işlem $t(IS!/Q)$ kadar zamanda gerçekleşir.

4. ADIM:

Bir geçiş boyunca S verilen medyana göre $S1$, $S2$ ve $S3$ dizilerine ayrılır. Böyle bir adım için gerekli zaman $C3$ gibi bir sabittir ve bu adımın toplam çalışma zamanı n eleman sayısı ile orantılı olarak $C3*n$ dir.

5. ADIM:

m , $IS!/Q$ elemandan oluşan bir alt kümenin medyanı ise $(IS!/Q)/2$ adet eleman medyandan büyük veya ona eşittir. Bu

$!S!/Q$ adet elemanın kendi de Q parçaya bölündüğüne göre ve $!S!/Q$ birincil medyan bulunduğuna göre, birer madyandır ve Q elemanlı medyanlar kümesini oluştururlar. Bu da kümede $Q/2$ tane medyandan büyük eleman olması demektir. Bunun nedenlerle

$$!S!/(2*Q) * Q/2 = !S!/4 \text{ olarak bulunur ve}$$

$!S!/4$ elemanın 'm' medyanların medyanı değerinden büyük olacağı garantilenir. Sonuç olarak

Şimdi $!S! \leq 3*!S!/4$ ve aynı nedenle tasarlanmış bir algoritma inceleyelim. Amaç n elemanlı bir dizinin n . en büyük elemanı $!S! \leq 3*!S!/4$ hesabı yapılabilir. dizinin elemanları

Tüm bu işlemlerin sonucunda bu adımın çalışması $t(n) = 3*!S!/4$ zaman alır. Daha önceki hesaplamalardan $C4 = C1 + C2 + C3$ iken

ve PARALEL-SELECT Algoritması'nın girişleridir. $t(n) = C4*n + t(n/Q) + t(3*n/4)$ toplam zamanı bulunur. olarak gösterilirler. Her işlemci 'n' değerini yeni bir Q Sayısının Belirlenmesi:

$(n/Q + 3*n/4) < n$ şeklinde bir Q değeri seçilirse procedüre içindeki iki rekürsif çağırım işlemi devamlı azalır.

$\log_2(N) = 1 - x$ ise $x = 1 - \log_2(N)$ olarak bulunur. $Q=5$ olarak alındığını farzedelim.

Örneğin $n=25$ elemanlı bir dizi ve $N=5$ işlemci için $t(n) = C4*n + t(n/Q) + t(3*n/4)$

$$t(n) \leq C5*n \text{ ise}$$

$$\begin{aligned} t(n) &\leq C4*n + t(n/Q) + t(3*n/4) \\ &= C4*n + C5*(19*n/20) \end{aligned}$$

$$x = 0.47728$$

$C5 = 20 * C4$ cık olan $\exp(n, (1-x))$ adet işlemciden herbiri, kendi yöresel belleğinde $\exp(n, x)$ adet dizi elemanı saklar $t(n) \leq C5 * (n/20) + C5 * (19 * n/20) \cdot 47796 = 4.98999$ olarak bulunur $k = C5 * n$ da 1. adım için 5 eleman demektir. $\exp(29, (1-0.47796)) = 5.7995$ olarak bulunur bu ise birinci bulunur ki bu da $t(n) = O(n)$ demektir. Bu işlemcilerin herisi daha önce işletilen SEÇMELE programına kullanılabilir yeterince hızlıdır. N medyanlar dizisi ise SEÇME İÇİN PARALEL BİR ALGORİTMA boyutunda elemanları $N(1)$ olan bir dizedir. PARALEL SEÇME Algoritmasının akış diyagramı Şekil 7.2' de verilmiştir.

Şimdi EREW SM SIMD tipi paralel bilgisayar için tasarlanmış bir algoritma inceleyelim. Amaç n elemanlı bir dizinin k. en küçük elemanını bulmaktır. Verilen S dizisinin elemanları

1. ADIM

$S = (s_1, s_2, s_3, \dots, s_n)$ diye gösterilir. $1 \leq k \leq n$ için

S ve k PARALEL SELECT Algoritması'nın girişleridir. Algoritmada N adet işlemci kullanılır. $P_1, P_2, P_3, \dots, P_n$ olarak gösterilirler. Her işlemci 'n' değerini yani ana dizinin eleman sayısını alarak;

$N = \exp(n, (1-x))$ formülünden $0 < x < 1$ olan X sayısı hesaplanırsa

$\log n(N) = 1-x$ ise $x = 1 - \log n(N)$ olarak bulunur.

2. ADIM

Örnek olarak $n=29$ elemanlı bir dizi ve $N=5$ işlemci için olay incelenirse,

$x = 1 - \log_{29}(5)$

$x = 1 - \log_{10}(5) / \log_{10}(29)$

$x = 1 - 0.69897 / 1.46239$

$x = 0.47796$ olarak hesaplanır

Kullanılacak olan $\exp(n, (1-x))$ adet işlemciden herbiri, kendi yöresel belleğinde $\exp(n, x)$ adet dizi elemanı saklayabilir. Gerçekten de $\exp(29, 0.47796) = 4.99999$ olarak bulunur ki bu da 1. adım için 5 eleman demektir. $\exp(29, (1-0.47796)) = 5.7999$ olarak bulunur bu ise birinci adımda 5 tane işlemci kullanılacak demektir. Bu işlemcilerin hepsi daha önce anlatılan SEQSELEC programını kullanabilir yetenektedirler. M medyanlar dizisi ise paylaşılan bellekte saklanan N boyutunda elemanları $M(i)$ olan bir dizidir. PARALEL SEÇME Algoritmasının akış diyagramı Şekil 7.2' de verilmiştir.

Algoritma:

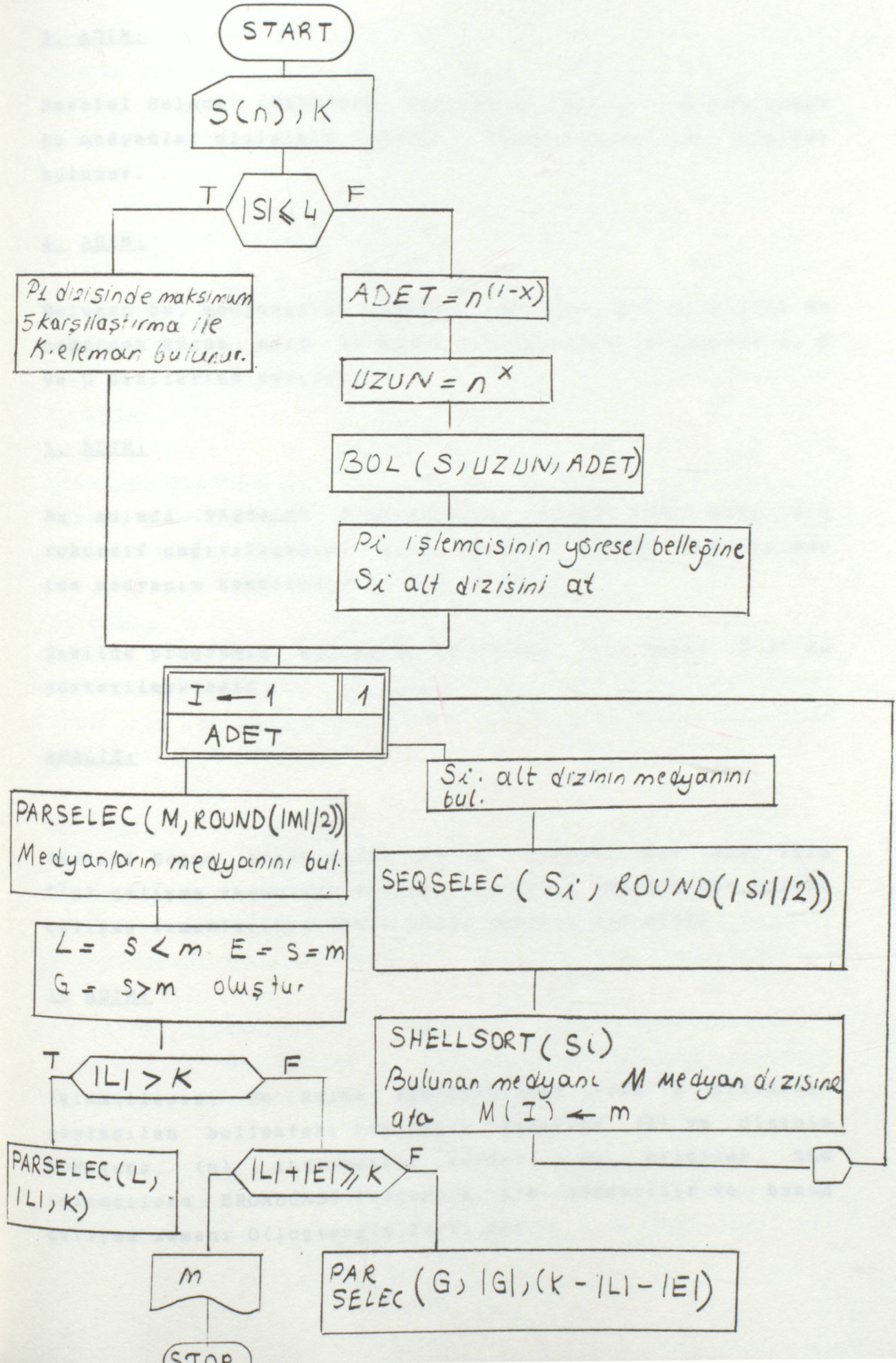
1. ADIM:

Verilen dizinin uzunluğu olan $|S| < 4$ ise dizi SEQSELEC Procedüründe olduğu gibi sadece sıralanır ve sıralı dizinin k. elemanı aranılan sayı olarak çıkışa verilir. Dizi daha büyük bir dizi ise $N = |S|$ tane eleman yukarıda verilen formülle bulunan 'x' değerinden yararlanılarak hesaplanan $ADET = \exp(N, (1-x))$ tane işlemciye dağıtılır. Her işlemci $UZUN = \exp(N, x)$ tane dizi elemanını kendi yöresel belleklerine alırlar.

2. ADIM:

İşlemcilerin hepsi paralel çalışarak kendi paylarına düşen dizilerin medyanlarını bulurlar. Bunu yapmak için SEQSELEC Procedürü ile işlemcinin alt dizisi için $k = |S|/2$. minimum elemanı bulma işini yaptırırlar. Her medyanı bulan P_i . işlemci $m(i)$ medyanını paylaşılan bellekte bulunan M medyan dizisinin i. gözüne yazar.

- PARALEL SELECT METODU -



3. ADIM: İlk P_1 işlemcisinin x elemanı doğrudan bulunur. Sabit bir zaman alır. Aksi halde P_1 işlemcisi dizinin 'A' Paralel Select (PARSELEC) Procedürü rekürsif kullanılarak bu medyanlar dizisinin $(\lfloor M \rfloor / 2)$ küçük elemanı yani medyanı bulunur.

4. ADIM:

Bulunan bu, medyanların medyanı değerine göre S dizisi bu değerden küçük, eşit ve büyük olanlar diye sırasıyla L, E ve G dizilerine ayrılır.

5. ADIM:

Bu adımda PARSELEC Procedürünün hangi yeni dizi ile rekürsif çağırılacağına karar verilir. Eleman E dizisinde ise medyanın kendisidir ve dizi biter.

Şekilde programın çalışması 29 eleman için Şekil 7.3' de gösterilmektedir.

ANALİZ:

Paralel Seçme Algoritması'nın 'n' elemanlı bir dizi için $t(n)$ çalışma zamanının analizi verilen 5 adımdan her birini çalışma zamanlarının teker teker analizi ile olur.

1. ADIM:

İşlemcilerin, bu adıma başlayabilmek için S dizisinin paylaşılan bellekteki başlangıç adresine (A) ve dizinin boyutuna (n) ihtiyaçları vardır. Bu bilgiler tüm işlemcilere BROADCAST Procedürü ile gönderilir ve bunun çalışma zamanı $O(\log(\exp(n, 1-x)))$ dir.

Eğer $n < 4$ ise P_1 işlemcisinin k . elemanı doğrudan bulması sabit bir zaman alır. Aksi halde P_i işlemcisi dizinin 'A' ve 'n' değerlerinden yararlanarak; dizinin ilk ve son değerlerini aşağıda verilen formülden hesaplar.

İlk değeri $--> A + (i-1) \cdot \exp(n, x)$

Son değeri $--> A + i \cdot \exp(n, x) - 1$

Bu işlem için bile sabit bir zaman harcanır. Sonuçta 1. adım C_1 bir sabit olmak üzere $C_1 \cdot \log(n)$ zamanında biter.

2. ADIM:

SEQSELEC Procedürü daha önce de incelendiği gibi $\exp(n, x)$ uzunluğundaki bir dizinin medyanını $C_2 \cdot \exp(n, x)$ kadar zamanda bulur.

3. ADIM:

Bu adımda PARSELEC Procedürü $\exp(n, 1-x)$ boyutundaki bir dizi ile çağrıldığı için 3. adım $t(\exp(n, 1-x))$ zamanında çalışır.

4. ADIM:

Üç farklı işlemden oluşur.

i) m değerinin BROADCAST Procedürü ile tüm işlemcilere gönderilmesi $O(\log(\exp(n, 1-x)))$ zamanında gerçekleşir.

ii) P_i işlemcisinin m değerine göre L , E , G dizilerini oluşturması kullanılan S_i alt dizisinin eleman sayısına bağlı olup çalışma zamanı $O(\log(\exp(n, x)))$ mertebesindedir.

iii) L_i , E_i ve G_i her işlemcinin medyan değerine göre oluşturdukları dizilerdir. L , E , G dizilerini oluşturmak için merge edilirler. i . işlemcinin L dizisi (L_i) nin

uzunluğu a_i ise $a_i = !L_i!$ olarak ifade edilir ve

$1 \leq i \leq \exp(n, 1-x)$ olmak şartıyla Z_i aşağıdaki ifade ile hesaplanır.

$$Z(i) = \frac{c(n)}{O(\log(\exp(n, 1-x)))} >$$

Tüm bu toplamalar $\exp(n, 1-x)$ adet işlemcide ALLSUM Procedürü kullanılarak $O(\log(\exp(n, 1-x)))$ zamanında gerçekleşir. $Z(0)=0$ olsun, tüm işlemciler paralel olarak kendi alt dizilerini L dizisini oluşturmak üzere merge ederler. P_i işlemcisi L_i dizisini $Z(i-1)+1$ adresinden başlamak üzere kopyalar. Bu işlem $O(\log(\exp(n, x)))$ zamanında gerçekleşir. Sonuç olarak 4. adımın çalışma zamanı $C3 \cdot \exp(n, x)$ dir.

5. ADIM:

Bu adımda artık L dizisinin eleman sayısı 4. adım sonunda elde edilen eleman sayısıdır. Aynı durum E ve G dizileri için de geçerlidir. m, M dizisinin medyanı olduğundan S 'in $\exp(n, 1-x)$ tane elemanının $'m'$ den büyük olması garantidir. Hatta M 'in her elemanı (her alt dizi gözönünde bulundurulursa) S dizisinin en az $\exp(n, x)/2$ elemanından küçüktür. Sonuçta $|L| < 3 \cdot n/4$ olur benzer olarak $|G| < 3 \cdot n/4$ değeri de aynıdır. O halde 5. adım $t(3 \cdot n/4)$ kadar zamanda

Yukarıda ayrıntılı olarak açıklanan bu 5 adımın sonuçlarını değerlendirirsek; Paralel Seçme Algoritmasının toplam çalışma zamanı $t(n)$ şöyle bulunur.

$$t(n) = C1 \cdot \log(n) + t(\exp(n, 1-x)) + C3 \cdot \exp(n, x) + t(3 \cdot n/4)$$

toplamından $n > 4$ için

$t(n) = O(\exp(n, x))$ olarak bulunur.

Kullanılan işlemci sayısı $p(n) = \exp(n, 1-x)$ olduğundan algoritmanın maliyeti $c(n) = t(n) * p(n)$ formülünden

$$c(n) = \exp(n, 1-x) * O(\exp(n, x))$$

$$c(n) = O(n) \text{ olarak bulunur.}$$

Bu değer $OMEGA(n)$ alt sınırı göz önünde bulundurulursa fiyat optimal bir algoritmadır. $\exp(n, x)$ asimptotik olarak her x değeri için $\log(n)$ den daha büyüktür.

$N = \exp(n, 1-x)$ ve $n/\exp(n, x) < n/\log(n)$ olduğundan Paralel Seçme Algoritması $N < n/\log(n)$ olması şartıyla fiyat optimaldir sonucuna varılır.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
S	3	14	16	20	8	31	22	12	33	1	4	9	10	5	13	7	24	2	14	26	18	34	36	25	14	27	32	35	33

(a)

3	14	16	20	8	31	22	12	33	1	4	9	10	5	13	7	24	2	14	26	18	34	36	25	14	27	32	35	33
---	----	----	----	---	----	----	----	----	---	---	---	----	---	----	---	----	---	----	----	----	----	----	----	----	----	----	----	----

(b)

M	14	9	7	25	32
---	----	---	---	----	----

(c)

L											E		G															
3	8	12	1	4	9	10	5	13	7	2	14	14	14	16	20	31	22	33	24	26	18	34	36	25	27	32	35	33

(d)

16	20	31	22	33
----	----	----	----	----

24	26	18	34	36
----	----	----	----	----

25	27	32	35	33
----	----	----	----	----

(e)

M	22	26	32
---	----	----	----

(f)

The diagram illustrates a sequence of elements grouped into three sections: L, E, and G. The elements are represented as a sequence of boxes, with vertical lines indicating the boundaries between the sections. Section L contains the first 7 elements, section E contains the next 2 elements, and section G contains the final 7 elements.

16	20	22	24	18	25	26	31	33	34	36	27	32	35	33
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

(g)



SIRALAMA

Sıralama konusu Bilgisayar Bilimlerinin Algoritma dalında en çok dikkat çeken konularından biridir. Bu ilginin iki nedeni hemen açıklayabiliriz. Sıralama pratikte bu olayla ilgilenenler için önemli bir konudur çünkü bilgisayarla yapılan zor işlemlerden biridir. Aynı zamanda zengin bir araştırma konusudur. Yani hem pratik hem de teorik olarak, sıralama üzerinde çok çalışılan bir konudur.

8.1 Problem ve Alt Sınır

Bir A dizisinin elemanlarına küçükten büyüğe sıralı denebilmesi için

$a, b, c \in A$ ve $A = (a, b, c)$ ise A dizisinin elemanları ancak ve ancak aşağıdaki şartlar sağlanırsa lineer sıralıdır denir.

- i) a, b gibi herhangi iki eleman için $a, b \in A$ ve $a < b$, $a = b$, veya $b < a$ olması
- ii) a, b, c gibi herhangi üç eleman için $a, b, c \in A$ ve $a < b$ ve $b < c$ iken $a < c$ olması gerekir.

Tanım: Verilen $S = (s_1, s_2, s_3, \dots, s_n)$ dizisinin sırsılaşmasının amacı $s''(i) < s''(i+1)$ $i=1, 2, 3, \dots, n-1$ şartını sağlıyorsa $S'' = (s_1, s_2, s_3, \dots, s_n)$ dizisini elde etmektir.

Şimdi analiz kısmında enstitülen kavramlara göre sırsılaşma Probleminin alt ve üst sınırlarını inceleyelim.

'n' elemanlı bir diziyi sıralama probleminin alt sınırı $OMEGA(n) = n \cdot \log(n)$ dir. Bu teorem bize asimptotik olarak, en kötü durumda diziyi sıralamak için $n \cdot \log(n)$ ' in sabit bir sayı ile çarpımı kadar adım gerektiğini anlatır. Pekçok seri sıralama algoritması vardır ama bunlardan Quicksort, Merge Sort, Shell Sort optimal dir ve çalışma zamanları $n \cdot \log(n)$ mertebesinde dir. Bu alt sınıra da uyar. Sıralama problemini çözerken kullanılacak algoritma ister seri olsun ister paralel aşağıdaki iki kavramı gözden geçirmek işleri kolaylaştıracaktır.

'n' Adet İşlemci İçin Alt Sınır

n elemanlı bir diziyi sıralamak için n adet işlemci kullanıldığını düşünelim. Açık olarak bellidir ki en kötü durumda, hiç bir algoritma $\log(n)$ mertebesinde daha az bir mertebeden adımıyla sıralayamaz. Aksi düşünülürse en kötü durumda algoritmanın maliyeti $n \cdot \log(n)$ mertebesinde daha küçük olacaktır ki bu da alt sınır teoremine karşı düşer.

Ardasıl Giriş Ve Çıkış İçin Alt Sınır

Paralel bilgisayarın, dizi elemanlarını ardaşıl olarak okuduğunu veya sonuçları dışarı ardaşıl olarak verdiğini; yani bir birim zamanda bir veri ile giriş veya çıkış yaptığını düşünelim. Böyle bir bilgisayar için tasarlanmış hiç bir algoritma 'n' adet elemanı 'n' adımdan daha az adımda sıralayamaz. Buna dikkat etmekte yarar vardır çünkü çoğu algoritmanın analizinde giriş, çıkış işlemleri gözönüne alınmaz. Oysa burada incelenen algoritmalarda bu konuya da dikkat edilmiş ve algoritmanın analizinde giriş, çıkış işlemlerinin etkisi de hesaplanmıştır.

8.2 Ardaşıl Quick Sort Algoritması

Quick sort seri sıralama yapan optimal seri algoritmalarından biridir. Rekürsif Quick Sort algoritmasının programı Şekil 8.1' de verilmiştir.

QUICK SORT PROGRAMI

procedure QUICKSORT (S)

if $|S| = 2$ and $s_2 < s_1$

then $s_1 \leftrightarrow s_2$

else if $|S| > 2$ then

(1) {Determine m , the median element of S }

SEQUENTIAL SELECT (S, $\lceil |S|/2 \rceil$)

(2) {Split S into two subsequences S_1 and S_2 }

(2.1) $S_1 \leftarrow \{s_i: s_i \leq m\}$ and $|S_1| = \lceil |S|/2 \rceil$

(2.2) $S_2 \leftarrow \{s_i: s_i \geq m\}$ and $|S_2| = \lfloor |S|/2 \rfloor$

(3) QUICKSORT(S_1)

(4) QUICKSORT(S_2)

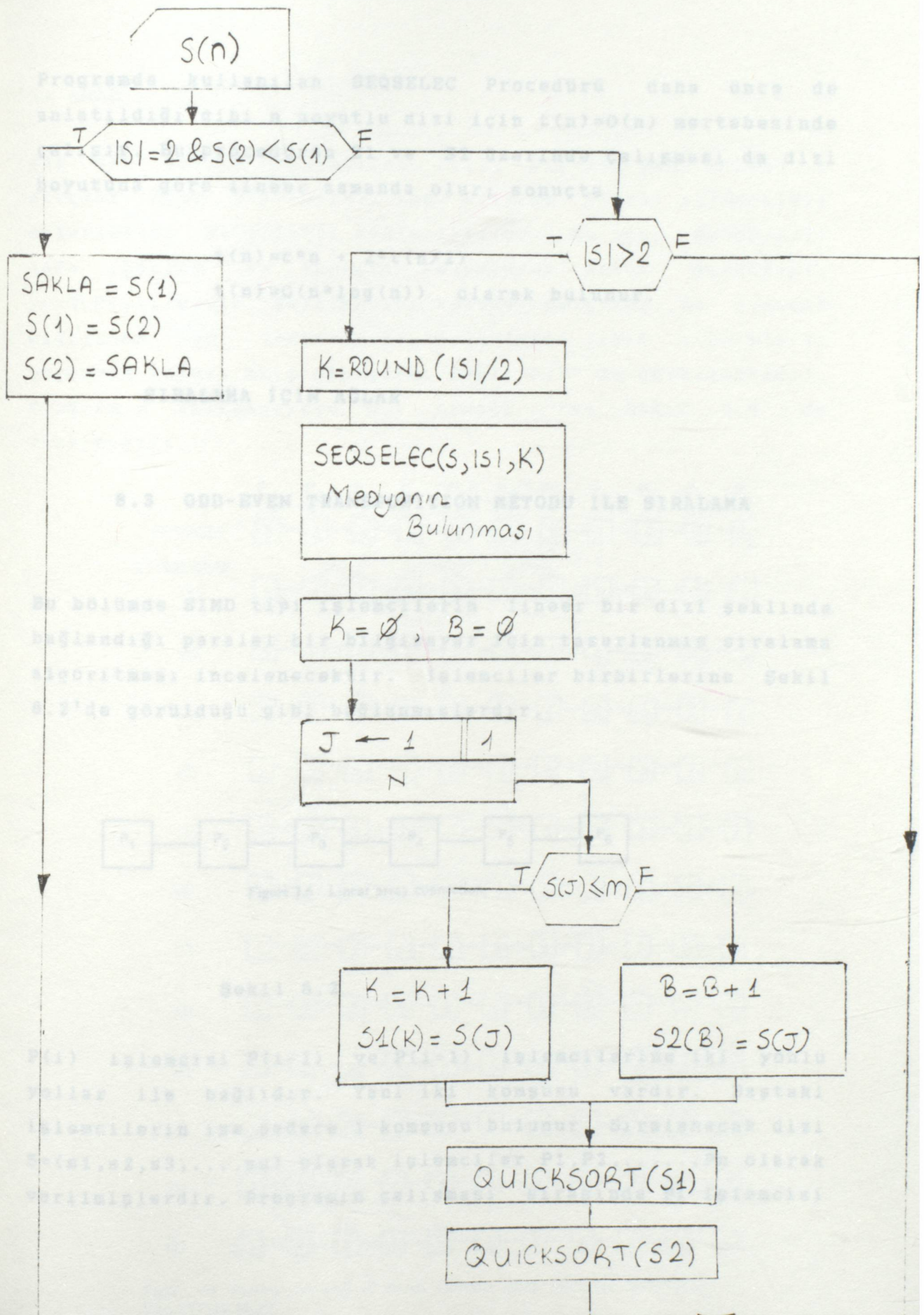
end if

end if. $\square$

Şekil 8.1

Her rekürsif dönüşte QUICKSORT Altprogramı üzerinde çalıştığı dizinin medyanını bularak; bu medyana göre diziyi küçük ve büyük olanlar diye iki parçaya ayırır. Bunlar sırasıyla S_1 ve S_2 dizileridir. Sonra program bu alt dizileri tekrar medyandan küçük ve büyük olanlar diye iki parçaya ayırır. Bu ayırma işlemi dizilerin boyutu 1 veya 2 oluncaya kadar devam eder. Programın hemen başındaki boyut kontrolünde dizinin yeteri kadar küçüldüğü anlaşılınca dizideki iki eleman karşılaştırılır ve gerekirse karşılıklı yer değiştirilir. Burada $|S_1| = \text{ROUND}(|S|/2)$ iken ikinci dizinin boyutunun $|S_2| = \text{TRUNC}(|S|/2)$ olmalıdır.

QUICK SORT METODU



ANALİZ:

Programda kullanılan SEQSELEC Procedürü daha önce de anlatıldığı gibi n boyutlu dizi için $t(n)=O(n)$ mertebesinde çalışır. Bu procedürün S1 ve S2 üzerinde çalışması da dizi boyutuna göre lineer zamanda olur; sonuçta

$$t(n)=c*n + 2*t(n/2)$$

$$t(n)=O(n*\log(n)) \text{ olarak bulunur.}$$

SIRALAMA İÇİN AÇLAR**8.3 ODD-EVEN TRANSPOSITION METODU İLE SIRALAMA**

Bu bölümde SIMD tipi işlemcilerin lineer bir dizi şeklinde bağlandığı paralel bir bilgisayar için tasarlanmış sıralama algoritması incelenecektir. İşlemciler birbirlerine Şekil 8.2'de görüldüğü gibi bağlanmışlardır.

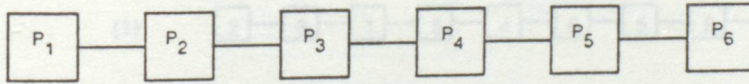


Figure 1.6 Linear array connection.

Şekil 8.2

$P(i)$ işlemcisi $P(i-1)$ ve $P(i+1)$ işlemcilerine iki yönlü yollar ile bağlıdır. Yani iki komşusu vardır. Baştaki işlemcilerin ise sadece 1 komşusu bulunur. Sıralanacak dizi $S=(s_1, s_2, s_3, \dots, s_n)$ olarak işlemciler $P_1, P_2, \dots, P_n$ olarak verilmişlerdir. Programın çalışması sırasında P_i işlemcisi

verilen dizinin $S(i)$. elemanını içerir. Algoritma tekrar tekrar işleme giren 2 adımdan oluşur.

1. ADIM

Tüm tek sayılı işlemciler $P(i)$, $P(i+1)$ den $S(i+1)$ i alırlar. Eğer $S(i) > S(i+1)$ ise $P(i)$ ve $P(i+1)$ işlemcileri değerlerini karşılıklı değiştirirler. Bu adım $ROUND(n/2)$ defa yapılır ve sonuçta elemanlar artık karşılıklı yerdeğiştirmenin gerelmediği konuma gelirler. Bu nedenle algoritma tüm $1 \leq i \leq (n-1)$ değerleri için $x(i) < x(i+1)$ şartıyla biter. Akış diyagramı Şekil 8.3' de görülmektedir. Programın çalışmasının bir örneği ise Şekil 8.4' de verilmiştir.

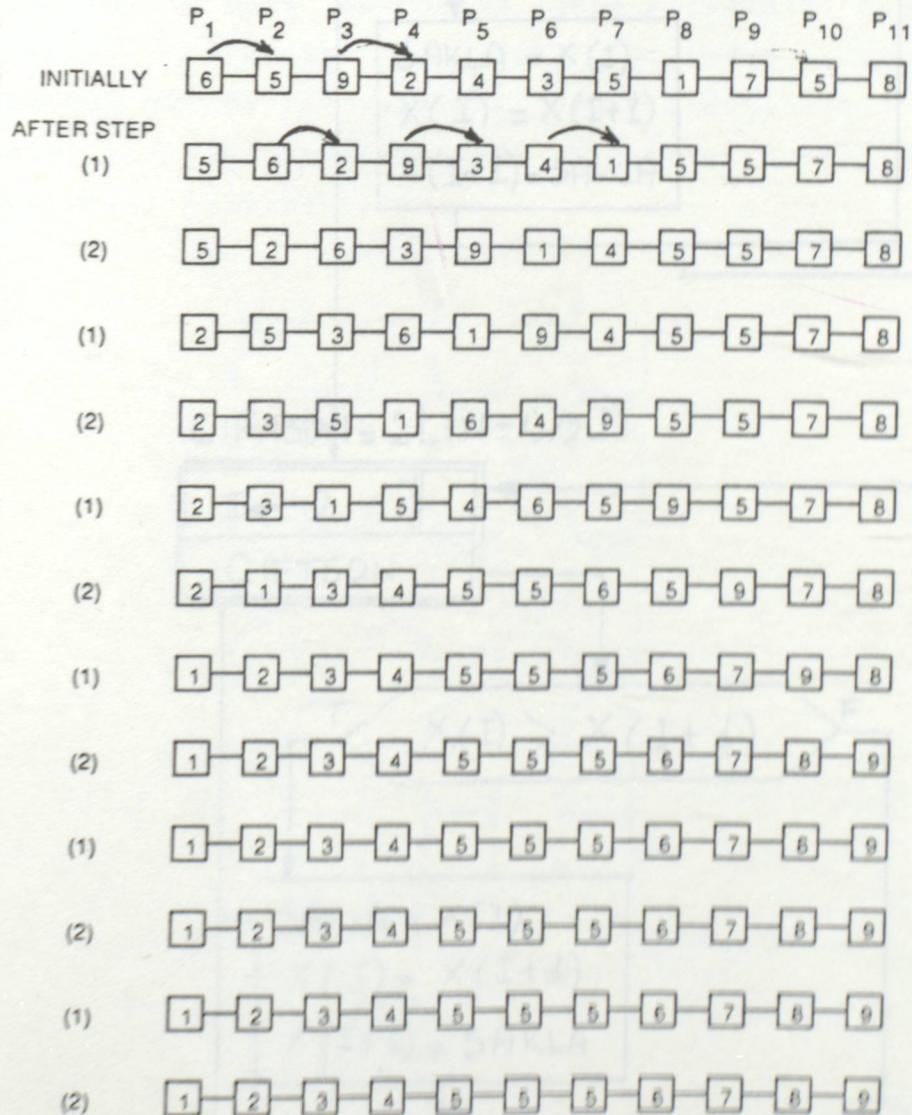
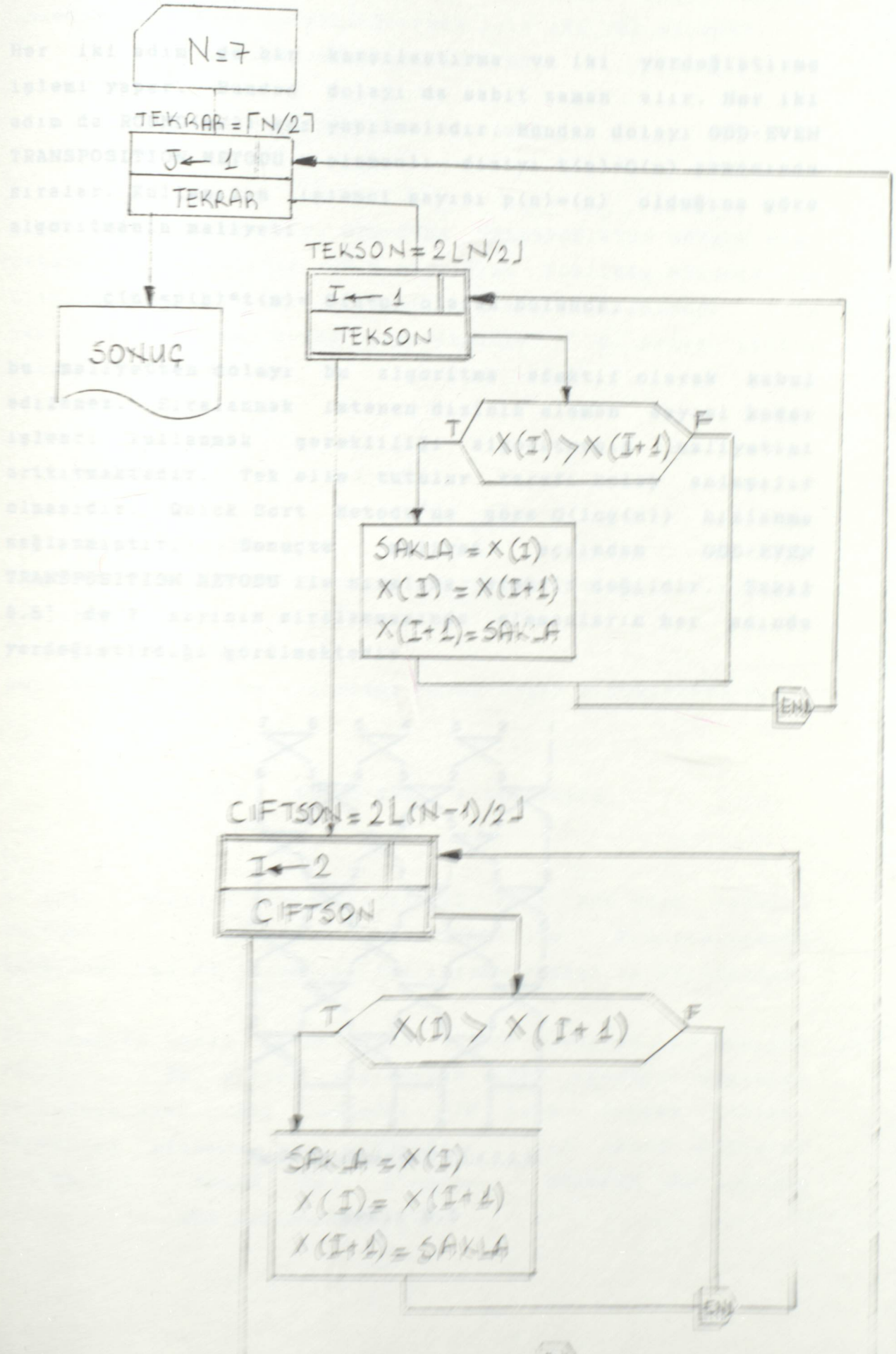


Figure 4.2 Sorting sequence of eleven elements using procedure ODD-EVEN TRANSPOSITION.

-SIRALAMA' da ODD-EVEN TRANSPOSITION METODU-



ANALİZ

Her iki adım da bir karşılaştırma ve iki yerdeğiştirme işlemi yapar. Bundan dolayı da sabit zaman alır. Her iki adım da $\text{ROUND}(n/2)$ kez yapılmalıdır. Bundan dolayı ODD-EVEN TRANSPOSITION METODU n elemanlı diziyi $t(n)=O(n)$ zamanında sıralar. Kullanılan işlemci sayısı $p(n)=(n)$ olduğuna göre algoritmanın maliyeti

$c(n)=p(n)*t(n)=O(n*n)$ olarak bulunur.

bu maliyetten dolayı bu algoritma efektif olarak kabul edilemez. Sıralanmak istenen dizinin eleman sayısı kadar işlemci kullanmak gerekliliği algoritmanın maliyetini arttırmaktadır. Tek elle tutulur tarafı kolay anlaşılır olmasıdır. Quick Sort Metodu'na göre $O(\log(n))$ hızlanma sağlanmıştır. Sonuçta maliyet açısından ODD-EVEN TRANSPOSITION METODU ile sıralama efektif değildir. Şekil 8.5'de 7 sayının sıralanmasında elemanların her adında yerdeğiştirdiği görülmektedir.

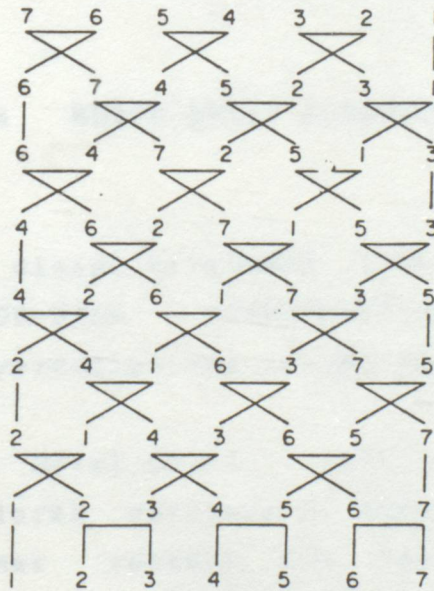


Fig. 3.2 Sort diagram for (7, 6, 5, 4, 3, 2, 1).

şimdi optimal fiyatlı bir sıralama algoritmasını inceleyelim. Bunu gerçekleştirmek için iki yol görünür.

1- Çalışma zamanını düşürmek.

2- İşlemci sayısını azaltmak.

Bitinci yol ümitsizdir. ODD-EVEN TRANSPOSITION METODU $O(n)$ zamanda çalışmaktadır ve lineer dizi şeklinde düzenlenmiş bilgisayarda n n elemanlı diziyi sıralamak için yazılabilecek en hızlı algoritmadır. O halde ikinci yaklaşımı deneyelim;

$$p(n) = N \quad N < n$$

$$c(n) = p(n) * t(n)$$

= $N * t(n)$ durumu oluşursa maliyet düşecektir.

Bunu gerçekleştiren algoritma MERGE-SPLIT ALGORİTMASI dır.

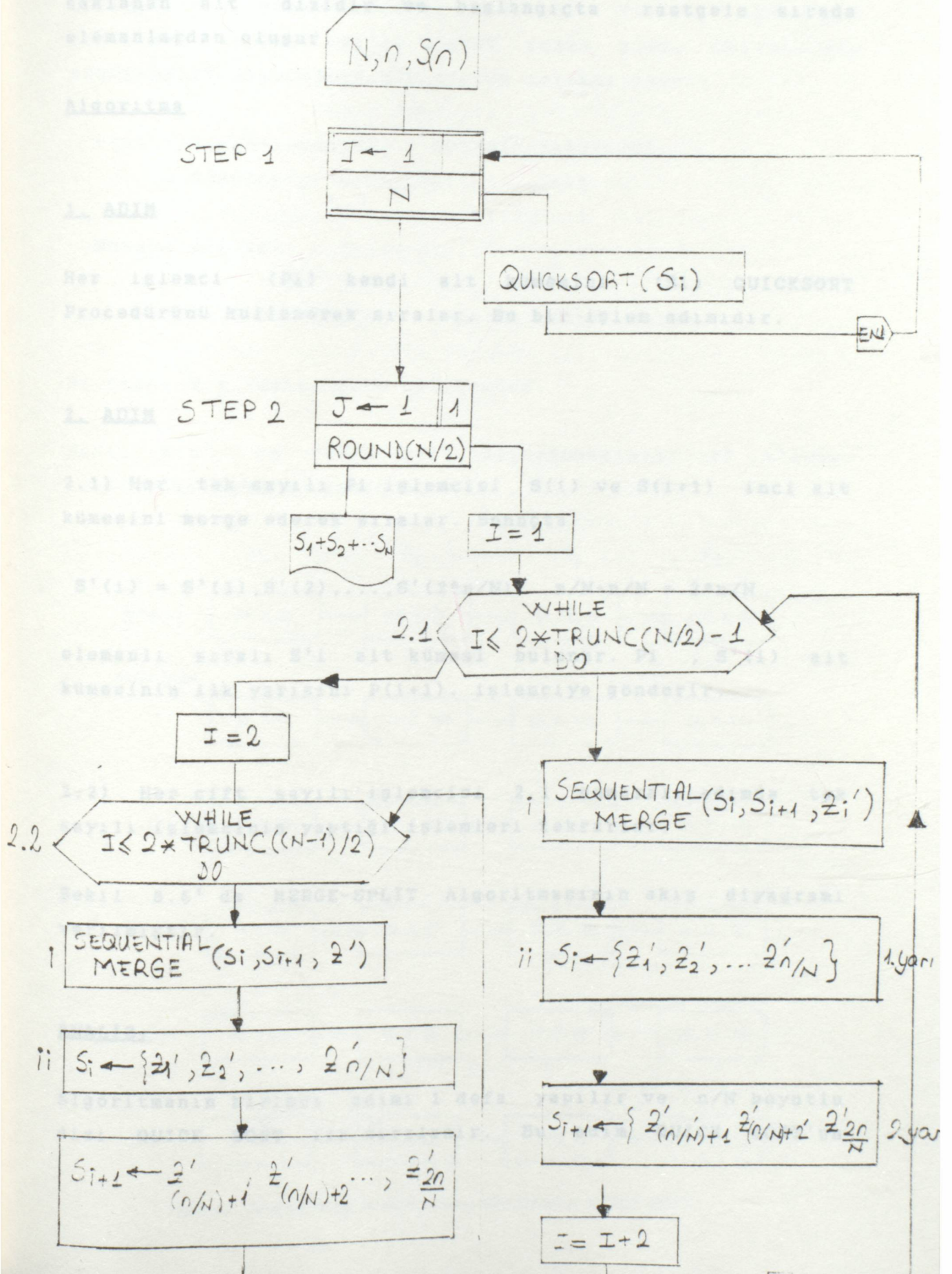
8.4 MERGE-SPLIT ALGORİTMASI

n adet elemanlı diziyi sıralamak için N/n adet işlemci kullanılır. ODD-EVEN TRANSPOSITION Procedüründeki karşılaştırma ve yerdeğiştirme işlemi burada da kullanılır.

Sıralanacak dizi $S=(s_1, s_2, s_3, \dots, s_n)$ olarak işlemciler $P_1, P_2, \dots, P_n$ olarak verilmiş N adet işlemci arasında paylaştırılır. Her işlemci n/N tane eleman saklar. Gerekirse bölümün tam olabilmesi için sahte değerler kullanılır. $S=n/N$ her işlemcinin yöresel belleğinde sakladığı eleman sayısıdır.

— SIRALAMA'da MERGE-SPLIT ALGORİTMASI —

N : Processor Sayısı n : Dizinin Eleman Sayısı $S(n)$: dizi



ODD-EVEN TRANSPOSITION Procedürü bu defa S boyutlu bu alt dizilerde çalıştırılır. S_i dizisi P_i işlemcisi tarafından saklanan alt dizidir ve başlangıçta rastgele sırada elemanlardan oluşur.

Algoritma

1. ADIM

Her işlemci (P_i) kendi alt kümesini (S_i) QUICKSORT Procedürünü kullanarak sıralar. Bu bir işlem adımıdır.

2. ADIM

2.1) Her tek sayılı P_i işlemcisi $S(i)$ ve $S(i+1)$ inci alt kümesini merge ederek sıralar. Sonuçta

$$S'(i) = S'(1), S'(2), \dots, S'(2*n/N) \quad n/N + n/N = 2*n/N$$

elemanlı sıralı S'_i alt kümesi bulunur. P_i , S'_i alt kümesinin ilk yarısını $P(i+1)$ işlemciye gönderir.

2.2) Her çift sayılı işlemcisi 2.1 numaralı adımda tek sayılı işlemcinin yaptığı işlemleri tekrarlar.

Şekil 8.6' da MERGE-SPLIT Algoritmasının akış diyagramı verilmiştir.

ANALİZ:

Algoritmanın birinci adımı 1 defa yapılır ve n/N boyutlu dizi QUICK SORT ile sıralanır. Bu adım QUICK SORT'un

karmaşıklıkından dolayı $O(n/N \cdot \log(n/N))$ zamanda yapılır. $S(i+1)$ alt dizisini P_i işlemciye almak, ardışıl MERGE ile birleştirmek sonuc $S(i+1)$ dizisini $P(i+1)$ işlemciye göndermek toplam toplam $O(n/N)$ zaman alır. Dolayısıyla MERGE-SPLIT algoritmasının toplam çalışma süresi

$t(n) = O(n/N \cdot \log(n/N)) + \text{ROUND}(n/2) \cdot O(n/N)$ için tasarlanan algoritma $= O(n \cdot \log(n)/N) + O(n)$ dir.

Burada maliyet

$c(n) = O(n \cdot \log(n)) + O(n \cdot N)$ olarak bulunur.

Bu değer $N \leq \log(n)$ için optimumdur.

Şekil 8.3' de MERGE-SPLIT Algoritmasının 12 elemanı sıralanması gösterilmektedir.

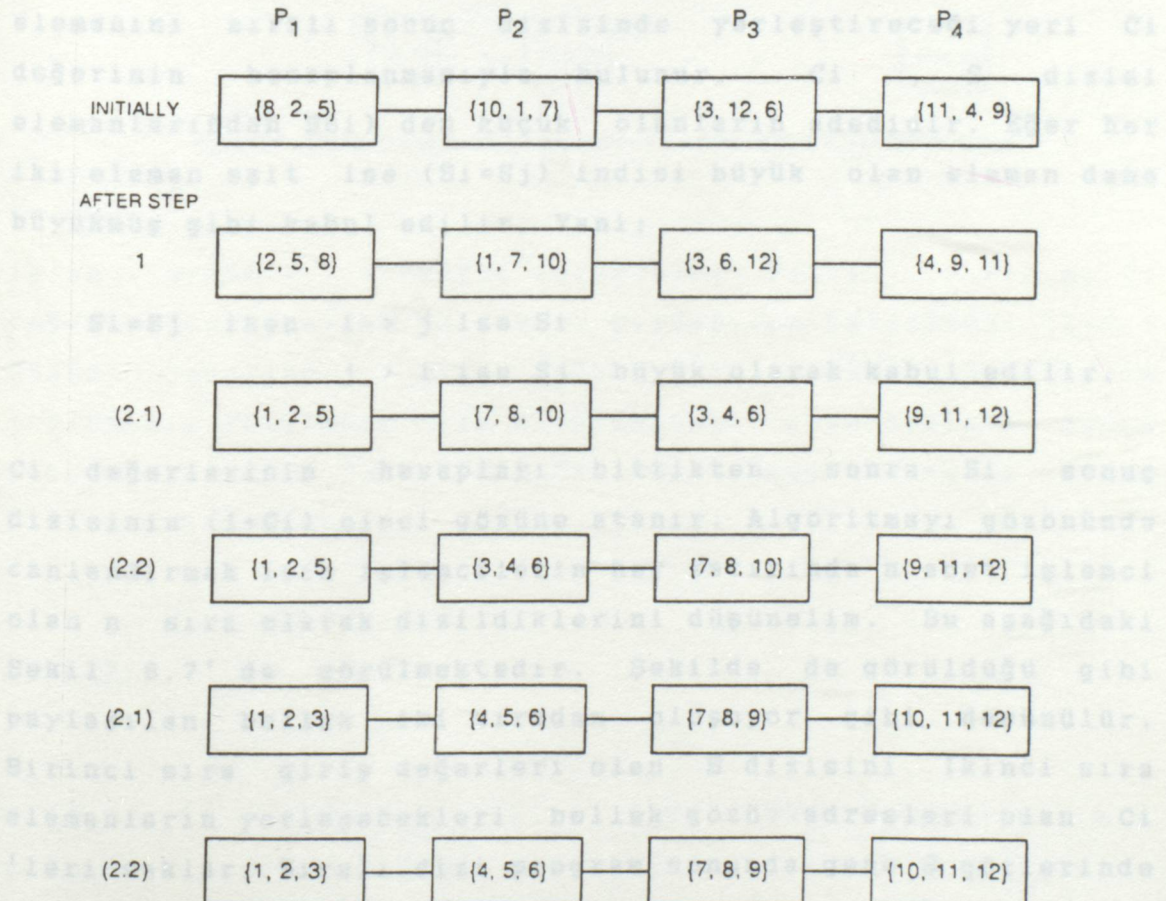


Figure 4.3 Sorting sequence of twelve elements using procedure MERGE SPLIT.

8.5 CRCW TİPİ PARALEL BİLGİSAYARDA SIRALAMA

SIMD (Single Instruction Stream Multiple Data Stream) SM CRCW özelliğine sahip bilgisayar, SIMD tipi bilgisayarların en gelişmiş tipidir. Bu tip bilgisayarlar için tasarlanan algoritmada işlemcilerin, paylaşılan bellekteki bir bellek bölgesine eşzamanlı yazma işlemleri isteklerine çözüm getirmek gerekir. Sıralama probleminde böyle bir durum olduğunda, birden fazla işlemci, tamamen farklı sayıları aynı yere yazmak istiyor demektir. Bu algoritmada bellek gözüne, istemde bulunan tüm işlemcilerin elindeki sayıların toplamı yazılır.

Bu algoritmada 'n' elemanlı bir diziyi sıralamak için $n \times n$ adet işlemci kullanılacaktır. Algoritmanın temelinde Sayarak sıralama (Enumeration Sort) yatar. S dizisinin $S(i)$ elemanını sıralı sonuç dizisinde yerleştireceği yeri C_i değerinin hesaplanmasıyla bulunur. C_i , S dizisi elemanlarından $S(i)$ den küçük olanların adedidir. Eğer her iki eleman eşit ise ($S_i = S_j$) indisi büyük olan eleman daha büyükmüş gibi kabul edilir. Yani;

İşlemcilerden 1. sıradaki işlemciler $P(1,1), \dots, P(1,n)$ C_i değ. $S_i = S_j$ iken $i > j$ ise S_i paylaşılan bellekteki $(1+C_i)$ gözüne yazılır $j > i$ ise S_j büyük olarak kabul edilir.

Toplamın verilmesi ile gerçekleştirildiği için toplam sırası

C_i değerlerinin hesapları bittikten sonra S_i sonuç dizisinin $(1+C_i)$ ninci gözüne atanır. Algoritmayı gözönünde canlandırmak için işlemcilerin her satırında n adet işlemci olan n sıra olarak dizildiklerini düşünelim. Bu aşağıdaki Şekil 8.7' de görülmektedir. Şekilde de görüldüğü gibi paylaşılan bellek iki sıradan oluşuyor gibi düşünülür. Birinci sıra giriş değerleri olan S dizisini İkinci sıra elemanların yerleşecekleri bellek gözü adresleri olan C_i 'leri saklar. Sıralı dizi program sonunda gene S gözlerinde

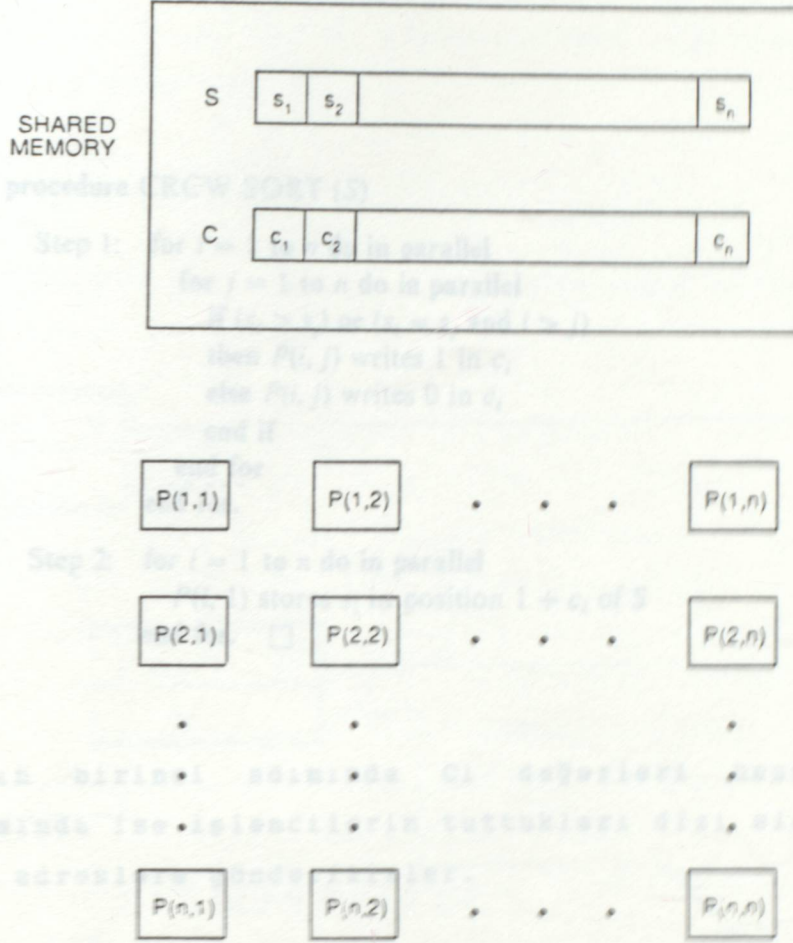


Figure 4.4 Processor and memory organization for sorting on CRCW SM SIMD model.

oluşur.

işlemcilerden i . sıradaki işlemciler $P(i,1), \dots, P(i,n)$ C_i değerini hesaplarlar ve sayıyı paylaşılan bellekteki $(1+C_i)$ görüne yazarlar. Aynı anda yazım isteği değerlerin toplamının yazılması ile karşılandığı için toplama zaten yapılmış olur. CRCW SIRALAMA Algoritmasının skışı Şekil 8.8' de verilmiştir.

ANALİZ:

$S=(5,2,4,5)$ değerleri için CRCW SORT algoritmasının çalışması aşağıdaki Şekil 8.9'da adım adım gösterilmiştir.

CRCW SIRALAMA METODU

procedure CRCW SORT (S)

Step 1: for $i = 1$ to n do in parallel
 for $j = 1$ to n do in parallel
 if $(s_i > s_j)$ or $(s_i = s_j$ and $i > j)$
 then $P(i, j)$ writes 1 in c_i
 else $P(i, j)$ writes 0 in c_i
 end if
 end for
 end for.

Step 2: for $i = 1$ to n do in parallel
 $P(i, 1)$ stores s_i in position $1 + c_i$ of S
 end for. $\square$

Algoritmanın birinci adımında C_i değerleri hesaplanır. İkinci adımında ise işlemcilerin tuttukları dizi elemanları hesaplanan adreslere gönderilirler.

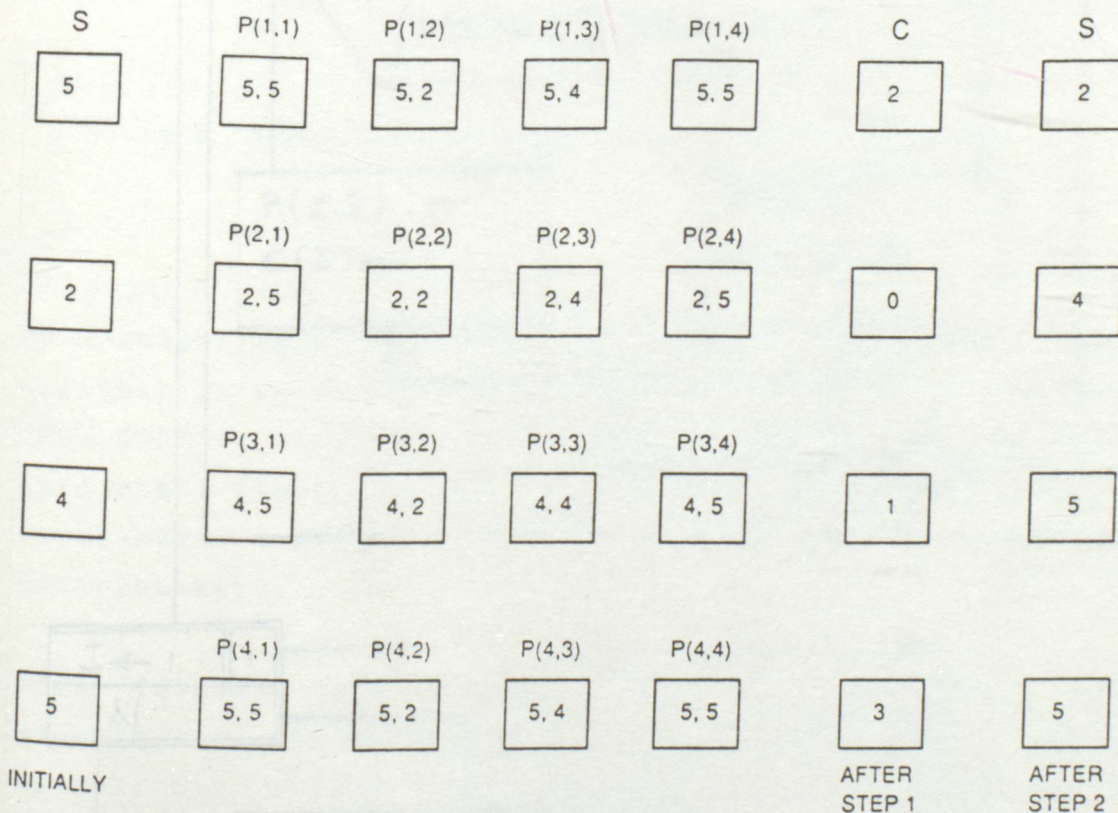
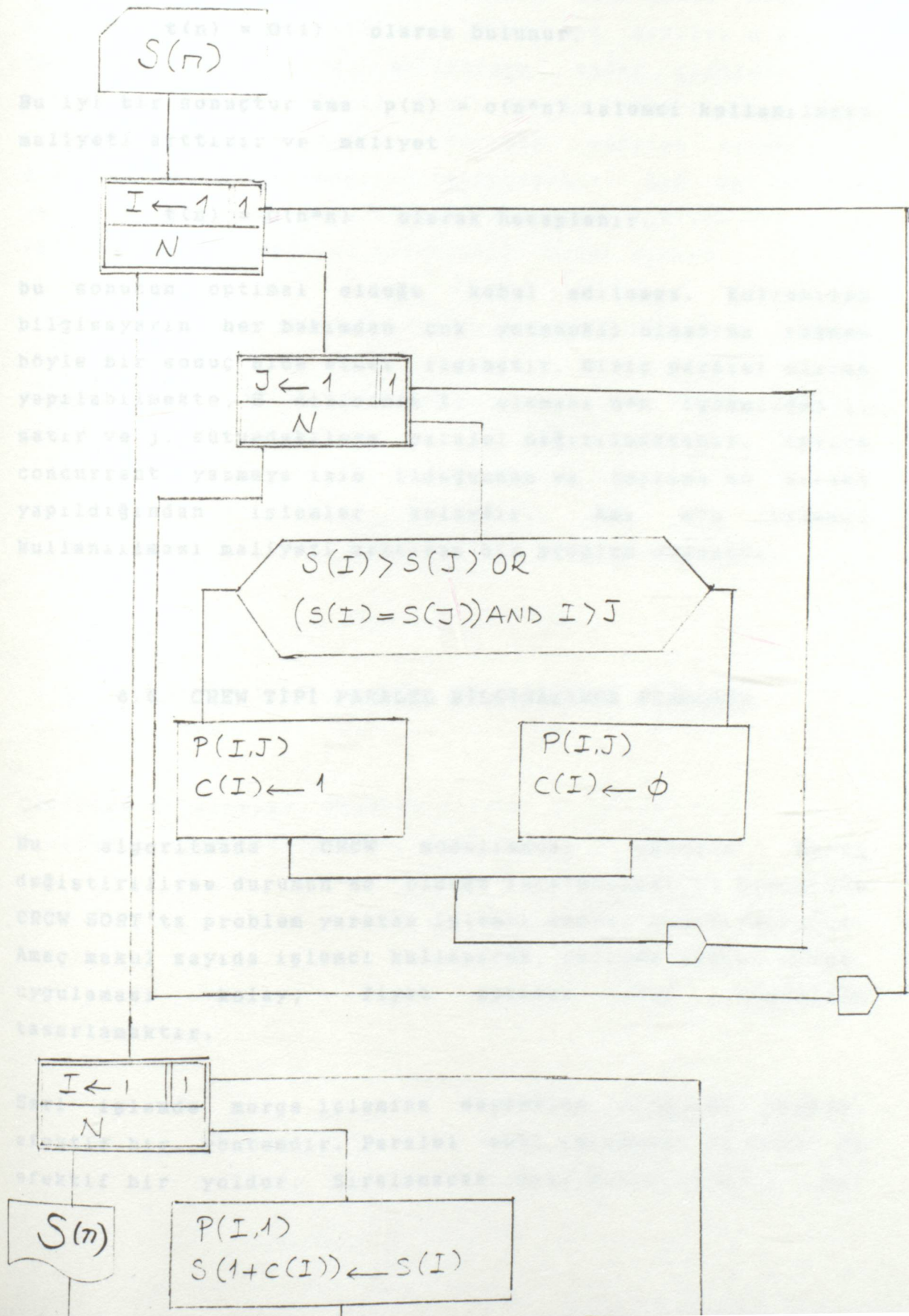


Figure 4.5 Sorting sequence of four elements using procedure CRCW SORT.

CRCW SIRALAMA METODU



$n \times n$ işlemci bu işlemi tamamen paralel olarak yaptıkları için her iki adım da sabit zaman alırlar. Bundan dolayı algoritmanın çalışma zamanı

kullanılarak $t(n) = O(1)$ olarak bulunur.

Bu iyi bir sonuçtur ama $p(n) = o(n \times n)$ işlemci kullanılması maliyeti arttırır ve maliyet

merge edilebilir $t(n) = O(n \times n)$ olarak hesaplanır.

bu sonucun optimal olduğu kabul edilemez. Kullanılan bilgisayarın her bakımdan çok yetenekli olmasına rağmen böyle bir sonuç elde etmek ilginçtir. Giriş paralel olarak yapılabilen, S dizisinin i . elemanı $n \times n$ işlemciden i . satır ve j . sütundakilere paralel dağıtılmaktadır. Ayrıca concurrent yazmaya izin olduğundan ve toplama bu sırada yapıldığından işlemler kolaydır. Ama $n \times n$ işlemci kullanılması maliyeti arttıran bir problem olmuştur.

P1

P2

P3

P4

8.6 CREW TİPİ PARALEL BİLGİSAYARDA SIRALAMA

Bu algoritmada CRCW modelindeki şartlar biraz değiştirilirse durumun ne olduğu incelenecektir. Birincisi CRCW SORT'ta problem yaratan işlemci sayısı azaltılacaktır. Amaç makul sayıda işlemci kullanarak, çalışma zamanı küçük, uygulaması kolay, fiyat optimal bir algoritma tasarlamaktır.

Seri işlemde merge işlemine dayanarak sıralama yapmak, efektif bir yöntemdir. Paralel veri işlemede bu daha da efektif bir yoldur. Sıralanacak dizi $S = (s_1, s_2, s_3, \dots, s_n)$

CREW SORT METODU

$N < n$ şartı ile N işlemcili, $P_1, P_2, \dots, P(N)$ bir CREW SM SIMD paralel bilgisayar gönderilir. N adet işlemci diziyi paylaşır ve herbiri kendi alt dizisini QUICK SORT kullanarak sıralar. Sıralı diziler CREW MERGE Procedürü kullanılarak birleştirilirler. Sonuç alt diziler n boyutlu tek bir dizi elde edilinceye kadar çiftleri ile birleştirilirler. Şekil 8.9' da bu algoritmanın akış diyagramı görülmektedir. Girişte verilen sırasız S dizisinin $S(i)$ elemanı i . işlemcidedir. S_{jk} iki dizinin merge edilmesiyle oluşan yeni alt dizi P_{jk} ise bu merge işlemini gerçekleştiren işlemcidedir. Örnek olarak

$S = 2, 8, 5, 10, 15, 1, 12, 6, 14, 3, 11, 7, 9, 4, 13, 16$ $N = 4$

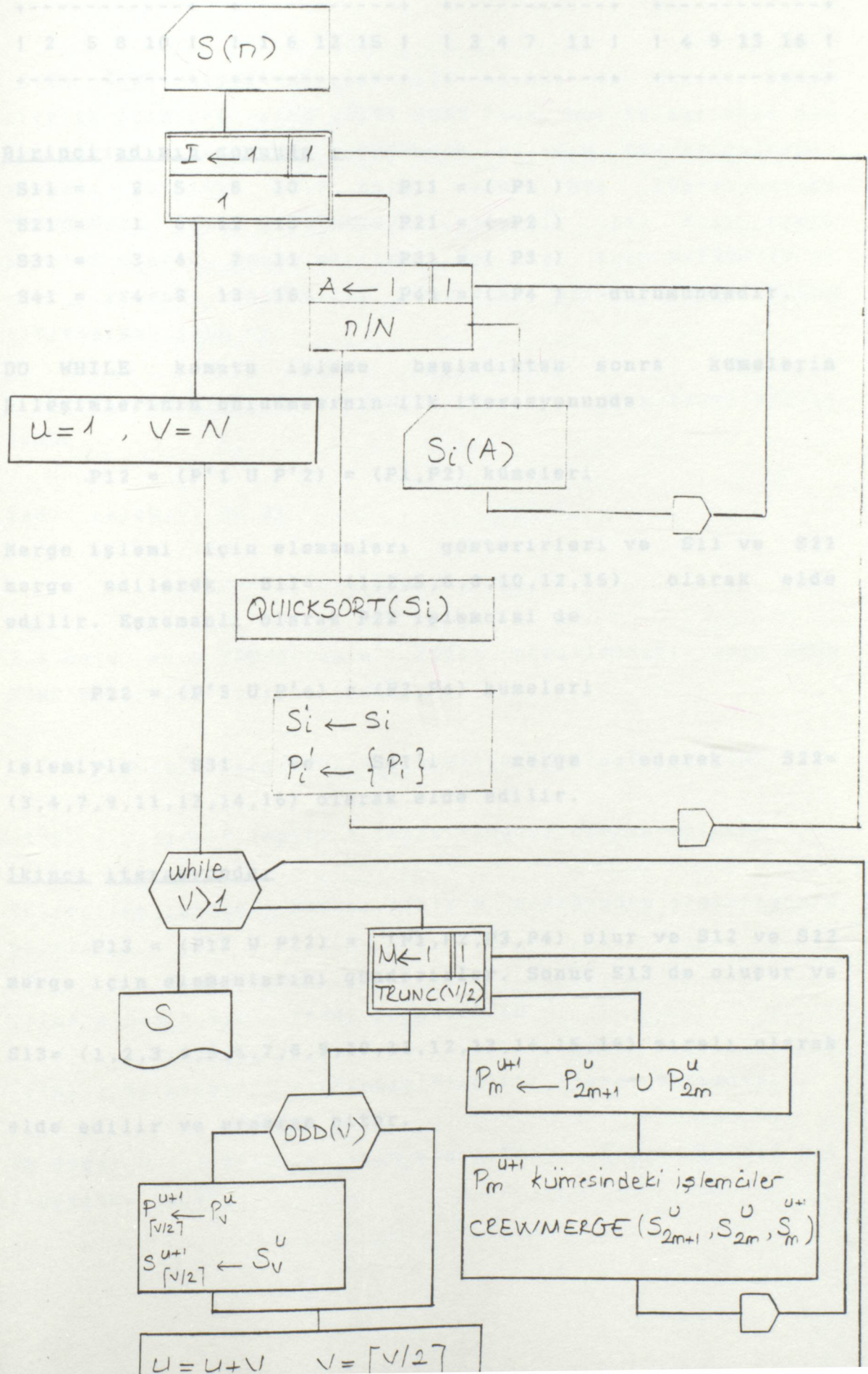
olmak üzere verilen 16 elemanlı S dizisini CREW SORT ile sıralayalım.

1. ADIMDA

P1	P2	P3	P4
+-----+	+-----+	+-----+	+-----+
! 2 8 5 10 !	! 15 1 12 6 !	! 4 3 11 7 !	! 9 4 13 16 !
+-----+	+-----+	+-----+	+-----+

Değerlerini alırlar. Paralel olarak işlemciler kendi dizilerini sıralar ve sonuç aşağıdaki gibi olur.

- CREW SORT METODU -



ANALİZ P1	P2	P3	P4
+-----+	+-----+	+-----+	+-----+
! 2 5 8 10 !	! 1 6 12 15 !	! 3 4 7 11 !	! 4 9 13 16 !
+-----+	+-----+	+-----+	+-----+

Birinci adımın sonunda :

S11 = 2 5 8 10 P11 = (P1)
 S21 = 1 6 12 15 P21 = (P2)
 S31 = 3 4 7 11 P31 = (P3)
 S41 = 4 9 13 16 P41 = (P4)

DO WHILE komutu işleme başladıktan sonra kümelerin bileşimlerinin bulunmasının ilk iterasyonunda

$P_{12} = (P'_1 \cup P'_2) = (P_1, P_2)$ kümeleri

Merge işlemi için elemanları gösterirleri ve S11 ve S21 merge edilerek $S_{11} = (1, 2, 5, 6, 8, 10, 12, 15)$ olarak elde edilir. Eşzamanlı olarak P22 işlemcisi de

$P_{22} = (P'_3 \cup P'_4) = (P_3, P_4)$ kümeleri

işlemiyle S_{31} ve S_{41} 'i merge ederek $S_{22} = (3, 4, 7, 9, 11, 13, 14, 16)$ olarak elde edilir.

İkinci iterasyonda:

$P_{13} = (P_{12} \cup P_{22}) = (P_1, P_2, P_3, P_4)$ olur ve S12 ve S22 merge için elemanlarını gönderirler. Sonuç S13 de oluşur ve

$S_{13} = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16)$ sıralı olarak

elde edilir ve program biter.

ANALİZ:

CREW SORT algoritmasında ilk adımda en etkili işlem sıralam için çağırılan QUICK SORT 'tur. Her işlemcideki alt dizilerin boyutu n/N olduğundan bu adım $O(n/n \cdot \log(n/N))$ zamanda yapılır. 2.3 nolu adımın her iterasyonunda $\text{TRUNC}(V/2)$ adet $n/\text{TRUNC}(V/2)$ elemanlı iki dizi çifti paralel olarak merge edilir. Her dizi için $N/\text{TRUNC}(V/2)$ adet işlemci kullanılır. Bundan dolayı CREW MERGE altprogramı için

$O(((n/\text{TRUNC}(V/2))/(n/\text{TRUNC}(V/2))) + \log(n/\text{TRUNC}(V/2)))$ kadar zaman gerekir. Bu da

$O(n/N \cdot \log(n))$ demektir.

2.3 nolu adım $\text{TRUNC}(\log(n))$ kadar tekrarlandığı için CREW SORT'un toplam çalışma zamanı;

$$t(n) = O((n/N) \cdot \log(n/N)) + O((n/N) \cdot \log(N) + \log(n) \cdot \log(N))$$

$$t(n) = O((n/N) \cdot \log(n) + \log(n) \cdot \log(n)) \text{ olarak bulunur.}$$

Kullanılan işlemci sayısı $p(n) = N$ olduğundan algoritmanın maliyeti:

$$c(n) = p(n) \cdot t(n) \text{ genel formülünden}$$

$$c(n) = O((n \cdot \log(n) + n \cdot \log(n) \cdot \log(n))) \text{ olarak hesaplanır.}$$

Bu değer $N \leq n \cdot \log(n)$ için algoritmayı fiyat optimal bir algoritma yapar.



FOURIER SERİSİ HESAPLAMALARI

9.1-Ayrık Fourier Dönüşümü (Discrete Fourier Transform)

Bu bölümde bilimsel hesaplamalarda çok sık kullanılan AYRIK FOURIER DÖNÜŞÜMÜ incelenecektir.

Bir işarete ait N adet örnek verilmiş olsun. Buna karşı düşen FOURIER dönüşümü hesabı istenir. Giriş N adet real sayıdan oluşan bir vektör çıkış ise aynı boyutta kompleks sayılardan oluşan yeni bir vektördür.

Verilen $(a_0, a_1, a_2, \dots, a_{(N-1)})$ sayı dizisi için hesaplanan AYRIK FOURIER DÖNÜŞÜMÜ $(b_1, b_2, b_3, \dots, b_{(N-1)})$ dir. Bu katsayılar şu formülle hesaplanırlar.

$$a(w_j) = b_j = \sum_{k=0}^{n-1} a_k w^{kj}, \quad w = e^{\frac{2\pi i}{n}}, \quad i = \sqrt{-1}, \quad j = 0, 1, \dots, n-1$$

(f.1)

9.2-Hızlı Fourier Dönüşümü (Fast Fourier Transform)

ileri doğru yapılan ardaşıl hesaplamalarda b_j 'yi bulmak için kompleks sayılarda, n adet çarpma ve n-1 adet toplama yapmak gereklidir. Bu da $(b_0, b_1, b_2, b_3, \dots, b_{(n-1)})$ serisinin tamamını hesaplamak için $O(n \cdot n)$ kadar zaman gerektirir. Eğer 'n' çok büyükse bu hesaplama imkansızdır. Oysa uygulamalarda çoğunlukla buna gerek duyulur. Çözüm daha iyi bir algoritmadır. Bu da hızlı FOURIER dönüşümüdür ve $O(n \cdot \log n)$ zamanda çalışır.

Ayrık fourier dönüşümü girişte verilen n boyutlu a vektörünü gene aynı boyutlu ama bu defa kompleks sayılardan oluşan b vektörüne dönüştürür. Bu işlem sırasında 1 in n inci köklerinden yararlanılır.

$$\omega_j = e^{2\pi i j / n} \quad j = 0, 1, \dots, (n-1) \quad (f.2)$$

$$j = 0 \quad \rightarrow \omega_0 = e^{2\pi i \cdot 0 / n} = 1$$

$$j = 1 \quad \rightarrow \omega_1 = e^{2\pi i / n}$$

.....

$$j = n-1 \quad \rightarrow \omega_{n-1} = e^{2\pi i (n-1) / n}$$

$\omega_j = e^{(\frac{2\pi j}{n})i}$ ifadesi bir kompleks sayının üstel ifadesidir. Bunu $(a+ib)$ şeklinde gösterelim.

$$e^{i\varphi} = \cos \varphi + i \sin \varphi \quad (f.4)$$

genel formülünden yararlanarak,

$$e^{(\frac{2\pi j}{n})i} = \cos \left(\frac{2\pi j}{n} \right) + i \sin \left(\frac{2\pi j}{n} \right) \quad (f.5)$$

Bundan sonra $\omega(j)$, $(a + i*b)$ formunda kullanılacak ve

$$\omega_j = \cos \left(\frac{2\pi j}{n} \right) + i \sin \left(\frac{2\pi j}{n} \right)$$

$$j = 0, 1, 2, \dots, (n-1)$$

(f.6)

kompleks ifadesi ile bulunacak $U = (2\pi * j) / n$ ise

$$\omega_j = \cos u + i \sin u$$

(f.7)

olarak kullanılabilir.

n yani verilen örnek sayısı 2 'nin katı olarak kabul

edilirse, S pozitif bir tamsayı olmak üzere, $n = 2^S$ dir.

Fourier dönüşümü şu formülle hesaplanır.

$$a(w_j) = \sum_{k=0}^{n-1} a_k e^{2\pi i j k / 2^S}, \quad j = 0, 1, 2, \dots, 2^{S-1} \quad (f.8)$$

şimdi bu ifadeyi k 'nin tek ve çift değerleri için ayrı toplamalar olarak yazalım. Birinci toplamda $k=2*m$ ve ikinci toplamda $k=2*m+1$ yerine konursa, $j = 0, 1, 2, \dots, 2^{S-1}$ için,

$$b_j = a(w_j) = \sum_{m=0}^{2^{S-1}-1} a_{2m} e^{2\pi i j m / 2^{S-1}} + \sum_{m=0}^{2^{S-1}-1} a_{2m+1} e^{2\pi i j (2m+1) / 2^S}$$

$$b_j = \sum_{m=0}^{2^{S-1}-1} a_{2m} e^{2\pi i j m / 2^{S-1}} + e^{2\pi i j / 2^S} \sum_{m=0}^{2^{S-1}-1} a_{2m+1} e^{2\pi i j m / 2^{S-1}} \quad (f.9)-(f.10)$$

Burada özel bir olay gerçekleşir. Birinci toplam $(a_0, a_2, a_4, \dots, a(2^{S-2}))$ dizisinin, ikinci toplam ise $(a_1, a_3, a_5, \dots, a(2^{S-1}))$ dizisinin fourier dönüşümüdür.

Ayrıca yukarıdaki her toplamda görülen ifade kendinden önce gelen daha kısa terimlerin fourier dönüşümüdür. Yani fonksiyon recursive bir fonksiyondur.

$$W = e^{2\pi i / n}$$

(f.11)

dönüşümü yapılırsa f.10 formülü,

$$b_j = \sum_{m=0}^{2^{S-1}-1} a_{2m} e^{2\pi i j m / 2^{S-1}} + W^j \sum_{m=0}^{2^{S-1}-1} a_{2m+1} e^{2\pi i j m / 2^{S-1}}$$

(f.12)

diğer bir gösterim tarzına (f.12) döndürülür. Daha kesin olarak bunu söyleyebiliriz. Bu Fourier dönüşümü, polinom katsayılar dizisinden belli bir nokta kümesi için hesap elde edilir. Her dizisine dönüştürülür. Bu iki form aras. Sonuçta elde edilen n adet b_j Fourier dönüşümünün sonucudur. Bunun programı SEQFFT'de verilmiştir.

$$Q_0 + Q_1 X + Q_2 X^2 + \dots + Q_{n-1} X^{n-1} + Q_n X^n$$

9.3-Ardışıl Hızlı Fourier Transform'u ALGORİTMASI

(b₀, b₁, b₂, ..., b_(n-1)) dizisi verildiğinde bu dizinin b_j

GİRİŞ : Örneklerin verildiği A dizisi

ÇIKIŞ : FFT'nin katsayılarını veren kompleks sayı dizisi B

KN : örnek sayısı

Hızlı Fourier Dönüşümü İçin Bir Uygulama

Hızlı fourier dönüşümünün efektif bir metod uygulaması, onun sayısal işaret işleme, kodlama teorisi, bilgisayarlı tomografi, konuşma iletme, hava tahmini, istatistik, görüntü işleme, çok büyük tamsayıların ve polinomların çarpımı gibi, hesaplamaların gerektiği alanlarda oldukça popüler bir teknik olarak kabul görmesini sağlamıştır.

İki polinom düşünelim. Birincisinin f(x) derecesi 77, ikincisinin g(x) derecesi 94 olsun. Bu iki polinomun çarpımı açıkça görüldüğü gibi çok fazla zahmetlidir. Sadece tek bir katsayısını bulmak bile zordur. x**50 nin katsayısını hesaplamak istersek, örneğin 50 sayının birbiriyle çarpımı ve sonunda toplamı gerekir. Daha hesaplanacak 171 katsayı vardır!

f(x)g(x) çarpımının katsayılarını teker, teker hesaplamak yerine sadece diyelim ki 172. noktada çarpımın değeri hesaplanır. Bunu yapmak için basitçe f(x)'in ve g(x)'in o noktadaki değerleri çarpılır. Bu 172 değer çarpım polinomunu tanımlamaya yeter.

Fourier dönüşümü ile polinom bir gösterim tarzından

diğer bir gösterim tarzına dönüştürülür. Daha kesin olarak şunu söyleyebiliriz. 'Fourier Dönüşümü, polinomu katsayılar dizisinden belli bir nokta kümesi için hesaplanmış değerler dizisine dönüştürür.' Bu iki form arasında dönüşüm kolaydır.

2. ADIM Polinomlar aşağıdaki gibi verilmiş olsun, FFT'yi hesaplanır.

$$a_0 + a_1x + a_2x^2 + \dots + a_{n-2}x^{n-2} + a_{n-1}x^{n-1}$$

3. ADIM Katsayılar dizisi : $(a_0, a_1, a_2, \dots, a_{n-1})$ dir. $(b_0, b_1, b_2, \dots, b_{n-1})$ dizisi verildiğinde bu dizinin b_j elemanı $x = W^j$ 'de $a(x)$ polinomunun aldığı değerdir.

4. ADIM $W^0, W^1, W^2, \dots, W^{n-1}$ n . kökleridir.

Buna zıt olarak $b_0 + b_1x + \dots + b_{n-1}x^{n-1}$ polinomunun

$x = (W^{-1})^k$ 'daki değeride, $k=0, 1, \dots, (n-1)$

5. ADIM $a_k = \frac{1}{n} \sum_{j=0}^{n-1} b_j (W^{-1})^{jk}$ (f.13)

olarak verilmiştir. bunun katsayılarıdır.

Yani $(a_0, a_1, a_2, \dots, a_{n-1})$ dizisi $(b_0, b_1, b_2, \dots, b_{n-1})$ dizisinin TERS AYRIK FOURIER DÖNÜŞÜMÜ (Inverse DFT) dır, ve $O(n \log n)$ zamanda hesaplanır.

iki polinomun EFT ile çarpımı

$$f(x) = a_0 + a_1x + a_2x^2 + \dots + a_{n-1}x^{n-1} = \sum_{j=0}^{n-1} a_j x^j$$

$$g(x) = c_0 + c_1x + c_2x^2 + \dots + c_{n-1}x^{n-1} = \sum_{k=0}^{n-1} c_k x^k$$

(f.14)-(f.15)

$h(x) = f(x) * g(x)$ $O(n^2)$ işlem ister. Ama FFT

kullanılırsa bu $O(n \log n)$ (n örnek sayısı) olur.

1.ADIM : N , 2^{n-1} 'den büyük olan minimum 2 'nin katıdır.
 $O(N)$ $(a_0, a_1, \dots, a_{(n-1)})$ ve $(c_0, c_1, \dots, c_{(n-1)})$ dizilerinin her ikisi $n-N$ adet sıfırla doldurulur.

2.ADIM : $(a_0, a_1, \dots, a_{(n-1)}, 0, \dots, 0)$ ın FFT'si hesaplanır.
 $O(N \log N)$ Bu $f(x)$ polinomunun N . kökteki değerini verir.

3.ADIM : $(c_0, c_1, \dots, c_{(n-1)}, 0, \dots, 0)$ ın FFT'si hesaplanır.
 $O(N \log N)$ Bu $g(x)$ polinomunun N . kökteki değerini verir.

4.ADIM : $W = e^{\frac{2\pi i}{N}}$ iken $j = 0, 1, \dots, (n-1)$ için $f(W^j) * g(W^j)$
 N adet hesaplanır. Bunun sonucu sayılar $h(x)$ çarpım
 çarpma polinomunun birliktteki N . kökünü verir.

5.ADIM : $(f(W^j) g(W^j), f(W^{2j}) g(W^{2j}), \dots, f(W^{(n/2-1)j}) g(W^{(n/2-1)j}))$ için Ters
 $O(N \log N)$ Fourier dönüşümü hesaplanır. Sonuçta çıkan sayılar
 $h(x)$ polinomunun katsayılarıdır.

$N < 4n$ olduğundan toplam işlem zamanı $O(n \log n)$ dir.

9.4-DFT (Discrete Fourier Transform)'in PARALEL HESAPLAMASI

DFT'nin hesaplanmasında hatırı sayılır derecede ,
 doğal olarak yapıdan gelen paralellik vardır. Bu
 paralelliği ortaya çıkarmak için iki genel yaklaşım
 benimsenebilir.

1- Birinci yaklaşımda $(b_0, b_1, b_2, \dots, b_{(n-1)})$ dizisi,
 $N \geq n$ ise N işlemci kullanılarak

$$b_j = \sum_{k=0}^{n-1} a_k w^{kj}$$

(f.16)

doğrudan hesaplanır. Bunun çalışma zamanı lineer olup en fazla 'n' dir. Fiyatı (cost) ise en azından 2'nin n. katıdır.

2- İkinci yaklaşımda, FFT'nin paralel versiyonları türetilmiştir. Bunlar arasında en iyi çalışanı 'n' adet işlemci kullanıp $O(\log n)$ zamanda çalışandır. Sonuçta maliyeti $O(n \log n)$ dir. Bu maliyet ardışıl FFT'nin (SEQUENTIAL FFT) maliyetine de uyar.

1.YAKLASIM : DFT'nin DOGRUDAN HESAPLANMASI

$$b_j = \sum_{k=0}^{n-1} a_k w^{kj}$$

(f.16) ifadesi aşağıdaki matris ve vektörün çarpımı olarak ifade edilebilir

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & w & w^2 & w^3 & \dots & w^{n-1} \\ 1 & w^2 & w^4 & w^6 & \dots & w^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{n-1} & w^{2(n-1)} & w^{3(n-1)} & \dots & w^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

(f.17)

$$b = wa$$

Hesaplama için yapılacak işlemler,

- 1.işlem : W matrisini hesaplayan, $O(\log n)$ zamanda çalışan $n \times n$ işlemci kullanılan bir algoritma oluşturmak,
- 2.işlem : Uygun bir içbağılantılı bir ağ yapısı sağlandığında işlemcilerden birinin DFT'nin hesaplanmasında nasıl kullanıldığını görmek

TARTIŞMA

1. İŞLEM

Matrisinde kullanılan SIMD bilgisayarın ortak bellek mi kullandığını W MATRİSİNİN HESAPLANMASI (örnek sayısı $n-1$) ile mi düzenlendiğinden hiç bahsedilmemiştir.

$n \times n$ işlemcili SIMD tipi bilgisayar kullanılır. İşlemciler $n \times n$ boyutlu matris şeklinde bağlanmışlardır. Bir $P(k, j)$ işlemcisi $W^{(k-1)(j-1)}$ kökünü $k \geq 1, j \leq n$ şartı ile hesaplamakla görevlidir. Bu işlem tekrar, tekrar kare olarak ve çarparak gerçekleştirilir. Örneğin

2. İŞLEM

$$W^{13} = [(W^2)^2 \times W] \times [(W^2)^2]^2 = W^5 \cdot W^8 = W^{13}$$

Başlangıçta her işlemci 'MESH OF TREES' Bu COMPUTE procedüründe verilmiştir. Her işlemcinin 3 saklayıcısı vardır. (Ref 8 / sh 233 fig 9.1)

M_{kj} : W'nun alınacak kuvvetini saklar. (binary tree)
 X_{kj} : Başlangıçta W değerini, hesaplama sırasında da ara değerleri saklar.
 Y_{kj} : Hesaplama sırasında da ara değerleri saklar. İşlem bittiğinde $Y_{kj} = W^{(k-1)(j-1)}$ sonucunu içerir. (Ref 8/sh 234)

ANALİZ

1, 2 ve 3. adımlar sabit bir zamanda gerçekleştirilirler. 4. adım $O(\log((k-1) \cdot (j-1)))$ iterasyondan oluşur ve her iterasyon sabit bir zaman alır. Bundan dolayı COMPUTE W PROCEDURE '1 $O(\log n)$ zaman alır. Gerçekte $W^{n/2} = -1$ olduğunda Procedure'ün gerçek çalışma zamanı biraz azaltılabilir. Bundan dolayı $W^{j+n/2} = -W^j$ dir. Sonuç olarak sadece W'nin $n/2$ 'den küçük kuvvetlerini hesaplamak gerekir.

procedure COMPUTE W (k, j)

Step 1: $M_{kj} \leftarrow (k-1)(j-1)$

Step 2: $X_{kj} \leftarrow W$

Step 3: $Y_{kj} \leftarrow 1$

Step 4: while $M_{kj} \neq 0$ do

(4.1) if M_{kj} is odd

then $Y_{kj} \leftarrow X_{kj} \times Y_{kj}$

end if

(4.2) $M_{kj} \leftarrow \lfloor M_{kj}/2 \rfloor$

TARTIŞMA

Burada kullanılan SIMD bilgisayarın ortak bellek mi kullandığı yoksa interconnection network ile mi düzenlendiğinden hiç bahsedilmemiştir.

Gerçekten de COMPUTEW procedure'ı diğer işlemciler ile iletişim kurma ihtiyacı hissetmez. Çünkü her işlemci W'nin bir kuvvetini diğer tüm işlemcilerden bağımsız olarak hesaplar. $\log n$ dir.

2. İŞLEM

DFT'nin HESAPLANMASI

Başlangıçtan beri sözü edilen $n \times n$ işlemci 'MESH OF TREES' adı verilen ve aşağıdaki şekilde gösterilen bir yapı ile işten bağlanmışlardır. (Ref 8 /sf 235 fig 9.1)

Bağlantı şu şartlarla gerçekleşmiştir.

- 1- k. satırdaki işlemciler bir ikili ağaç (binary tree) oluşturacak şekilde $P(k, 2 \lfloor n/2 \rfloor + 1)$ işlemci ile bağlantılıdır. $n = \text{çift}$ ise bu işlemci yoktur.
 $j = 0, 1, \dots, \lfloor n/2 \rfloor$ için $P(k, j)$. işlemci $P(k, 2j)$. ve $P(k, 2j+1)$. işlemciyle doğrudan bağlantılıdır.
- 2- j. sütundaki işlemci $P(2 \lfloor n/2 \rfloor + 1, j)$. işlemciye ikili ağaç oluşturacak şekilde işten bağlıdır. $n = \text{çift}$ ise bu işlemci yoktur. $j = 0, 1, \dots, \lfloor n/2 \rfloor$ için $P(k, j)$. işlemci $P(2k, j)$. ve $P(2k+1, j)$. işlemciyle doğrudan bağlantılıdır.

Bu tür mimariye MESH OF TREES adı verilir ve $n=4$ için olanı şekilde görülmektedir.

1. satırdaki işlemciler bu giriş işlemlerini, 1. sütundakiler ise çıkış işlemlerini yapmakla yükümlüdürler. Buna göre örneğin $P(1, j)$. işlemci a_j verisini okuyabilir ve ikili ağaç bağlantısını kullanarak a_j yi j sütunundaki tüm işlemcilere yayar. Bu algoritma PROPAGATE procedure'da verilmiştir ve $O(\log n)$ zamanda çalışır. (Ref 1/ Sh 238 Fig 9.2)

procedure PROPAGATE (a_j)

```

for m = 1 to (log n) - 1 do
  for k =  $2^{m-1}$  to  $2^m - 1$  do in parallel
    P(k, j) sends  $a_j$  to P(2k, j) and P(2k + 1, j)
  end for

```


Benzer olarak k sütununa yerleşmiş her işlemcinin bir ' d_{kj} ' sayısını tuttuklarını ve bunların toplamlarının gerektiğini düşünelim. Gene ikili ağaç bağlantısını kullanarak toplam bulunabilir ve bu sütunun çıkış işlerine bakan $P(k,1)$ işlemci bunu çıkış olarak tutar. Bu algoritma SUM procedure'ında verilmiştir. Bu algoritmanın da çalışma zamanı $O(\log n)$ dir.

Buraya kadar incelenen COMPUTEW, PROPAGATE ve SUM procedurelarını kullanarak 'Fast Fourier Transform' u aşağıda verilen SIMDDFT programı ile hesaplanabilir.

SIMDDFT programı

procedure SIMD DFT (A, B)

Step 1: for $k = 1$ to n do in parallel
 for $j = 1$ to n do in parallel
 COMPUTE W (k, j)
 end for
 end for.

Step 2: for $j = 1$ to n do in parallel
 (2.1) $P(1, j)$ receives a_{j-1} as input
 (2.2) PROPAGATE (a_{j-1})
 end for.

Step 3: for $k = 1$ to n do in parallel
 for $j = 1$ to n do in parallel
 $d_{kj} \leftarrow Y_{kj} \times a_{j-1}$
 end for
 end for.

Step 4: for $k = 1$ to n do in parallel
 (4.1) SUM (k)
 (4.2) $b_{k-1} \leftarrow d_{k1}$
 (4.3) $P(k, 1)$ produces b_{k-1} as output
 end for. □

$b = Wa$ hesabı ile DFT hesaplar. Program 4 adımdan oluşur.

1.ADIM : İlk olarak W matrisinin elemanları oluşturulur. Bu matrisin her bir elemanını $n \times n$ işlemciden biri hesaplar. Bunu COMPUTEW procedure'ü yapar.

2.ADIM : a vektörünün elemanları dışarıdan okunur. Matris şeklinde yerleşmiş işlemcilerden 1.sırada olanlardan herbiri dışarıdan a 'nın bir elemanını okur. Dolayısıyla giriş işleminde paralel olarak yapılmaktadır. a sayısını alan her birinci satır işlemcisi bu sayıyı kendi sütununun diğer işlemcilerine dağıtır. Bunu gerçekleştiren PROPAGATE (k-1)(j-1) procedur'ı yapar. Bu noktada $P(k, j)$ numaralı işlemci W değerini ve a_{j-1} sayısını tutar.

3.ADIM : Tüm $a \times W$ çarpımları eşzamanlı (concurrently, simultaneously) yapılır. Tüm d_{kj} ler bir anda hesaplanmış olurlar.

4.ADIM : Son olarak her satır için bu d sayıları, yani $a \times W$ çarpımının sonuçları toplanır ve sonuç 1. sütundaki çıkışıyla görevli işlemcide birikir. Bu işlemde de n adım paralel olarak yapılır. Yani her satır toplamı eş zamanlı yapılan bir işlemdir.

ANALİZ

SIMDDFT programında 1,2. ve 4. adımlar $O(\log n)$ zamanda çalışırlar. 3.adım ise sabit bir miktar zamanda gerçekleşir. Bu çarpma işlemi için geçen zamandır. Dolayısıyla programın tüm çalışma zamanı

$$t(n) = O(\log n) \text{ dir.}$$

Hatırlanırsa SEQFFT programı $O(n \log n)$ zamanda çalışmaktaydı. SIMDDFT, SEQFFT'ye oranla $n(\log n)/(\log n) = O(n)$ kadar daha hızlı çalışır. Gerçekte $O(\log n)$ mümkün olan en iyi zamandır. b değeri n adet büyüklüğün toplanmasıyla bulunmaktadır ki bu işlem $\Omega(\log n)$ paralel zaman alır.

Kullanılan işlemci sayısı $p(n) = n$ olduğundan toplam maliyet,

$$c(n) = p(n)t(n) = O(n)O(\log n) = O(n \log n) \text{ dir.}$$

Bu da SIMDDFT sequential FFT'ye göre $O(1/n)$ daha efektif demektir.

2.YAKLASIM

PARALEL FFT ALGORİTMASI

Daha önce incelenen SIMDDFT algoritması $O(\log n)$ zamanı ile gerçekte hızlıdır ve DFT'yi hesaplayan en iyi seri

algoritmaya göre mümkün olan en iyi hızlanmayı verir ama $n \times n$ işlemci ile iş yapmak çok zordur.

Bu bölümde daha efektif bir paralel algoritma tanıtılacaktır. Bu algoritma daha önce incelenen SEQFFT'nin paralel ve recursive olmayan bir versiyonudur. n işlemcili Mesh yapısında bir SIMD bilgisayarda çalışmak üzere tasarlanmıştır. İşlemciler $P_0, P_1, P_2, \dots, P_n$ ve $n = 2^{2s}$ olmak üzere 2×2 boyutlu bir matris olarak satır esaslı olarak düzenlenmişlerdir. Şekilde $n = 16$ için işlemcilerin durumu görülmektedir (Ref 1/ Sh 239 Fig 9.3)

k sayısı $\log n$ adet bitten oluşan binary bir tamsayı olsun. $r(k)$, k sayısının bitlerini ters sırada çevrilmesinden oluşan $\log n$ bitlik yeni bir integerdir.

$k = 01011$ ise $r(k) = 11010$ dir.

MESH FFT ALGORİTMASI

Aşağıda MESHFFT verilmiştir.

Giriş $(a_0, a_1, \dots, a_{n-1})$ mesh içinde işlemcilerde ilk değer olarak tutulur. Her işlemci bir a değerini tutar. $k = 0, 1, \dots, n-1$ için P_k işlemcisi a_k değerini n . işlem bittiğinde çıkış dizisi $(b_0, b_1, \dots, b_{n-1})$ elemanları olan b_k P_k işlemci tarafından içerilir. $k = 0, 1, 2, \dots, n-1$

Programa bakılırsa 2., 4. adımın (ii) kısmında, C_k 'nin (i) kısmında hesaplanan yeni değerinin değil, eski değerinin C_{k+p} kullanıldığı görülür. Bu C_k ve C_{k+p} eşzamanlı olarak güncelleştiriliyor demektir.

ANALİZ

1. adımın amacı giriş dizisinin her bir elemanını her işlemcinin yöresel belleğine göndermektir. Ve sabit bir zaman alır.

2. adım atamalardan ve hesaplamalardan oluşur.

3. adım sadece atamalardan oluşur. Şimdi bu iki tip işlem için gerekli zamanı inceleyelim. Her işlemci $2(2^h - 1)$ kenarla birbirinden ayrılmıştır ki bu da atama işlemi HESAPLAMALAR birim zaman demektir. Sonuç olarak 2. ve 3. adımda atama işlemleri $O(n^2)$ zaman alır ki bu da $O(n^2)$ demektir.

2. adımda $\log n$ iterasyon vardır. Her iterasyonda P işlemci sabit sayıda hesaplama yapar. Bunlardan en fazla zamanı alan üst almadır ve $O(\log n)$ zaman alır. Bundan dolayı hesaplamalar kısmı toplam $O(\log n * \log n)$ zamanda biter.

$$c(n) = O(n^2)O(n) = O(n^3) \text{ dir.}$$

ATAMA İŞLEMLERİ

Procedure $O(n^2 \log n)$ ilk başlangıçta sağlanmıştır ve efektifliği $O(\log n / n^2)$ dir.

Bir verinin bir işlemciden hemen yanında bulunan komşu işlemciye aktarımı için bir zaman birimi gereklidir. 2. ve 4. adımda $k \bmod p = k \bmod 2p$ ise P_k işlemci c_k verisini (c_k ve c_{k+p} 'yi güncelleştirmek için) P_k işlemciden alır ve c_{k+p} değerini P_k işlemciye geri gönderir. Bu işlemler için gerekli zaman ise h 'a bağlıdır. $h = 0$, $p = 1$ iken iletişim aynı satır veya sütun üzerinde olup ta indisleri farkı 1 olan işlemciler (yani matris, mesh tipi bağlantıda birbirine doğrudan bağlı işlemciler) arasındadır. $h = 1$ $p = 2$ iken iletişim aynı satır veya sütun üzerinde olan ve indis farkları 2 olan işlemciler arasında demektir ve iletişim 2 birim zaman alır.

Aynı mantıkta devam edilirse $h = \log n - 1$ ve $p = n/2$ iken aynı sütun üzerinde ve indis farkları $n/2$ olan işlemciler arasındaki iletişim $n^{1/2}/2$ zaman alır.

Genelde $p = 2^h$, $h = 2s-1, 2s-2, \dots, 0$ için atamalar için gerekli zaman $2^{(h \bmod s)}$ dir. Bundan dolayı 2. adımdaki tüm atamalar için gerekli zaman

$$2(1+2+4+\dots+2^{s-1}) = 2(2^s - 1) \text{ dir.}$$

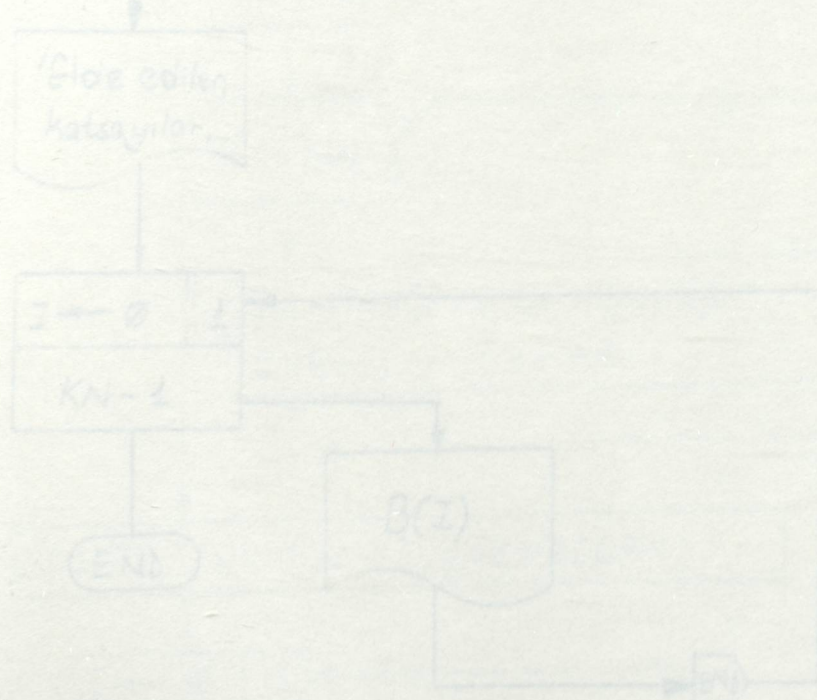
3. adımda $c_{r(k)}$, $P_{r(k)}$ dan P_k 'ya yollanmaktadır. Bu iki işlemci birbirine en uzak işlemcilerdir. $P_{2^{s-1}}$ (kuzeydoğu) $P_{2^s(2^{s-1})}$ (güneybatı) köşesinde yer alırlar. Bu iki işlemci $2(2^s - 1)$ kenarla biribirinden ayrılmışlardır ki bu da atama işlemi için $2(2^s - 1)$ birim zaman demektir. Sonuç olarak 2. ve 3. adımda atama işlemleri $O(s \cdot s)$ zaman tutar ki bu da $O(n^{1/2})$ demektir.

Yeteri kadar büyük n değerleri ile işlem yapıldığında atamalar için geçen zaman hesaplamalar için harcanan zamanın üstüne çıkar. Bundan dolayı MESHFFT'nin baştan sona çalışma zamanı $t(n) = O(n^{1/2})$ dir. $p(n) = n$ idi. O halde

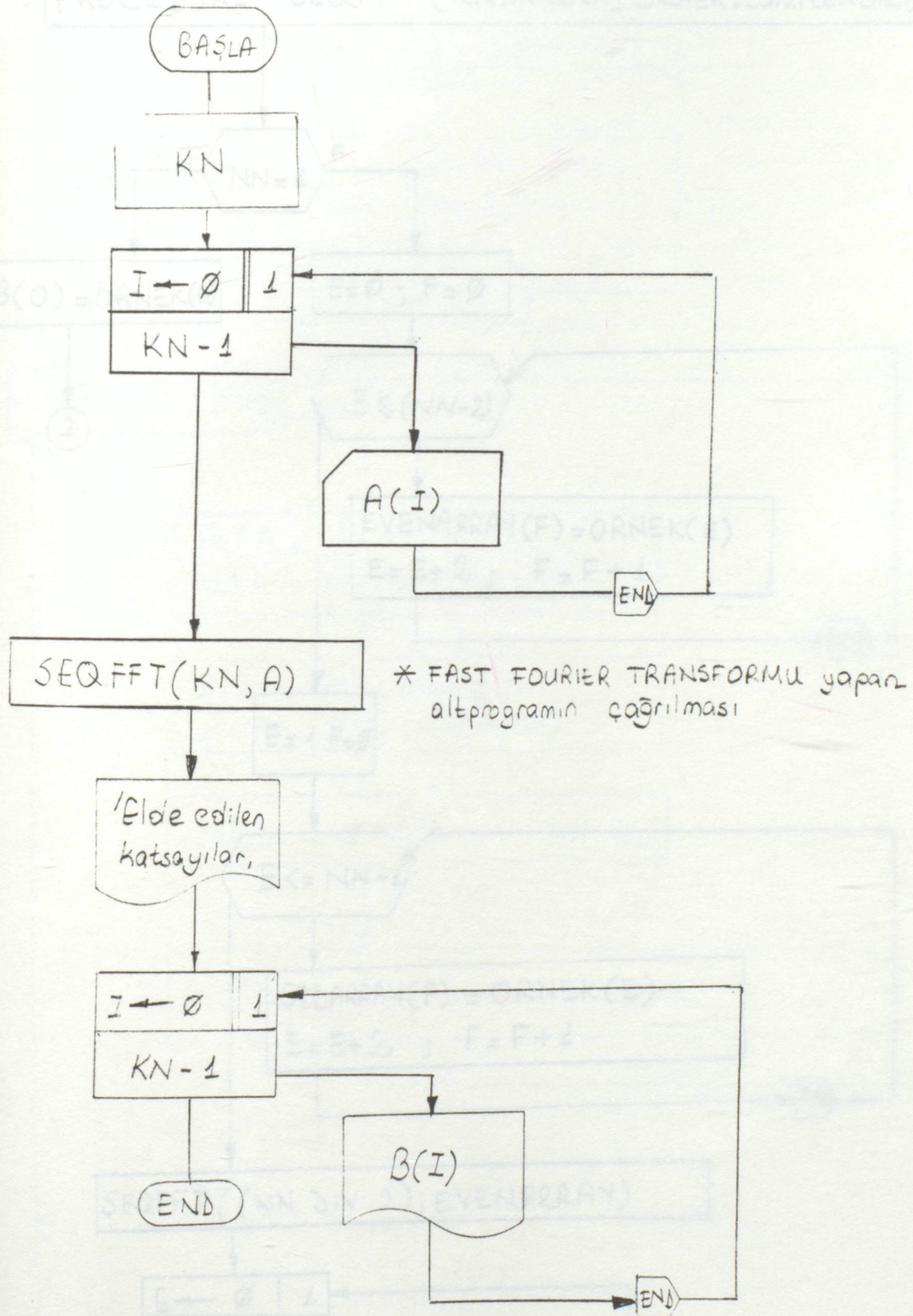
$$c(n) = O(n^{1/2})O(n) = O(n^{3/2}) \text{ dir.}$$

Procedure $O(n^{1/2} \log n)$ lik hızlanma sağlanmıştır ve efektifliği $O(\log n / n^{1/2})$ dir.

SIMDDFT ile karşılaştırılırsa MESHFFT daha yavaştır ve bundan dolayı SEQFFT'ye göre SIMDDFT'nin sağladığından daha az bir hızlanma sağlayabilir. öte yandan daha az işlemci kullanır, maliyeti düşük olup efektifliği fazladır.

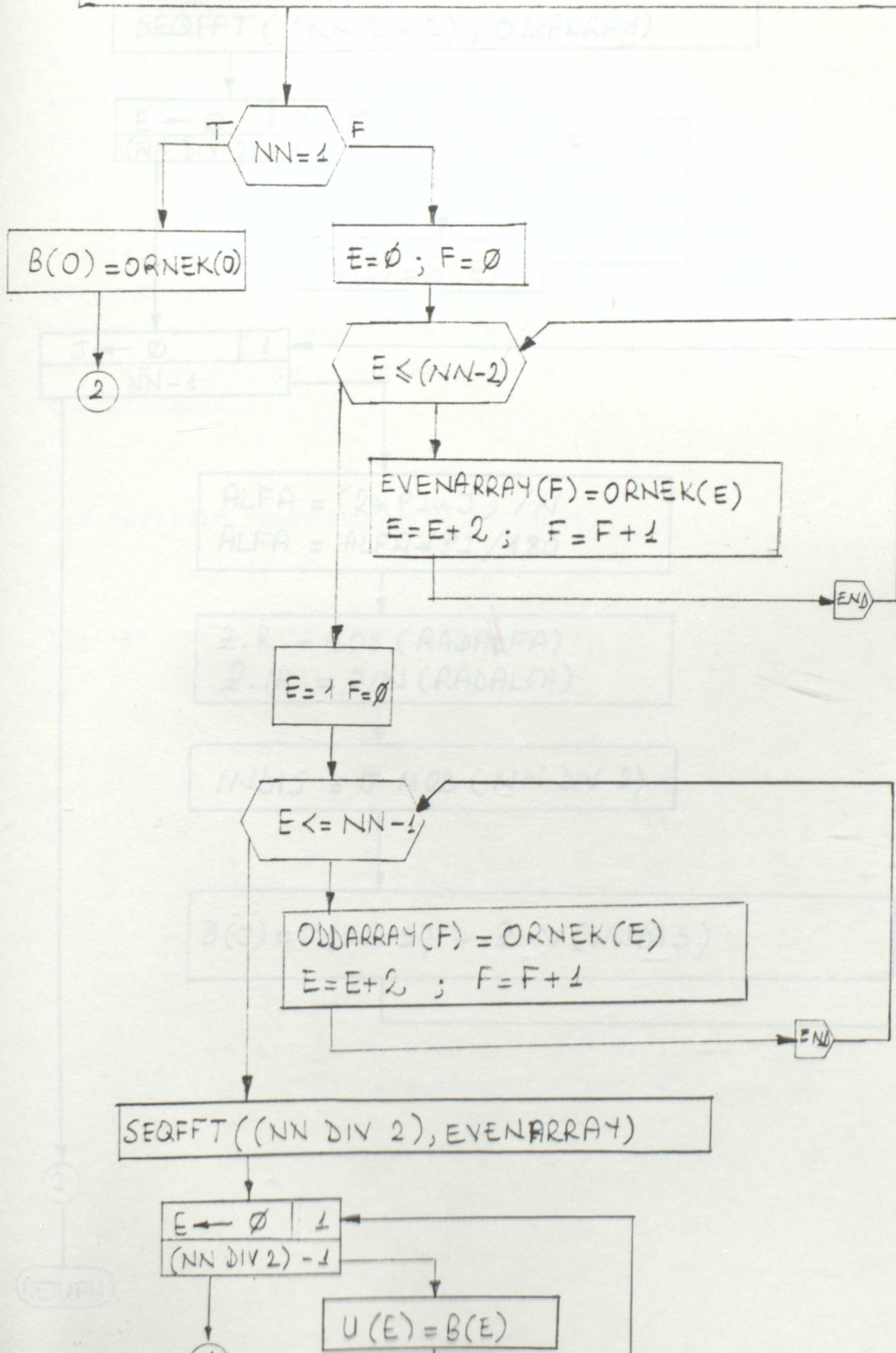


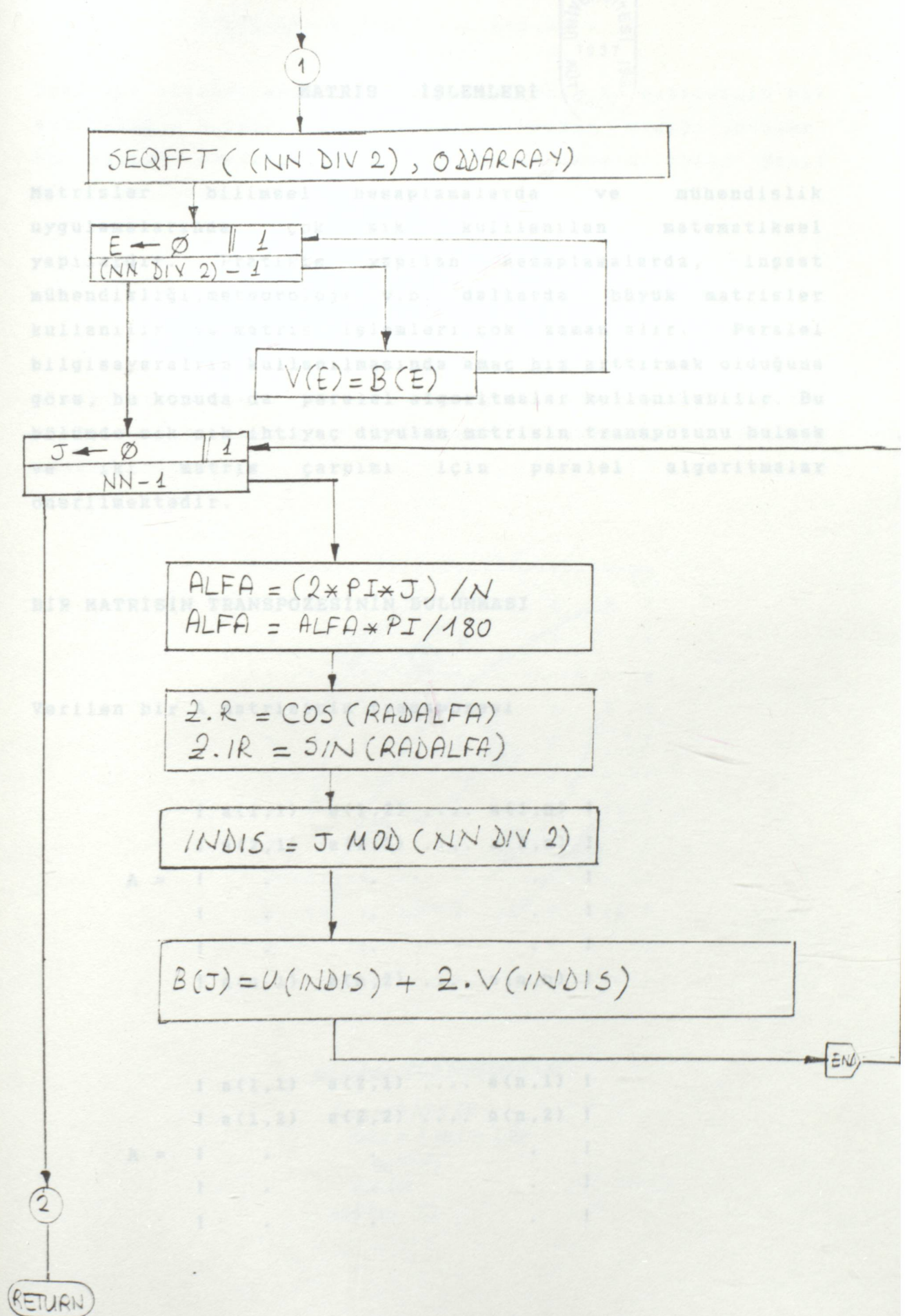
— ARDAŞIL HIZLI FOURIER DÖNÜŞÜMÜ —



SEQFFT Procedure 'ü

PROCEDURE SEQFFT (NN:INTEGER, ÖRNEK:COMPLEXDIZI)







Şekilde gösterilen MATRİS İŞLEMLERİ A matrisinin bir satırının A matrisinin bir sütunu halinde aldığı görülür. Bu işlemi gerçekleştiren seri algoritmanın şekli Şekil 10.1'de görülmektedir.

Matrisler ver bilimsel hesaplamalarda ve mühendislik uygulamalarında çok sık kullanılan matematiksel yapılardır. Pratikte yapılan hesaplamalarda, inşaat mühendisliği, meteoroloji v.b. dallarda büyük matrisler kullanılır ve matris işlemleri çok zaman alır. Paralel bilgisayarların kullanılmasında amaç hız arttırmak olduğuna göre, bu konuda da paralel algoritmalar kullanılabilir. Bu bölümde sık sık ihtiyaç duyulan matrisin transpozunu bulmak ve iki matris çarpımı için paralel algoritmalar önerilmektedir.

BİR MATRİSİN TRANSPOZESİNİN BULUNMASI

Verilen bir A matrisinin transpozesi

$$A = \begin{pmatrix} a(1,1) & a(1,2) & \dots & a(1,n) \\ a(2,1) & a(2,2) & \dots & a(2,n) \\ \vdots & \vdots & \ddots & \vdots \\ a(n,1) & a(n,2) & \dots & a(n,n) \end{pmatrix}$$

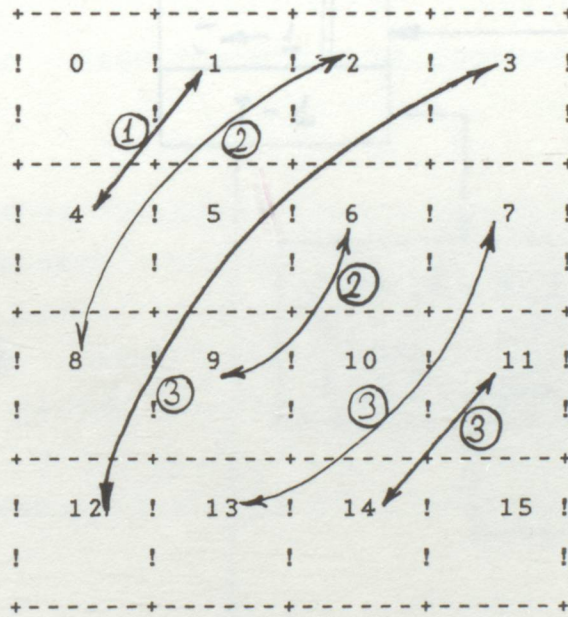
$$A^T = \begin{pmatrix} a(1,1) & a(2,1) & \dots & a(n,1) \\ a(1,2) & a(2,2) & \dots & a(n,2) \\ \vdots & \vdots & \ddots & \vdots \\ a(1,n) & a(2,n) & \dots & a(n,n) \end{pmatrix}$$

! a(1,n) a(2,n) a(n,n) !

Şeklinde gösterilebilir. Dikkat edilirse A matrisinin bir satırının A matrisinin bir sütunu halini aldığı görülür. Bu işlemi gerçekleştiren seri algoritmanın akışı Şekil 10.1' de verilmiştir.

10.1 ARDAŞIL MATRIS TRANSPOZU ALGORITMASI

A matrisinin satır ve sütun sayısı N ile gösterilirse N=4 için algoritmanın çalışması aşağıdaki örnekte gösterilmiştir.



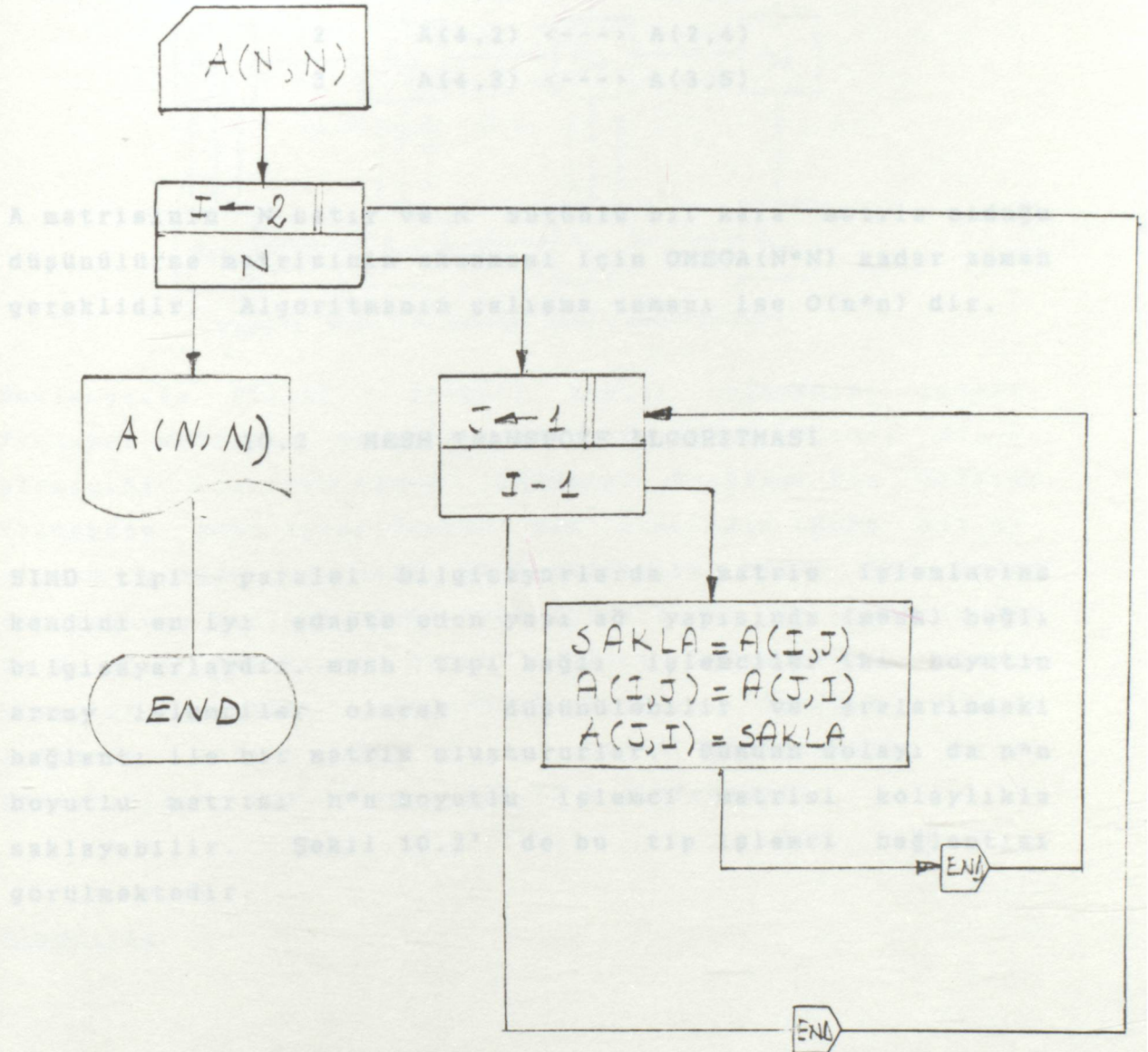
procedure TRANSPOSE (A)

```

for i = 2 to n do
  for j = 1 to i - 1 do
    aij ↔ aji
  end for
end for. □

```


- ARDAŞIL MATRIS TRANPOZU METODU -



	I	J	Yer deęiřtiren elemanlar
1. adım	2	1	$A(2,1) \longleftrightarrow A(1,2)$
2. adım	3	1	$A(3,1) \longleftrightarrow A(1,3)$
		2	$A(3,2) \longleftrightarrow A(2,3)$
3. adım	4	1	$A(4,1) \longleftrightarrow A(1,4)$
		2	$A(4,2) \longleftrightarrow A(2,4)$
		3	$A(4,3) \longleftrightarrow A(3,5)$

A matrisinin N satır ve N sütünlü bir kara matris olduęu düşünülürse matrisinin okunması için $OMEGA(N*N)$ kadar zaman gereklidir. Algoritmanın çalışma zamanı ise $O(n*n)$ dir.

Bařlangıçta $P(i,j)$. işlemci $A(i,j)$ elemanını ierir. Transpoz matr 10.2 MESH TRANSPOZE ALGORİTMAđı işlemci $A(i,j)$ elemanını ieriyor olur. $OMEGA(n)$ herhangi bir matrisi transpoze eden algoritmanın alt sınırıdaır $OMEGA(n)$.

SIMD tipi paralel bilgisayarlar da matris işlemlerine kendini en iyi adapte eden yapı aę yapısında (mesh) baęlı bilgisayarlardır. mesh tipi baęlı işlemciler iki boyutlu array işlemciler olarak düşünölebilir ve aralarındaki baęlantı ile bir matris oluřtururlar. Bundan dolayı da $n*n$ boyutlu matrisi $n*n$ boyutlu işlemci matrisi kolaylıkla saklayabilir. řekil 10.2' de bu tip işlemci baęlantısı görölmektedir.

Algoritma

Algoritmanın matrisi ödekte bařlıtır. A matrisinde A transpoz matrisine bařlıdır. diğeryanl matrisdeki elemanların aynı koldıkları göröür. $A(1,2)$ için A matrisinde $A(2,1)$ elemanı A transpoz $A(1,2)$ elemanına eşittir. diğeryanl altındaki elemanlar da aynı

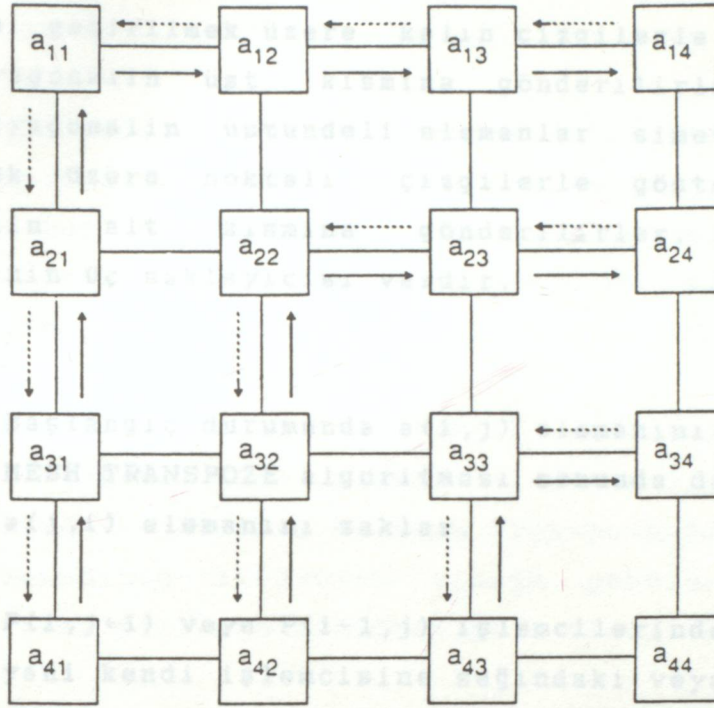


Figure 7.1 Matrix to be transposed, stored in mesh of processors.

Başlangıçta $P(i,j)$. işlemci $a(i,j)$ elemanını içerir. Transpoz matrisin oluşmasından sonra aynı işlemci $a(j,i)$ elemanını içeriyor olur. OMEGA(n) herhangi bir matrisi transpoze eden algoritmanın alt sınırındır çünkü $a(1,n)$ eleman gitmesi gereken yere gidinceye kadar iki kenar uzunluğunun toplamı kadar yok dolaşır. Bu da $(n-1)+(n-1)=2n-2$ adım demektir. Bunun nedeni işlemcilerin sadece komşularına bilgi gönderebilmeleridir. Örneğin $P(1,4)$ işlemcisi sadece $P(1,3)$ ve $P(2,4)$ işlemcisine bilgi gönderebilir; $P(2,3)$ veya $P(4,1)$ e gönderemez.

Algoritma

Algoritmanın mantığı oldukça basittir. A matrisine ve A transpoz matrisine bakılırsa diyagonal üzerindeki elemanların aynı kaldıkları, görülür. $i=1,2,\dots,n-1$ için A matrisinin $A(i,i)$ elemanı A transpozun $A(i,i)$ elemanına eşittir. Diyagonalin altındaki elemanlar simetrik

pozisyona getirilmek üzere kalın çizgilerle gösterildiği gibi diyagonalin üst kısmına gönderilirler. Eşzamanlı olarak Diyagonalin üstündeli elemanlar simetrik pozisyona getirilmek üzere noktalı çizgilerle gösterildiği gibi diyagonalin alt kısmına gönderilirler. Her $P(i,j)$ işlemcisinin üç saklayıcısı vardır. Bu şekilde kesikli çizgiler ile gösterilmektedir.

$A(i,j)$: Başlangıç durumunda $a(i,j)$ elemanını, saklar. Bunların MESH TRANSPÖZE algoritması sonunda da olmak üzere $(2n-2)$ kere $a(j,i)$ elemanını saklar. Transpose Algoritması'nın çalışma zamanının $t(n)=O(n^2)$ olduğu görülmüştür. Kullanılan $B(i,j)$: $P(i,j+1)$ veya $P(i-1,j)$ işlemcilerinden, maliyeti yani kendi işlemcisine sağındaki veya $P(i,j)$ yukarısındaki işlemcilerden gelen bilgiyi tutar. $n) = O(n^2)$

$C(i,j)$: $P(i,j-1)$ veya $P(i+1,j)$ işlemcilerinden, Bu maliyet yani kendi işlemcisine solundaki veya $P(i,j)$ varabilen aşağısındaki işlemcilerden gelen bilgiyi tutar. $n) = O(n^2)$

Algoritmanın Çalışması :

$i > j$ şartını sağlayan her $a(i,j)$ elemanı bulunduğu sütun içinde satır numarasını azaltarak $P(j,j)$ işlemcisine kadar gider. Buraya varınca i . satır içinde sütun numarasını arttırarak $P(j,1)$ işlemcisine varıncaya kadar sağa doğru hareketine devam eder. Böylece matris içinde önce yukarı sonra sağa hareket tamamlanmış olur. Örnek olarak $P(3,1)$ işlemcisi $a(3,1)$ değerini $P(1,1)$ sonra da $P(1,3)$ e doğru hareket ettirir. $P(1,1)$ işlemcisine gelince yön değiştirir ve $P(1,1)$ sonra da $P(1,2)$ yolundan giderek $P(1,3)$ 'e yerleşir. Bu Şekil 10.2' de kalın çizgiler ile

gösterilmiştir. 3 SHUFFLE TRANSPOZE ALGORİTMASI

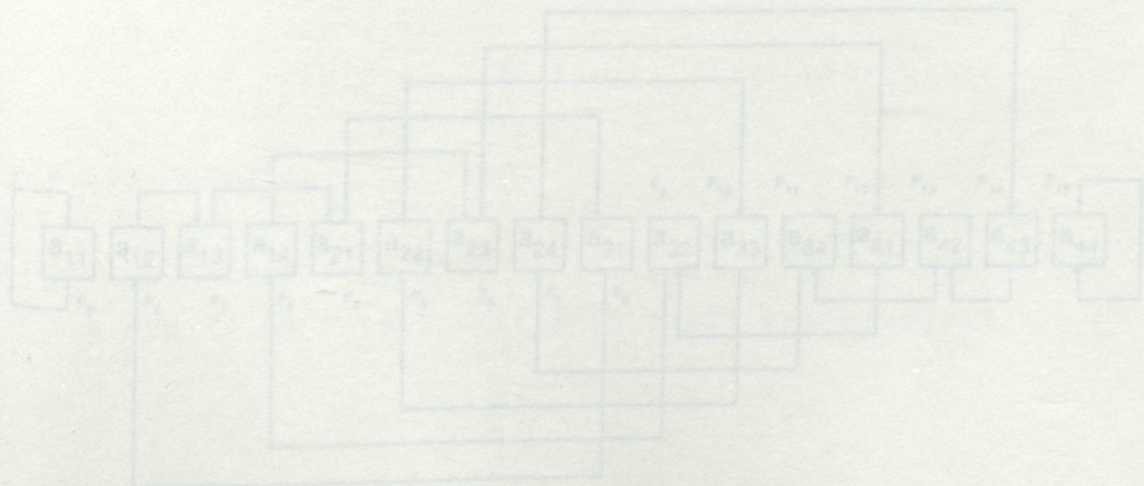
$j > i$ şartını sağlayan $a(i, j)$ elemanı bulunduğu satır içinde sütun numarasını azaltarak $P(i, i)$ işlemcisine kadar gider. sonra da yön değiştirerek satır numarasını azaltarak $P(j, i)$ ye kadar hareketine devam eder. Bu şekilde kesikli çizgiler ile gösterilmiştir.

Bu metoda göre $a(1, n)$ ve $a(n, 1)$ işlemcileri alırlar. Bunların aldıkları yol sayısı $(n-1) + (n-1)$ olmak üzere $(2n-2)$ kadardır. Buradan da Mesh Transpoze Algoritması'nın çalışma zamanının $t(n) = O(n)$ olduğu görülür. Kullanılan işlemci sayısı $p(n) = n \times n$ olduğuna göre algoritmanın maliyeti

$$c(n) = t(n) * p(n) \\ = O(n) * O(n^2)$$

$$c(n) = O(n^3) \text{ olarak bulunur.}$$

Bu fiyatla algoritma optimal olmaktan uzaktır sonucuna varabiliriz. Şekil 10.3'de Mesh Transpoze Algoritmasının akış diyagramı görülmektedir.



Bu algoritma 10.3 SHUFFLE TRANSPOZE ALGORİTMASI $P(k)$, $a(i,j)$ elemanları $P(k)$ transpoz matrisinin elemanları içerir. Bunu görmek için şunu hatırlayalım. $P(k)$ işlemcisi $P(k)$ Mesh Transpoze algoritmasının $n \times n$ boyutlu matrisin transpozunu $O(n)$ zamanda bulur. Bu, ağ yapısında işlemciler kullanıldığında ve her işlemci matrisin bir elemanını sakladığında elde edilebilecek olan en iyi çalışma hızıdır. Bir matrisin transpozunu ardışıl bir algoritma ile $O(n^2)$ zamanda bulabiliriz. İki durum karşılaştırılırsa Mesh Transpoze algoritmasının sağladığı hızlanma lineerdir. Bu hızlanma procedüre 2 'nin tamkatı kadar işlemci kullandığı için oldukça azdır. Shuffle Transpoze Algoritması'nda aynı P sayıda işlemci farklı bir geometrik yapıda düzenlenirse çalışma hızının nasıl logaritmik bir zamanda nasıl biteceğini göreceğiz.

$n=9$ için $A(n \times n)$ boyutlu bir matris ise $n \times n$ işlemcili PERFECT SHUFFLE Metodu ile arabağlantıları yapılan bir ağ yapısı kullanılacaktır.

$k = n \cdot \exp(2, q)$ olmak şartıyla
 $n \cdot n = \exp(2, 2 \cdot q)$
 $P(n) = \exp(2, 2 \cdot q)$ işlemci kullanılmakta ise
 için $k = \exp(2, q) \cdot (i-1) + (j-1)$ olmak üzere

İlk durumda A matrisinin $a(i,j)$ elemanı $P(k)$ işlemcide saklanır. Örnek olarak $q=2$ ise $n=4$ olarak $P(n)=16$ olarak bulunur.

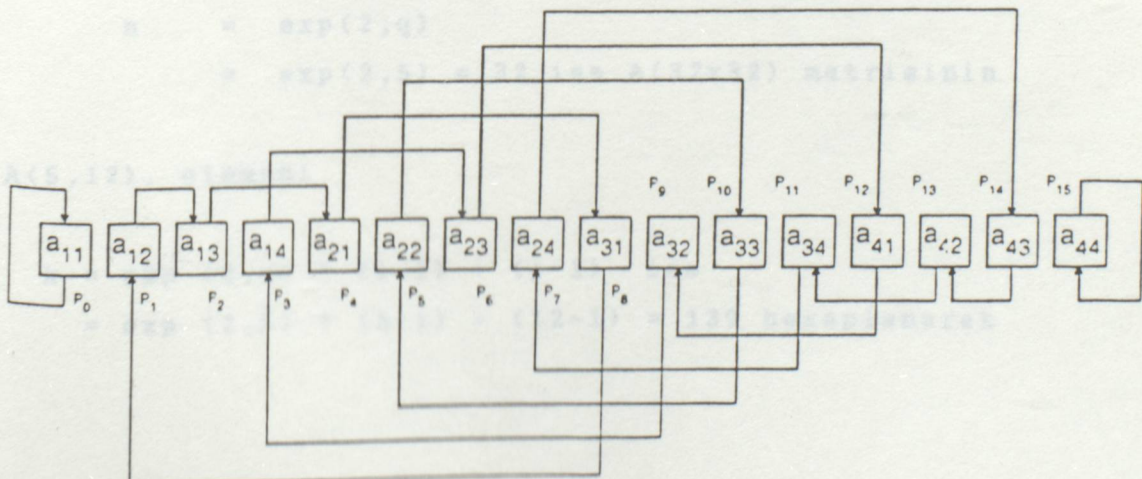


Figure 7.3 Matrix to be transposed, stored in perfect shuffle-connected computer.

Bu algoritmada q adet karşılaştırmadan sonra $P(k)$, $a(j,i)$ elemanını yani A transpoze matrisinin elemanını içerir. Bunu görmek için şunu hatırlayalım. $P(k)$ işlemcisi $P(m)$ işlemcisine P bağlı ise m sayısı k sayısından elde edilebilir. Söyle ki k sayısının binary gösterilişi 1 bit sola ötelenir. Bu nedenle Perfect Shuffle yapısında $P(0)$ işlemcisi 0 kendine bağlıdır. Diğer bağlı işlemciler şöyledir.

$S = \exp(2, q) * (i-1) + (j-1)$ formülüyle hesaplanır.

yukarı $P(0000) \langle \text{----} \rangle P(0000)$

$P(0001) \langle \text{----} \rangle P(0010)$

$S = P(0010) \langle \text{----} \rangle P(0100) (5-1) = 356$ olarak bulunur.

$P(1001) \langle \text{----} \rangle P(0011)$

Demek ki $P(1010) \langle \text{----} \rangle P(0101)$ işleminin sonucu $P(356)$.

İşlem $P(1111) \langle \text{----} \rangle P(1111)$ in sağlanmasını yapmak için geri döndürülürse $P(356)$ işlemcisi ilk verilen A matrisinde

k . numarası ile belirtilen bir işlemcinin $2q$ adet bite sahip olduğunu

$k = \exp(2, q) * (i-1) + (j-1)$ ise

$= \exp(2, 5) * (12-1) + (5-1) = 259$ bulunur.

k 'nın q adet en çok anlamlı bitini (MSB) $(i-1)$; q tane en az anlamlı bitini (LSB) $(j-1)$ gösterir. $q=5$ ise $i=5$ $j=12$ için durum incelenirse

hesaplanabilir. Şekil 10.4'de Perfect Shuffle

TRANS $p(n) = \exp(2, 2*q)$ olarak verilmektedir.

$= \exp(2, 10) = 1024$ işlemci kullanıldığı

$n = \exp(2, q)$

$= \exp(2, 5) = 32$ ise $A(32 \times 32)$ matrisinin

$A(5, 12)$. elemanı

$k = \exp(2, q) * (i-1) + (j-1)$ ise

$= \exp(2, 5) * (5-1) + (12-1) = 139$ hesaplanarak

İlk durumda A matrisinin $a(5,12)$ elemanının $P(139)$ işlemcide saklandığı sonucuna varılır. $q=5$ adet karşılaştırmadan sonra $P(139)$ $a(12,5)$ elemanını içeriyor olacaktır. $P(139) \cdot 2 \cdot q = 10$ tane bite sahipse bu 10 bitin MSB $q=5$ biti $(i-1)=4$ 'ü LSB $q=5$ biti $(j-1)=11$ 'i gösterir. $q=5$ defa sola öteleme yapıldıktan sonra eleman $P(k)$ işlemciye gelir bunun indisi ise

$$S = \exp(2, q) * (j-1) + (i-1) \text{ formülüyle hesaplanır.}$$

yukarıdaki örnek için S sayısı

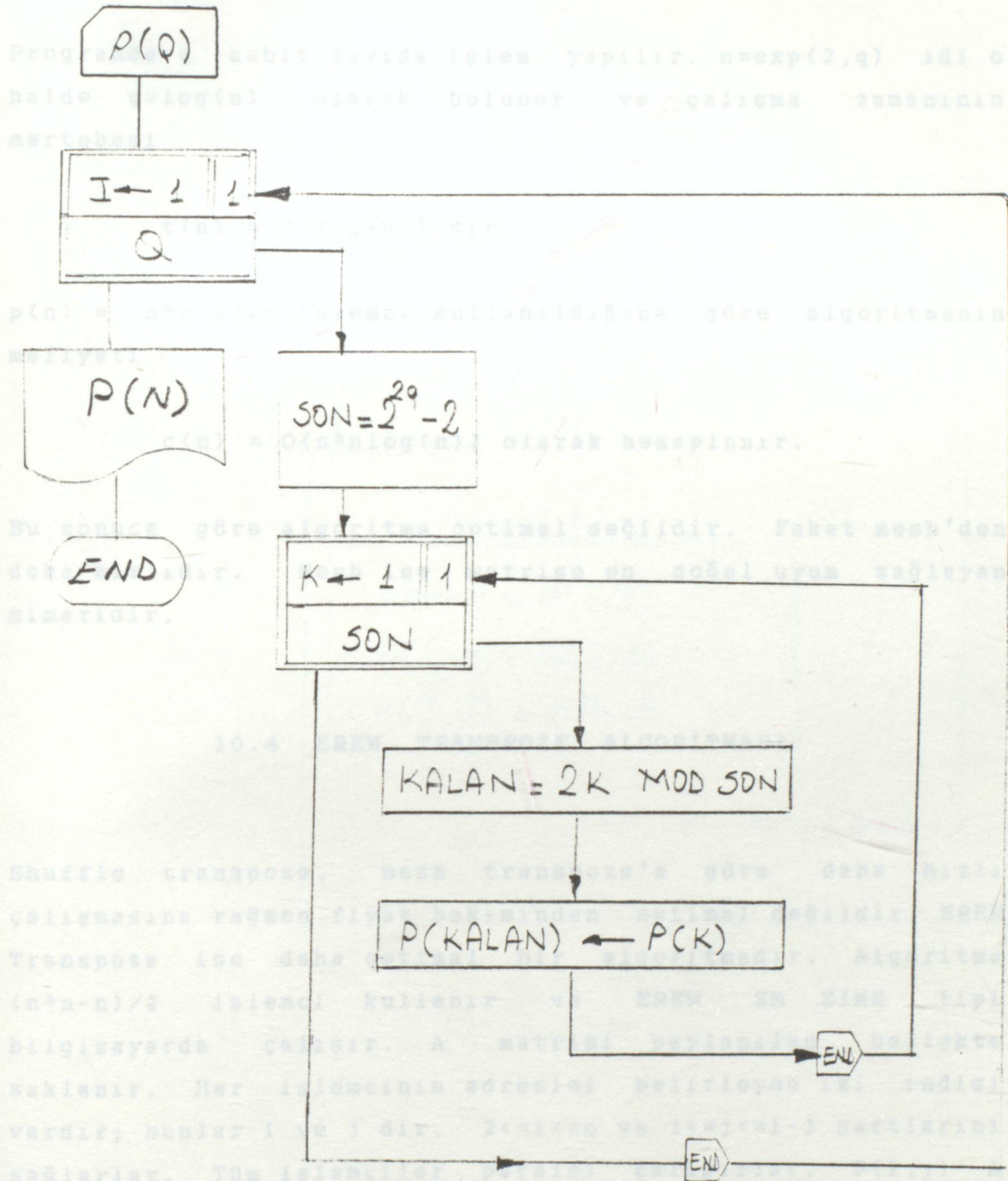
$$S = \exp(2, 5) * (12-1) + (5-1) = 356 \text{ olarak bulunur.}$$

Demek ki $A(5,12)$ transpoz işleminden sonra $P(356)$ işlemcide olacaktır. İşlemin sağlamasını yapmak için geri dönülürse ; $P(356)$ işlemcisi ilk verilen A matrisinde $A(12,5)$ elemanını tutmaktadır. $q=5$ ve $i=12$ için gerçekten de

$$\begin{aligned} k &= \exp(2, q) * (i-1) + (j-1) \text{ ise} \\ &= \exp(2, 5) * (12-1) + (5-1) = 356 \text{ bulunur.} \end{aligned}$$

Algoritmanın avantajı transpoze işlemi başladığında her bir elemanın hangi işlemcide bulunduğu basit bir formülle hesaplanabilmesidir. Şekil 10.4' de PERFECT SHUFFLE TRANSPOZE algoritmasının akış diyagramı verilmiştir.

SHUFFLE TRANSPOZ METODU



ANALİZ:

Programda q sabit sayıda işlem yapılır. $n = \exp(2, q)$ idi o halde $q = \log(n)$ olarak bulunur ve çalışma zamanının mertebesi

$$t(n) = O(\log(n)) \text{ dir.}$$

$p(n) = n \cdot n$ adet işlemci kullanıldığına göre algoritmanın maliyeti

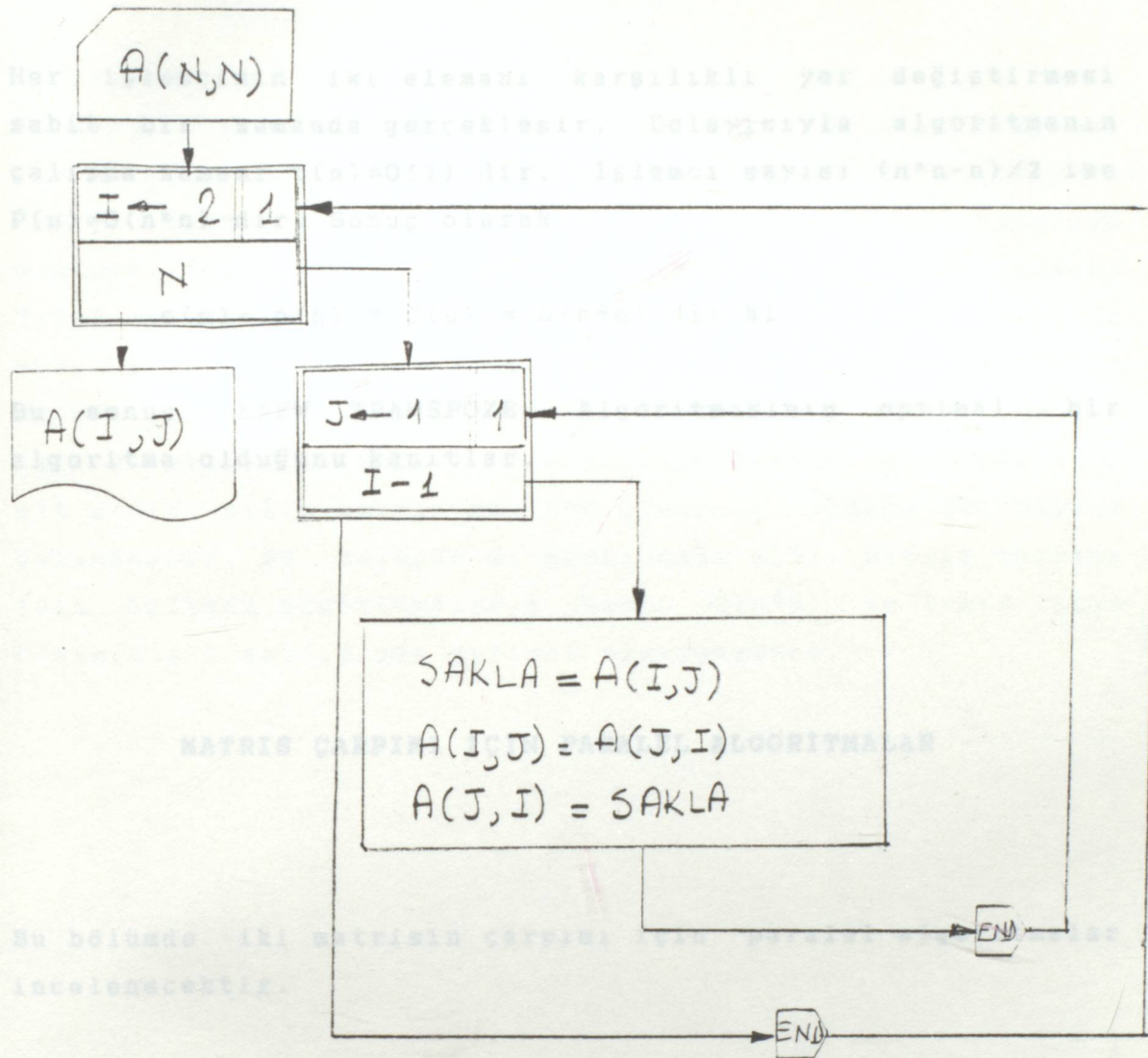
$$c(n) = O(n \cdot n \log(n)) \text{ olarak hesaplanır.}$$

Bu sonuca göre algoritma optimal değildir. Fakat mesh'den daha hızlıdır. Mesh ise matrise en doğal uyum sağlayan mimaridir.

10.4 EREW TRANSPÖZE ALGORİTMASI

Shuffle transpoze, mesh transpoze'a göre daha hızlı çalışmasına rağmen fiyat bakımından optimal değildir. EREW Transpoze ise daha optimal bir algoritmadır. Algoritma $(n \cdot n - n)/2$ işlemci kullanır ve EREW SM SIMD tipi bilgisayarda çalışır. A matrisi paylaşılan bellekte saklanır. Her işlemcinin adresini belirleyen iki indisi vardır; bunlar i ve j dir. $2 \leq i \leq n$ ve $1 \leq j \leq i-1$ şartlarını sağlarlar. Tüm işlemciler paralel çalışırlar. $P(i, j)$ A matrisinin $A(i, j)$ ve $A(j, i)$ elemanlarını karşılıklı yer değiştirir. Algoritmanın akış diyagramı Şekil 10.5'de verilmiştir.

- EREW TRANSPOZ METODU -



10.5 ARKADIL MATRİS ÇARPIMI ALGORİTMA

A matrisi $m \times n$ boyutlu, B matrisi $n \times k$ boyutlu olarak kabul edilmelidir. C matrisi $m \times k$ boyutlu olarak tanımlanır.

$$C = A \cdot B \text{ ise } C(i,j) \text{ elemanı}$$

ANALİZ:

Her işlemcinin iki elemanı karşılıklı yer değiştirmesi sabit bir zamanda gerçekleşir. Dolayısıyla algoritmanın çalışma zamanı $t(n)=O(1)$ dir. İşlemci sayısı $(n*n-n)/2$ ise $P(n)=O(n*n)$ dir. Sonuç olarak matris çarpma algoritmasının $O(n^3)$ zamanında çalıştığı görülmüştür. Çalışma zamanı 2×3 için $c(n) = p(n) * O(n) = O(n^3)$ dir ki burada çalışan her matris çarpma algoritması da vardır. Bununla birlikte Bu sonuç EREW TRANSPOZE Algoritmasının optimal bir algoritma olduğunu kanıtlar. Sınırının $O(n^3)$ olduğu, bu alt sınır çıkış matrisinin n^3 elemanı olduğu görülmüş bulunmaktadır. Bu bölüme de görüleceği gibi matris çarpma için bulunan algoritmaların hepsi $O(n^3)$ ve 2×3 için $O(\exp(n, x))$ aralığında optimal olmaktadır.

MATRİS ÇARPIMI İÇİN PARALEL ALGORİTMALAR

Bu bölümde iki matrisin çarpımı için paralel algoritmalar incelenecektir.

10.5 ARDAŞIL MATRİS ÇARPIMI ALGORİTMASI

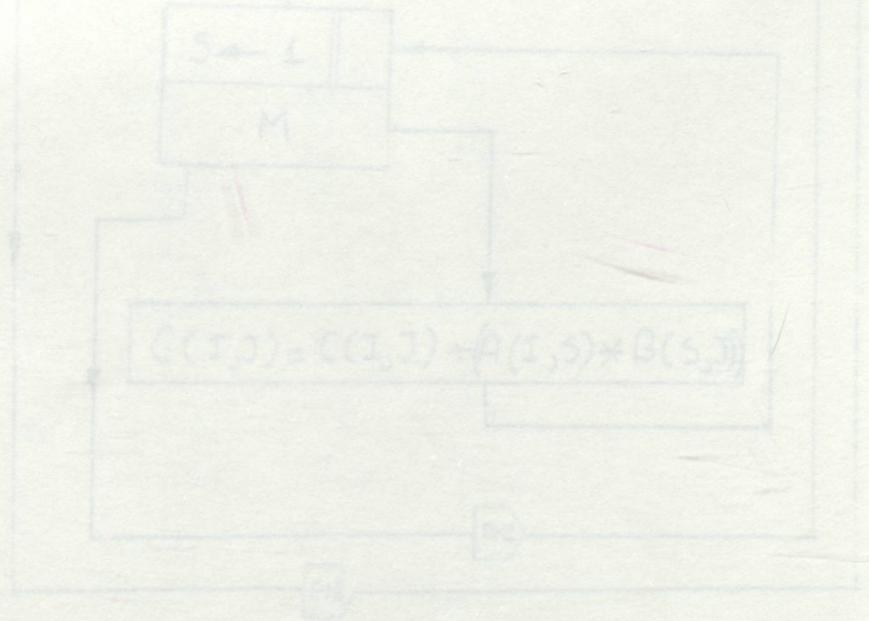
A matrisi $m \times n$ boyutlu, B matrisi $m \times k$ boyutlu olarak kabul edilmiştir. C sonuç matrisin $m \times k$ boyutlu olacağı aşıkardır.

$$C = A * B \text{ ise } C(i, j) \text{ elemanı}$$

ARDAŞIL MATRİS ÇARPIMI METODU

$$C(i,j) = \sum_{s=1}^n A(i,s) * B(s,j) \quad 1 \leq i \leq m, 1 \leq j \leq k,$$

Bu ardışıl programın akış diyagramı Şekil 10.6'da Programı ise MATMUL.PAS olarak Ek. te bulunmaktadır. $m \leq n$ ve $k \leq n$ olduğu düşünülürse seri matris çarpımı algoritmasının $O(n*n*n)$ zamanında çalıştığı aşıkardır. Çalışma zamanı $2 < x < 3$ olmak şartıyla $O(\exp(n,x))$ karmaşıklığında çalışan seri matris çarpımı algoritmaları da vardır. Bunların hangisinin daha efektif olduğu tartışılabilir ama matris çarpımı algoritmasının alt sınırının $O(n*n)$ olduğudur. bu alt sınır çıkış matrisinin $n*n$ elemanlı olduğu gözlenerek bulunmuştur. Bu bölümde de görüleceği gibi matris çarpımı için bulunan algoritmaların hepsi $O(n*n)$ ve $2 < x < 3$ için $O(\exp(n,x))$ aralığında optimal olmayacaktır.



ARDAŞIL MATRIS ÇARPIMI METODU -

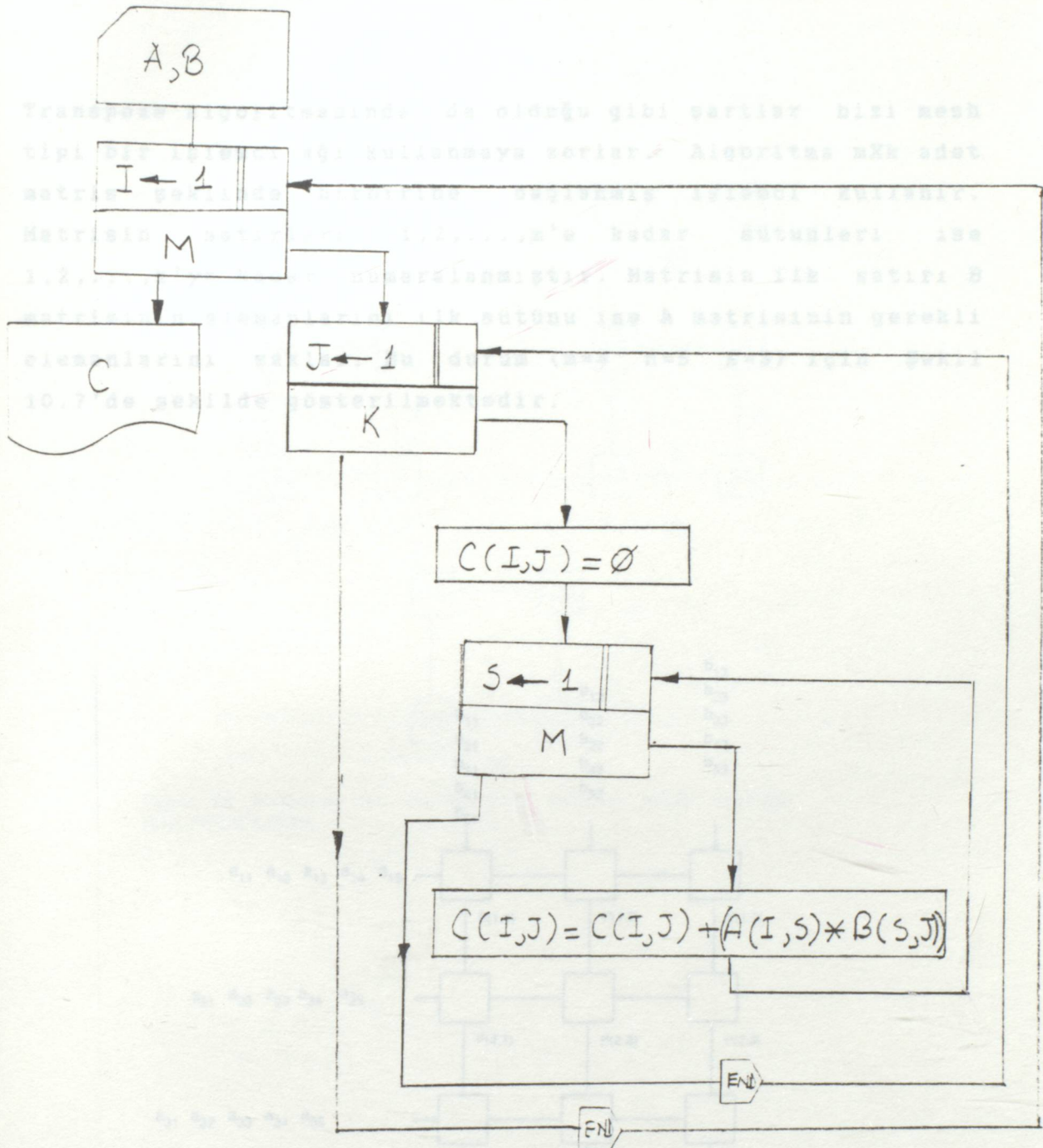


Figure 12 - Two matrices to be multiplied, being fed as input to each of processors

MESH MATRİS ÇARPIMI METODU

Transpoze algoritmasında da olduğu gibi şartlar bizi mesh tipi bir işlemci ağı kullanmaya zorlar. Algoritma $m \times k$ adet matris şeklinde birbirine bağlanmış işlemci kullanır. Matrisin satırları $1, 2, \dots, m$ 'e kadar sütunları ise $1, 2, \dots, k$ 'ya kadar numaralanmıştır. Matrisin ilk satırı B matrisinin elemanlarını ilk sütunu ise A matrisinin gerekli elemanlarını saklar. Bu durum ($m=4$ $n=5$ $k=3$) için Şekil 10.7'de şekilde gösterilmektedir.

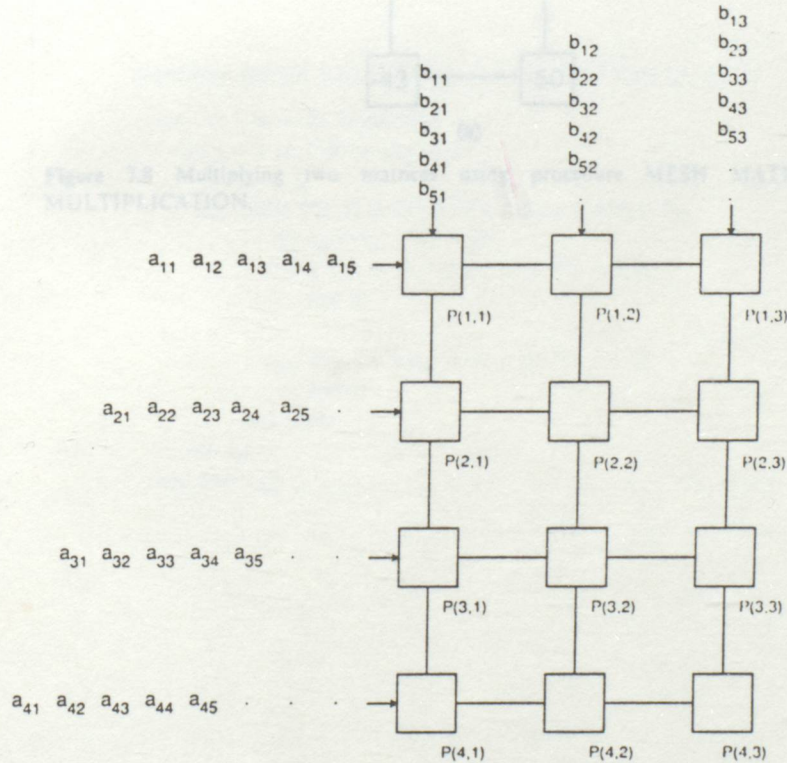


Figure 7.7 Two matrices to be multiplied, being fed as input to mesh of processors.

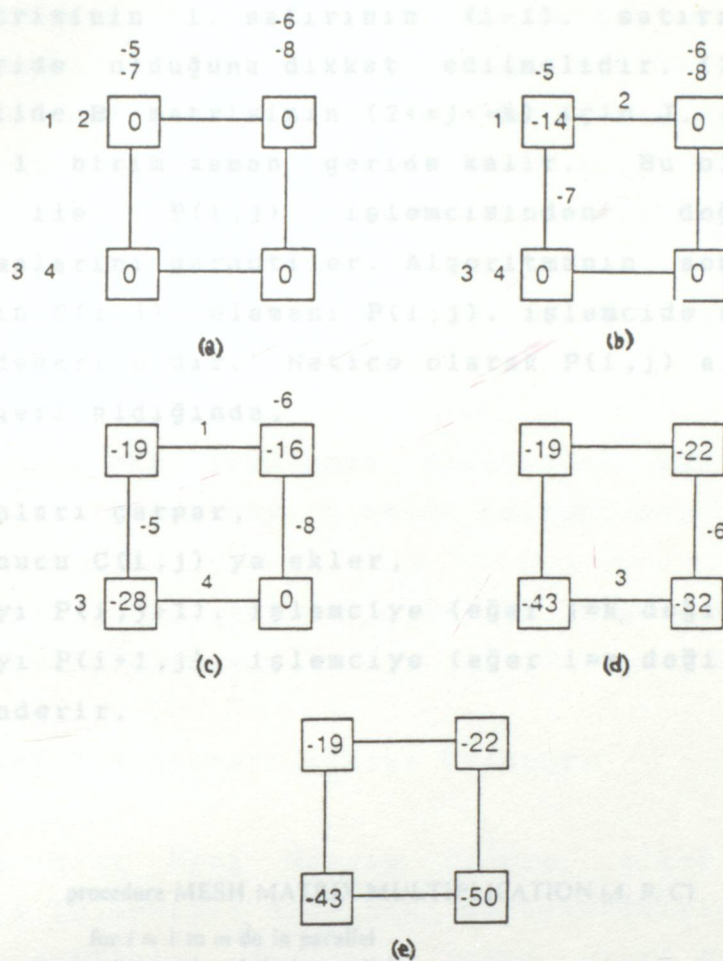


Figure 7.8 Multiplying two matrices using procedure MESH MATRIX MULTIPLICATION.

$A(m,1)$ ve $B(m,k)$ elemanlarının, işlem başında bulundukları yerden $P(m,k)$. işlemciye erişimleri $m+k+n-2$ uadım sürer. $P(m,k)$ en son işlemci olduğu için algoritmanın bitip sonucun elde edilmesi için bu kadar adım işlemin mutlaka yapılması gerekir. $m \leq n$ ve $k \leq n$ şartlarının sağlanması durumunda çalışma zamanı

$$t(n) = O(n) \text{ dir.}$$

Sonuç matris $m \times k$ boyutunda olduğundan $m \times k$ tane işlemci kullanılması zorunludur. O halde işlemci sayısı birbirinden farklı olduğu görüldü. Küp bağlantısının özelliği tüm $0 \leq i \leq g$ değerleri $p(n) = O(n^2)$ dir. Bu iki değerden algoritmanın maliyeti hesaplanırsa $c(n) = O(n^2)$ olarak bulunur.

görüldüğü gibi Mesh Matris Çarpımı Algoritması matris bağlantılı işlemciler üzerinde gerçekleştirilebilen en iyi algoritma değildir. Sadece 1. (ilk) satır ve 1. sütundaki elemanlar giriş çıkış işlemleri için kullanılabilirler. Bu şartlar altında giriş için alt sınır olan $OMEGA(n)$ ve benzer olarak çıkış işlemi için de alt sınır olan $OMEGA(n)$ işleminin gerekmektedir.

10.7 KÜP YAPISINDA BAĞLI (CUBE CONNECTED) ve sütun SIMD BİLGİSAYARDA MATRİS ÇARPIMI İŞLEMLERİ

Matris çarpımını bu sefer işlemcileri birbirine küp şeklinde bağlanmış bir bilgisayarda inceleyelim. Bu durumda $g \geq 1$ için $N = \exp(2,g)$ olmak şartıyla N adet işlemci $P(0), P(1), \dots, P(2^g-1)$ olarak kullanılmaktadır. i ve ib tamsayılar olmak üzere

$r(3 \cdot g - 2) \cdot 0 \leq i \leq r$ sayısının binary ifadesi
 $ib \leq \exp(2, g) - 1$ şartını sağlar ve bu iki sayının binary ifadeleri yazılırsa $0 \leq b \leq g$ olmak şartıyla sadece b . bitlerinin birbirinden farklı oldukları görülür. Başka bir deyişle:

$i(g-1) \dots i(b+1) i(b) i(b-1) \dots i(1) i(0)$ i sayısının

$i(g-1) \dots i(b+1) i'(b) i(b-1) \dots i(1) i(0)$ ib sayısının binary ifadesi olursa sadece b . bitlerinin birbirinden farklı olduğu görülür. Küp bağlantısının özelliği tüm $0 \leq b \leq g$ değerleri için P_i işlemcisinin $P_i(b)$ işlemcisine iki yöllü bir yol ile bağlanmasıdır. P_i ye bağlı g adet işlemciye bu işlemcinin komşuları denir. k değeri de sabit olmaktadır. Bu değer şu formülle hesaplanır.

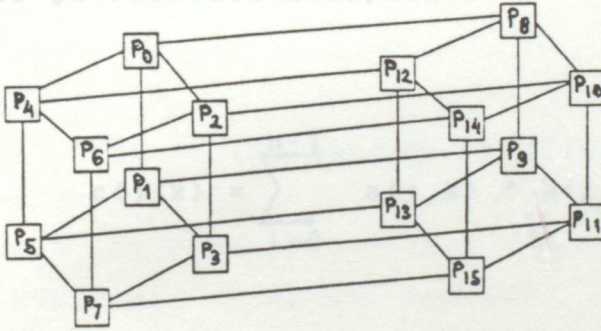


Figure 7.9 Cube-connected computer with sixteen processors.

Şekil 10.8' de $g=4$ için küp bağlantılı bilgisayar verilmiştir. $n = \exp(2, g)$ olsun; Algoritmada $n \times n$ boyutlu iki matrisi çarpmak için $N = n \times n \times n = \exp(g, 3)$ adet işlemci kullanacağız. İfadesi kolay olsun diye iki matrisin satır ve sütun sayılarının birbirine eşit olduğunu kabul edeceğiz. İşlemciler $n \times n \times n$ boyutlu bir küp şeklinde organize edilmişlerdir. bir $P(r)$ işlemcisinin küpteki yeri $P(i, j, k)$ notasyonu ile gösterilir.

$$r = i \cdot n \cdot n + j \cdot n + k \quad 0 \leq i, j, k \leq n \text{ dir.}$$

Buna satır esaslı (Row Major) sıralı denir.

$r(3*q-1) \ r(3*q-2) \ \dots \ r(0)$ r sayısının binary ifadesi

$r(3*q-1) \ \dots \ r(2q)$ i sayısının binary ifadesi

$r(2*q-1) \ \dots \ r(q)$ j sayısının binary ifadesi

$r(2*q-1) \ \dots \ r(0)$ k sayısının binary ifadesi

olarak kabul edilmiştir. Her $P(r)$ işlemcisi 3 saklayıcıya sahiptir. Bunlar $A(i,j,k)$ $B(i,j,k)$ ve $C(i,j,k)$ dır. İlk durumda $0 \leq j \leq n$ $0 \leq k \leq n$ olmak koşuluyla $(0,j,k)$ konumunda bulunan P_s işlemcisi A_s saklayıcısından $a(j,k)$; B_s saklayıcısından $b(j,k)$ değerlerini alır. Diğer tüm işlemcilere ilk değer olarak 0 verir. Hesaplamanın sonucunda C saklayıcısı $c(j,k)$ değerine sahip olacaktır. Bu değer şu formülle hesaplanır.

$$c(j,k) = \sum_{i=0}^{n-1} a(j,k) * b(i,k)$$

Daha önce de incelendiği gibi matris çarpımı algoritmaları $n*n*n$ adım hesaplama gerektirir. Bu algoritma C 'nin $n*n$ değeri için bu sayıda işlemi paralel, eşzamanlı olarak yapar. İşlemler 3 safhadan oluşur.

1. ADIM:

A ve B matrisinin elemanları $A(i,j,k)=a(i,j)$ ve $B(i,j,k)=b(i,k)$ olacak şekilde n küp işlemci arasında dağıtılır.

2. ADIM:

$C(i,j,k) = A(i,j,k) * B(i,j,k)$ çarpımı hesaplanır. (i,j,k)

3. ADIM:1. ADIM:

2. ad. $c(j,k) = \sum_{i=0}^{n-1} c(i,j,k)$ toplamı bulunur. $c(i,j,k)$

Bu işlemin yarı kodlanmış durumu aşağıda verilmiştir.

4. ADIM:

(i,j,k) pozisyonundaki işlemci tarafından $0 \leq i,j,k < n$ için tüm işlemciler paralel olarak $C(i,j,k) = A(i,j,k) * B(i,j,k)$ çarpımı hesaplanır. hesaplamaları yepiye.

ref 1 sh 183/184 program fotokopısı

5. ADIM:

$(N, rm = d)$ kümesi $0 \leq r \leq N-1$ olmak üzere r tamsayılarının bir kümesidir ve binari ifadesi

$r(3*q-1) \dots r(m+1) d r(m-1) \dots r(0)$ dir.
si $(0,j,k) = \sum_{i=0}^{n-1} c(i,j,k)$ toplamı bulunur.

Algoritma başlıca 5 safhadan oluşmaktadır. 1. safhası 1. ve 3. adımlardan oluşur.

1. ADIM:ANALİZ

$A(0,j,k)$ ve $B(0,j,k)$ da ilk değer olarak bulunan sayılar $0 \leq i \leq n$ olmak üzere (i,j,k) adreslerindeki işlemcilere atanır. Bu adımın sonunda $A(i,j,k) = a(i,j)$ ve $B(i,j,k) = b(i,k)$ olur.

2. ADIM: $0 \leq k \leq n$ olmak üzere $A(i, j, i)$ işlemcinin verisi (i, j, k) pozisyonundaki işlemcilere atanır. Böylece $0 \leq i \leq n$ olmak üzere $A(i, j, k) = a(j, i)$ olur.

3. ADIM: $\log(n)$ olarak bulunur.

2. adıma benzer olarak $0 \leq j \leq n$ olmak üzere $b(i, j, k)$ işlemcinin verisi (i, j, k) pozisyonundaki işlemcilere atanır. Böylece $0 \leq j \leq n$ olmak üzere $B(i, j, k) = b(i, k)$ değerini içerir.

4. ADIM: $\log(n)$ dir. n göre $\log(n)$ için n adet sayının toplamının en az S sayısı kadar adımda gerçekleştirildiği (i, j, k) pozisyonundaki işlemci tarafından $0 \leq i, j, k \leq n$ için tüm işlemciler paralel olarak $C(i, j, k) = A(i, j, k) * B(i, j, k)$ çarpımı hesaplanır. hesaplamasını yaparlar.

5. ADIM: $\log(n)$ olarak bulunur.

Tüm işlemciler aynı anda paralel çalışarak

$$C(0, j, k) = \sum_{i=0}^{n-1} c(i, j, k) \text{ toplamı bulunur.}$$

formülünden $n*n$ adet toplam bularak sonucu elde ederler.

ANALİZ n için bu algoritmanın maliyeti:

1, 2, 3 ve 5. adımlar q kadar sabit iterasyondan oluşurlar. 4. adım ise tamamen paralel yapıldığı için iki işlemcinin taşıdıkları değerlerin çarpımı ne kadar zaman

alıyorsa; o kadarlık sabit bir zaman sonunda işlem biter. Sonuçta küp bağlantılı SIMD bilgisayara göre tasarlanan bu algoritma $O(q)$ zamanda çalışır. $n=\exp(2,q)$ kabulü yapıldı buradan çekilen q değeri yerine konulursa algoritmanın çalışma zamanı

$$t(n) = O(\log(n)) \text{ olarak bulunur.}$$

Bu küp bağlantılı SIMD bilgisayarda matris çarpımını gerçekleştiren en hızlı algoritma dır. Her $c(i,j)$ değerinin n elemanın toplamı olduğuna dikkat edilirse bu işlemin veya daha fazla işlemciden oluşan interconnection network tipi paralel bilgisayarlarda çözümünün alt sınır $OMEGA(\log(n))$ dir. bunu görmek için n adet sayının toplamının en az S tamsayısı kadar adımda gerçekleştiğini düşünelim. Son adım sırasında; son toplamı yapmak ve sonucu elde etmek için 1 işlemciye ihtiyaç vardır. (S-1). adım sırasında en fazla 2 işlemci kullanılır. (S-2). adımda en fazla 4 işlemci kullanılır... Sonuçta S adet adımdan sonra yapılan yararlı toplamaların adedi

$$\sum_{i=0}^{S-1} \exp(2,i) = \exp(2,S)-1 \text{ dir.}$$

Sonuçta n sayının toplanması için en az $n-1$ adım toplama yapmak gerekir. Bizde $n-1 = \exp(2,S)-1$ dir bu da $S \geq \log(n)$ demektir.

$p(n)=n*n*n$ olduğu için bu algoritmanın maliyeti

$$c(n) = O(n*n*n*\log(n)) \text{ olarak hesaplanır.}$$

Algoritmanın maliyetine bakıldığında pahalı gelebilir ama

hızı Mesh Matris Çarpımı Metodu'ndan daha yüksektir. maliyeti arttıran yine kullanılan fazla miktarda işlemci olmuştur.

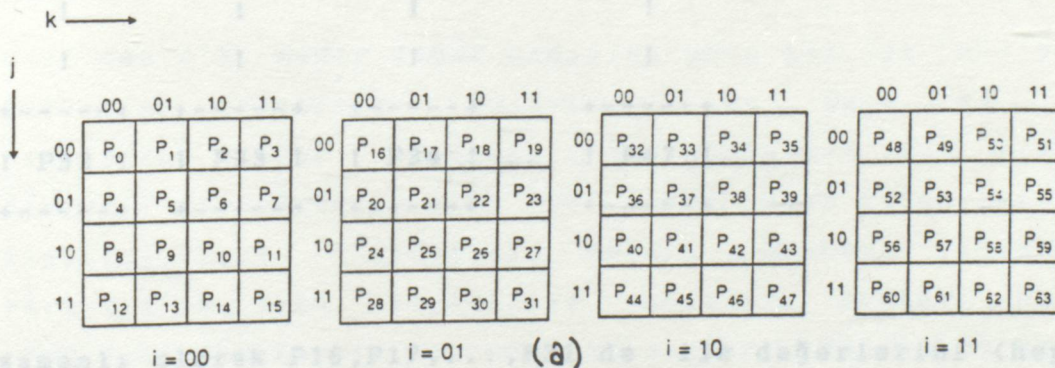
Küp Bağlantılı SIMD Matris Çarpımı Örneği

Satır ve sütun sayısı $n=2^2$ olan ($q=2$) 4×4 boyutlu A ve B matrislerinin küp bağlantılı işlemciler ile nasıl çarpıldığını adım adım inceleyelim.

1. ADIM:

$q=2$ olduğuna göre 1. adımda n değişken $3^q-1=9$ ile 2^q-4 arasındadır. $q=2$ için $3^2-1=9$ ve $2^2-4=0$ olur. Birinci adımda $q=2$ için $3^2-1=9$ ve $2^2-4=0$ olur. Birinci adımda $q=2$ için $3^2-1=9$ ve $2^2-4=0$ olur.

64 tane işlemci kullandığımızı düşünelim. $N=64$ $N=\exp(2,g)$ formülünden $g=6$ olarak bulunur. Bu işlemcilerin 6 boyutlu bir küp oluşturacak şekilde bağlandığı anlamına gelir. İşlemciler $P_0, P_1, \dots, P_{63}$ olarak adlandırılırlar. Aşağıdaki şeklin (a) şikkında ki gibi 3 boyutlu bir dizi olarak gösterebilir.



Her (i, j, k) P_r işlemcisinin indisi olan r 'nin binary ifadesine 2 bit göndererek katılır. r sayısının binary ifadesi $r_5r_4r_3r_2r_1r_0$ ise $P_r: i = r_5r_4, j = r_3r_2, k = r_1r_0$ dir. İlk durumda A ve B matrislerinin $P_0 \dots P_{15}$ kadar 16

adet işlemciye nasıl yerleştirildiği aşağıdaki şekilde gösterilmiştir. imaktadır.

17	23	27	3
-7	-25	-19	-5
9	1	14	15
-18	-30	-28	-12
31	26	22	8
-13	-21	-11	-32
15	4	10	29
-20	-2	-6	-24

1. ADIM:

(b)

$q=2$ olduğuna göre 1. adımda m değişken $3*q-1=5$ ile $2*q=4$ arasında değer alacağından iki iterasyon olacaktır. Biri $m=5$ diğeri $m=4$ için

Eşzamanlı olarak P32,P33,...,P47 de birinci iterasyonda

Birinci iterasyonda $m=5$ binary indisleri $r_5r_4r_3r_2r_1r_0$ olup ta bunun 5. biti $r_5=0$ olan işlemciler taşıdıkları değeri $r_5'r_4'r_3'r_2'r_1'r_0'$ olan yani $r_5'=1$ olan işlemcilere atarlar. Atamalar aşağıdaki şekilde gösterilmektedir.

```

+-----+ +-----+ +-----+ +-----+
! P0 ! ! P1 ! ! P2 ! ... ! P15 !
+-----+ +-----+ +-----+ +-----+
! ! ! !

```

$m=4$! ! ! ! !

+-----+ +-----+ +-----+ +-----+

! P32 ! ! P33 ! ! P34 ! ... ! P47 !

+-----+ +-----+ +-----+ +-----+

saklayıcılarında taşıdıkları değeri binary indisleri

$r_5r_4r_3r_2r_1'r_0$ olan işlemcilere atarlar. Örneğin P0,P1

Eşzamanlı olarak P16,P17,...,P31 de ilk değerlerini (hepsi 0 dır) sırasıyla P48,P49,...,P63'e atarlar.

İkinci iterasyonda binary indisleri $r_5r_4r_3r_2r_1r_0$ olan ve 4. bitleri $r_4=0$ olan işlemciler taşıdıkları değeri 4 numaralı

bitleri $r_4'=1$ olan işlemcilere atarlar. Atamalar aşağıdaki şekilde gösterilmektedir.

```

+-----+ +-----+ +-----+ +-----+
! P0 ! ! P1 ! ! P2 ! ..... ! P15 !
+-----+ +-----+ +-----+ +-----+
! ! ! !
! ! ! !
+-----+ +-----+ +-----+ +-----+
! P16 ! ! P17 ! ! P18 ! ..... ! P31 !
+-----+ +-----+ +-----+ +-----+

```

Eşzamanlı olarak $P_{32}, P_{33}, \dots, P_{47}$ de birinci iterasyonda aldıkları değerlerini sırasıyla $P_{48}, P_{49}, \dots, P_{63}$ 'e atarlar. Birinci adımın sonunda 64 işlemcinin içerdikleri değerler aşağıda gösterilmiştir.

17	23	27	3
-7	-25	-19	-5
9	1	14	16
-18	-30	-28	-12
31	26	22	8
-13	-21	-11	-32
15	4	10	29
-20	-2	-6	-24

17	23	27	3
-7	-25	-19	-5
9	1	14	15
-18	-30	-28	-12
31	26	22	8
-13	-21	-11	-32
15	4	10	29
-20	-2	-6	-24

17	23	27	3
-7	-25	-19	-5
9	1	14	15
-18	-30	-28	-12
31	26	22	8
-13	-21	-11	-32
15	4	10	29
-20	-2	-6	-24

17	23	27	3
-7	-25	-19	-5
9	1	14	-5
-18	-30	-28	-12
31	26	22	8
-13	-21	-11	-32
15	4	10	29
-20	-2	-6	-24

2. ADIM:

(c)

$m=q-1'$ den 0'a kadar değer aldığına göre $q=2$ ise $m=1$ ve 0 değerlerini alabilir. Dolayısıyla bu akım da iki iterasyondan oluşur. Birinci iterasyonda $m=1$ dir. Binary indisleri $r_5r_4r_3r_2r_1r_0$ olan ve $r_1=r_5$ olan işlemciler A saklayıcılarında taşıdıkları değeri binary indisleri $r_5r_4r_3r_2r_1'r_0$ olan işlemcilere atarlar. Örneğin P_0, P_1 işlemcileri A saklayıcılarında taşıdıkları değeri P_2 ve P_3 ün A saklayıcılarına atarlar.

İkinci iterasyonda $m=0$ dir. Binary indisleri $r_5r_4r_3r_2r_1r_0$ olan ve $r_0=r_4$ olan işlemciler A saklayıcılarında

taşıdıkları değeri binary indisleri $r5r4r3r2r1r0'$ olan işlemcilerle gönderirler. Örneğin $P0, P2$ işlemcileri A saklayıcılarında taşıdıkları değeri $P1$ ve $P3$ ün A saklayıcılarına gönderirler. 2. Adımın sonunda A matrisinin bir elemanı aşağıdaki şekilde (d) şıkında görüldüğü gibi karşı düşen satır boyunca kopyalanmış olur.

17	17	17	17
-7	-25	-19	-5
9	9	9	9
-7	-25	-19	-5
31	31	31	31
-7	-25	-19	-5
15	15	15	15
-7	-25	-19	-5

23	23	23	23
-18	-30	-28	-12
1	1	1	1
-18	-30	-28	-12
26	26	26	26
-18	-30	-28	-12
4	4	4	4
-18	-30	-28	-12

27	27	27	27
-13	-21	-11	-32
14	14	14	14
-13	-21	-11	-32
22	22	22	22
-13	-21	-11	-32
10	10	10	10
-13	-21	-11	-32

3	3	3	3
-20	-2	6	24
16	16	16	16
-20	-2	6	24
8	8	8	8
-20	-2	6	24
29	29	29	29
-20	-2	6	24

3. ADIM:

Bu adım bir önceki adıma benzer. 2. adımda A matrisi ile satırlar üzerinde yapılan işlemlerin bu defa B matrisinin elemanları ile sütunlar üzerinde yapılmasıdır. B matrisinin elemanları gerekli sütunlar boyunca kopyalanır. Durum (d) şıkında görülmektedir.

4. ADIM:

Tüm işlemciler aynı anda A ve B saklayıcılarının çarpımını hesaplarlar ve sonucu C saklayıcısına koyarlar.

5. ADIM:

$m=2*q$ ile $3*q-1$ arasında değer aldığı için iki iterasyondan oluşur. A matrisi ile B matrisini çarpma için

Birinci iterasyonda $m=4$ için gerçekleşir. N_k işlemci çiftlerinden sadece adreslerinin 4. biti farklı olanların değerleri toplanarak C saklayıcısına konur. Her iki işlemcinin C saklayıcısına bu toplam değeri aktarılır. Zaten CRCW modeli olduğundan, birkaç işlemci İkinci iterasyonda birinci iterasyonda 4. bir için yapılan işlemler bu defa $m=5$ için gerçekleşir. İşlemci çiftlerinden sadece adreslerinin 5. biti farklı olanların değerleri toplanarak C saklayıcısına konur. Her iki işlemcinin C saklayıcısına bu toplam değeri aktarılır.

Algoritmanın sonunda C matrisinin aralan 16 elemanı $P_0...P_{15}$ işlemcilerinde oluşmuş olur.

1	17	23	27	3	-944	-1666	-1282	-1297	-25	-19	-6	1
2	9	1	14	16	-583	-581	-449	-889	-30	-28	-12	1
3	31	26	22	8	-1131	-2033	-1607	-1363	-21	-11	-32	1
4	15	4	10	29	-667	-763	-681	-1139	-2	-6	-24	1

(c)

10.8. CRCW MATRİS ÇARPIMI ALGORİTMASI

Şimdi matris çarpımını küp bağlantılı SIMD bilgisayarda olduğundan daha düşük maliyetle yapan bir algoritma inceleyelim. Algoritma CRCW SIMD paralel bilgisayarlar için tasarlanmıştır. Bu tip bir bilgisayarda paylaşılan bellekteki bir bölgeye tüm işlemciler eşzamanlı olarak yazma yapabilirler. Bunlara eğer böyle bir hak veriliyorsa işlemciler arasında olacak yazma mücadelesini de çözmek gerekir. Bu algorithma bu soruna getirilen çözüm o bölgeye o anda yazma isteğinde bulunan işlemcilerin hepsinin değerlerini toplayarak yazmaktır. CRCW çarpım algoritması bölümün başında incelenen seri matris çarpımı algoritmasının doğrudan paralelleştirilmiş durumudur. $m \times n$ boyutlu A matrisi ile $n \times k$ boyutlu B matrisini çarpmak için

$m \times n \times k$ adet işlemci kullanılmaktadır. Bunların $m \times n \times k$ boyutlu bir dizi olarak bağlandıkları ve (i, j, k) şeklinde üç indis ile çağrıldıkları kabul edilmiştir. $1 \leq i \leq m$ $1 \leq j \leq n$ $1 \leq k \leq k$ dır. Başlangıçta A ve B matrisi paylaşılan bellektedir. Zaten CRCW modeli olduğundan, birkaç işlemci tarafından eşzamanlı okunması gereken bir değer olduğundan yapı bakımından bir problem yoktur. CRCW carpım programı bittiğinde C carpım matrisi de paylaşılan bellekte oluşur. Algoritmanın akış diyagramı 10.9' da verilmiştir.

CRCW MATRIS CARPIMINA BİR ÖRNEK

Carpılacak matrisin satır sayısının kübü kadar işlemci kullanıldığından,

$$\begin{array}{ccccccccc}
 ! & 17 & 23 & 27 & 3 & ! & ! & -7 & -25 & -19 & -5 & ! \\
 ! & 9 & 1 & 14 & 16 & ! & ! & -18 & -30 & -28 & -12 & ! \\
 A = ! & 31 & 26 & 22 & 8 & ! & B = ! & -13 & -21 & -11 & -32 & ! \\
 ! & 15 & 4 & 10 & 29 & ! & ! & -20 & -2 & -6 & -24 & !
 \end{array}$$

matrislerinin çarpımlarını bu metod ile bulalım. $n \times n \times n$ adet işlemci kullanıldığına göre $n=4$ ise 64 işlemci kullanılıyor demektir. Aşağıdaki şekilde de görüldüğü gibi 64 adet çarpımın tamamı eşzamanlı olarak yapılmakta ve 4 lü grupların toplamları bulunmaktadır. Bundan dolayı, örneğinc(1,1) hesaplanırken

P1		P2		P3		P4	
+-----+		+-----+		+-----+		+-----+	
! 17*-7 !		! 23*-18 !		! 27*-13 !		! 3*-20 !	
+-----+		+-----+		+-----+		+-----+	
!		!		!		!	
!		!		!		!	
-191	+	-414	+	-351	+	-60	= -944

aynı anda $c(1,1)$ gözüne yazmak isteyen 4 işlemcinin değerlerinin toplanarak yazıldığı görülür.

ANALİZ:

CRCW çarpım programının sabit zamanda çalışacağı aşıkardır. Bu bir çarpım işleminin alabileceği zaman kadardır. Dolayısıyla

$$t(n) = O(1) \text{ dir.}$$

Çarpılacak matrisin satır sayısının kübü kadar işlemci kullanıldığından

$$c(n) = n \cdot n \cdot n \cdot O(n) \text{ olarak hesaplanır.}$$

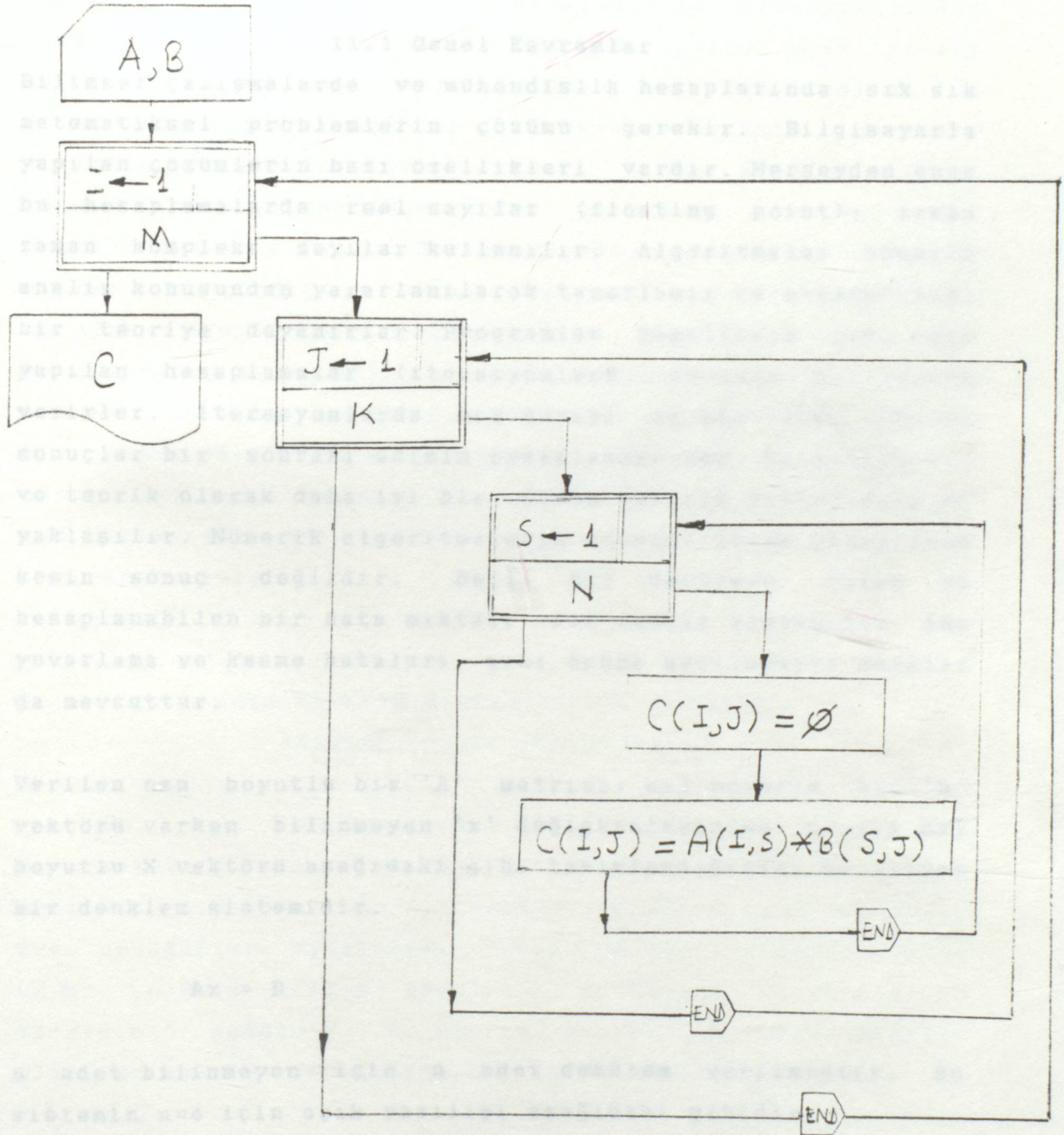
procedure CRCW MATRIX MULTIPLICATION (A, B, C)

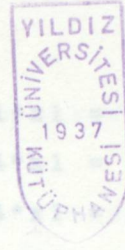
```

for  $i = 1$  to  $m$  do in parallel
  for  $j = 1$  to  $k$  do in parallel
    for  $s = 1$  to  $n$  do in parallel
      (1)  $c_{ij} \leftarrow 0$ 
      (2)  $c_{ij} \leftarrow a_{is} \times b_{sj}$ 
    end for
  end for
end for
end for. □

```


- CRCW MATRIS ÇARPIMI METODU -





LİNEER DENKLEM SİSTEMLERİNİN ÇÖZÜMÜ

Burada A matrisi verilen denklemlerin katsayılarından oluşur ve giriste 11.1 Genel Kavramlar verilen eşitliklerin Bilimsel çalışmalarda ve mühendislik hesaplarında sık sık matematiksel problemlerin çözümü gerekir. Bilgisayarla yapılan çözümlerin bazı özellikleri vardır. Herşeyden önce bu hesaplamalarda reel sayılar (floating point); zaman zaman kompleks sayılar kullanılır. Algoritmalar nümerik analiz konusundan yararlanılarak tasarlanır ve matematiksel bir teoriye dayanırlar. Programlar genellikle ard arda yapılan hesaplamalar (iterasyonlar) sonunda bir çözüm verirler. İterasyonlarda bir önceki adımda elde edilen sonuçlar bir sonraki adımın hesaplanmasında da kullanılır ve teorik olarak daha iyi bir çözüm verirle sonuca daha da yaklaşırlar. Nümerik algoritmaların sonunda çözüm genellikle kesin sonuç değildir. Belli bir dereceye varan ve hesaplanabilen bir hata miktarı ile sonuca yaklaşırlar. Ama yuvarlama ve kesme hataları gibi önüne geçilemeyen hatalar da mevcuttur.

PARALEL DİREKT ALGORİTMALAR

Verilen $n \times n$ boyutlu bir 'A' matrisi; $n \times 1$ boyutlu bir 'b' vektörü varken bilinmeyen 'x' değişkenlerinden oluşan $n \times 1$ boyutlu X vektörü aşağıdaki gibi tanımlandığında; bu lineer bir denklem sistemidir.

$Ax = B$

n adet bilinmeyen için n adet denklem verilmiştir. Bu sistemin $n=4$ için açık yazılışı aşağıdaki gibidir.

1- Herhangi bir matrisin M_i ile çarpma ve hesaplamaları gerektirir.

$$\begin{aligned}
 a_{11}x_1 + a_{21}x_1 + a_{31}x_1 + a_{41}x_1 &= b_1 \\
 a_{21}x_1 + a_{22}x_1 + a_{23}x_1 + a_{24}x_1 &= b_2 \\
 a_{31}x_1 + a_{32}x_1 + a_{33}x_1 + a_{34}x_1 &= b_3 \\
 a_{41}x_1 + a_{42}x_1 + a_{43}x_1 + a_{44}x_1 &= b_4
 \end{aligned}$$

Burada A matrisi verilen denklemlerin katsayılarından oluşur ve girişte verilir. B vektörü verilen eşitliklerin sağ tarafındaki sabitlerden oluşan bir vektördür bu da giriş değeri olarak verilir. Programın sonunda n elemanlı x vektörünün çıkışı olarak alınır. Lineer denklem sistemi çözmek için kullanılan metodlar direkt ve iteratif olmak üzere ikiye ayrılırlar. Direkt metodlar sonucu sonlu sayıda n küp mertebesinde işlem yaparal bulurlar. İteratif metodlar sonlu bir zaman dilimi içinde kesin sonucu bulamayıp sadece belli ölçüde yaklaşır. Ama gene de iteratif metodlar belli adım işlemten sonra kabuledilebilir bir sonuç verebilirler. Özellikle bilinmeyen sayısı çok büyük ise direkt metodlar tercih edilir. İteratif metodlar ise kullanılan matris sparse matris ise daha az belleğe ihtiyaç duyarlar.

Bu metodda $i = 1, 2, 3, \dots, (n-1)$ için x_i değeri olan x_i , $x_{i+1}, \dots, x_n$ cinsinden ifade edilerek, bu sırada elemanlar $x_1, x_2, \dots, x_n$ cinsinden ifade edilerek, bu sırada bilinmeyen $x_1, x_2, \dots, x_n$ cinsinden ifade edilerek, bu sırada doğru yoketme denir. $(n-1)$ adımda sonra bu işlem tamamlanır. x_i cinsinden kalan tek değişken olur.

LİNEER DENKLEM SİSTEMLERİNİN ÇÖZÜMÜNDE

PARALEL DİREKT ALGORİTMALAR

$Ax=b$ şeklinde lineer bir denklem sistemini çözmek için A matrisine diyagonalinin alt veya üst kısmı 0 oluncaya kadar bazı dönüşümler uygulanır. Sonra de geri giderek yerine koyma yoluyla sistem çözülür. Söz konusu dönüşümler A matrisinin soldan M...M matris sırası ile çarpılmasıyla olur. M_i matrisi seçilirken iki hedef düşünülmelidir.

- 1- Herhengi bir matrisin M_i ile çarpımı az hesaplama gerektirmeli,

2- $M M \dots M * A$ çarpımı üçgensel olmalıdır.

Bunu gerçekleştirmenin pekçok yolu vardır. Burada bunlardan ikisi incelenmiştir. Sistemde eşitliğin sol tarafı M matrisi ile çarpılırken eşitliğin sağ tarafı da aynı M matrisi ile çarpılmalıdır.

$$\begin{matrix} (k) & & (i) \\ (M & . & . & . & M * A) * X = \end{matrix} \begin{matrix} (k) & & (i) \\ (M & . & . & . & M * A) * b \end{matrix} \text{ dir.}$$

Sistem bu duruma getirildikten sonra geri giderek yerine koyma ile $O(n)$ zamanda çözülebilir.

GAUSS JORDAN METODU (GAUSS ELEMINASYONU)

Lineer denklem sistemini çözmek için kullanılan iyi bilinen bir yöntem Gauss-Jordan (Gauss Eleminasyonu) metodudur.

Bu metodda $i = 1, 2, 3, \dots, (n-1)$ için i . değişken olan X_i , $x(i+1), \dots, X(n)$ cinsinden ifade edilerek, bu sırada elemanter satır işlemlerinden yararlanılır, $x(i)$ bilinmeyeni $i+2, \dots, n$. denklemden elenir. Bu olaya ileri doğru yoketme denir. $(n-1)$ adım sonra bu işlem tamamlandığında $X(n)$ sistemde kalan tek değişken olarak belirlenmiş olur.

Metodun ikinci adımında $X(n-1), X(n-2), \dots, X(1)$ belirlenerek bütün bilinmeyenler bulunmuş olur.

$$\begin{matrix} (0) & & (i) & & (i) & & (i) \\ C & = & A & \text{ ve } & C & = & M \dots \dots \dots M A \end{matrix} \text{ olsun}$$

$C^{(i-1)}$ daha önce i için $1 \leq i \leq n-1$ için hesaplanmıştır.

sütunları +---+---+---+---+---+---+---+---+ şeklinde bulunur.

! 0 ! 0 ! 0 ! 0 ! 0 ! 0 !

+---+---+---+---+---+---+---+---+

(i) ! 0 ! 0 ! 0 ! 0 ! 0 ! 0 !

Cji = +---+---+---+---+---+---+---+---+

(i) ! 0 ! 0 ! 0 ! 0 ! 0 ! 0 !

N = +---+---+---+---+---+---+---+---+ i

! 0 ! 0 ! x ! 0 ! 0 ! 0 !

+---+---+---+---+---+---+---+---+

Bu ise ! 0 ! 0 ! x ! 0 ! 0 ! 0 !

+---+---+---+---+---+---+---+---+

tanımla ! 0 ! 0 ! x ! 0 ! 0 ! 0 !

Kalan elemanlar +---+---+---+---+---+---+---+---+

oldukları ispatlanmıştır.

Bu matris -i. sütunundaki satır sayısı i den büyük elemanları 0 dan farklı olan bir matristir

(i) ! 0 (i-1) x (i-1) x x x

Nji = Cji / Cii j>i ifadesi

(i) ! 0 ! 0 ! x ! x ! x ! x !

C (i) +---+---+---+---+---+---+---+---+

N nin sıfırdan farklı elemanları hesaplanarak;

(i) ! 0 ! (i) x (i-1) x x x

C = (I - N) C dir.

! 0 ! 0 ! x ! x ! x ! x !

(i) +---+---+---+---+---+---+---+---+

N matrisinin ilk (i-1) sütunu 0 olduğundan

(i) (i-1)

C matrisinde ilk (i-1). C nin

Sıfır +---+---+---+---+---+---+---+---+

ve +---+---+---+---+---+---+---+---+ (i)

karşı düşen sütunlarına eşittir. Şimdi C nin i.

sütunundaki elemanı $j > i$ ise şu şekilde bulunur.

$$C_{ji} = C_{ji(i-1)} - \frac{C_{ji(i-1)} \cdot C_{ii(i-1)}}{C_{ii(i-1)}} = 0 \text{ dır.}$$

Bu işlemlerle M değerini yukarıda verilen şekilde bulunur.

tanımladığımızda C matrisinin diyagonalinin altında kalan elemanlarından satır sayısı i den küçük elemanların 0 oldukları ispatlanmış olur. Gauss Eliminasyonu denir.

(i-1)

C_{ii} +-----+
 0! 0! x! x! x! x! x!
 (i-1)! 0! x! x! x! x! x!
 C_{ii} +-----+
 (i) ! 0! 0! x! x! x! x!
 C_{ii} +-----+
 (i) ! 0! 0! x! x! x! x!
 C_{ii} = +-----+ formundadır. 'pivoting'
 ! 0! 0! x! x! x! x!
 +-----+
 ! 0! 0! x! x! x! x!
 +-----+
 ! 0! 0! x! x! x! x!
 +-----+
 ! 0! 0! x! x! x! x!
 +-----+

(i) (i-1)

Şimdi C nin C in ilk i adet satırının eşit olduğu ve $j > i$ $k > i$ olmak üzere

$$C_{jk}^{(i)} = C_{jk}^{(i-1)} - \sum_{l=0}^n N_{jl}^{(i)} * C_{lk}^{(i-1)}$$

(i-1) (i) (i-1) hesaplanır.

$$= C_{jk}^{(i-1)} - N_{ji}^{(i)} * C_{ik}^{(i-1)}$$

i ve j. sıraların yer değiştirilmesi demek matrisin soldan P_{ij} permutasyon matrisi ile çarpılması demektir. Bu matris

$$C_{jk}^{(i)} = C_{jk}^{(i-1)} - \frac{C_{ji}^{(i-1)}}{C_{ii}^{(i-1)}} * C_{ik}^{(i-1)} \quad \text{olarak bulunur.}$$

Bu metoda Pivot Kullanmadan Gauss Eleminasyonu denir.

(i-1)

$C_{ii}^{(i-1)} = 0$ ise başarısız olur. Dolayısıyla bu metod özellikle A matrisi simetrik ise kullanılır. Çünkü bu durumda

(i-1)

$C_{ii}^{(i-1)}$ hiç bir zaman 0 olamaz.

Simetrik olmayan problemler için bir çeşit 'pivotting' işlemi kullanılır. Matrisin iki satır veya sütunu aralarında yer değiştirilir. Böylece i. diyagonalin değeri 0 olmaktan kurtarılır. En çok kullanılan 'Row pivotting' dir ve şöyledir.

(i-1) $C_{ii}^{(i-1)}$ * Hesaplanmış olan C için $j \geq i$ olacak şekilde

For (i-1) ile hesaplanmış (i-1) idi. Dikkat edilirse işleminde bir $C_{ji}^{(i-1)} = \max |C_{ji}^{(i-1)}|$ için $j > i$ değeri hesaplanır. Sonra işleminde kullanılır ve matrisin her elemanı bir işleme

$a_{ij}(i)$ olarak hesaplanırsa

C hesaplanmadan önce i. ve j. satırlar yer değiştirerek yeni-1)

$C(i-1)$ değeri hesaplanma (i) için geçen toplam zaman

y C olarak matrisi kullanılır. C hesaplanır.

i ve j. sıranın yer değiştirmesi demek matrisin soldan P_{ij} permütasyon matrisi ile çarpılması demektir. Bu matris Şekil 11.1'de verilmiştir.

ref 2 sh 121 deki matris

GAUSS-JORDAN Metodunun Paralel Gerçeklenmesi

Maliyet ise $p(n) = O(n^2 \cdot n)$ olarak bulunur. Çünkü sistemin çözümü için (i) den algoritmada $a_{ij}(i)$ işlemi

Gauss eliminasyonunda M matrisini bulurken C değeri ve değeri $a_{ij}(i)$ olarak bulunur.

Gauss Eliminasyon $M(i-1)$ den farklı matrisler üzerinde $a_{ij}(i)$ için (i-1) C_{ji} matrisi (i-1) olarak Şekil 11.3'de

$$C_{jk} = C_{jk} - \frac{C_{ji} \cdot C_{ik}}{C_{ii}}$$

TABLE 2.1. Comparison of Gauss-Jordan elimination for sparse matrices. The number of operations is the same as that obtained for the same number of operations. A communication is assumed instantaneous.

Formülüyle hesaplanmakta idi. Dikkat edilirse işlemde bir bölme, bir çıkarma ve bir çıkarma işlemi vardır. $n \cdot n$ adet işlemci kullanılır ve matrisin her elemanı bir işlemciye

	Rowwise	Columnwise	Blockwise
n	$O(n^2)$	$O(n^2)$	$O(n^2)$
n	$O(n^2)$	$O(n^2)$	$O(n^2)$

eşzamanlı olarak hesaplatılırsa **JORDAN METODU**

(n-1)

Her adımda değeri hesaplanmak için geçen toplam zaman yaklaşık olarak $3n(n-1)$ tane işlemci ve bu işlemcileri $n \times (n-1)$ lik bir matris şeklinde dağıtılarak paralel çalıştırarak $3n$ olacaktır. Bu da çalışma zamanının $t(n)=O(n)$ olması demektir. Verilerin işlemciler arasındaki hareketi Şekil 11.2' de gösterilmiştir. Şekil 11.4' de görüldüğü gibi SIMDCA'de prosedüründe verilmiştir. Programı ise EX'te SIMDCA.D.PAS olarak çıktı ile bulunmaktadır.

Prosedüre de $n \times n$ boyutlu A matrisinde her satırın $(n-1)$ elemanı olarak b vektörünün elemanları elelenmektedir. Yani $b(i) = A(i, ref 2)$ (sh 122 fig 2.2.1) gibi her satırda değerleri paylaşılan bellekten eşzamanlı okunabilmektedir.

Maliyet ise $c(n)=O(n*n*n)$ olarak bulunacaktır. Denklem sistemini çözmek için seçilen algoritmada daha az işlemci örneğin n işlemci kullanılırsa maliyet düşecek ve $c(n)=O(n*n)$ olacaktır. **JORDAN METODU (GAUSS İTERASYONU)**

Gaus Eleminasyon Metodu'nun farklı mimariler üzerinde geliştirilen algoritmalarının karmaşıklıkları Şekil 11.3' de verilmiştir.

TABLE 2.1 Bounds on the timing of Gaussian elimination for several architectures. The upper bounds are the same as those obtained for the same number of processors, if communication is assumed instantaneous.

Number of Processors	Architecture	Without Pivoting	With Pivoting
n^2	Hypercube	$O(n)$	$O(n \log n)$
n^2	Mesh	$O(n)$	$\Omega(n^{4/3})$
$n^2 / \log n$	Hypercube	$O(n \log n)$	$O(n \log n)$
n	Hypercube	$O(n^2)$	$O(n^2)$
n	Linear Array	$O(n^2)$	$O(n^2)$

11.2 PARALEL GAUSS JORDAN METODU

Burada incelenecek olan Paralel Gauss-Jordan Algoritması CREW SM SIMD tipi ($n \times n \times n$) tane işlemcili ve bu işlemcileri $n \times (n-1)$ lik bir matris şeklinde bağlı paralel bilgisayar için tasarlanmıştır. CREW tipi seçildiğinden tüm işlemciler gerektiğinde ortak bellekten eşzamanlı okuma yapabilirler. Algoritmanın akış diyagramı Şekil 11.4' de görüldüğü gibi SIMDGAJD Procedüründe verilmiştir. Programı ise Ek te SIMDGAJD.PAS olarak çıkışı ile bulunmaktadır.

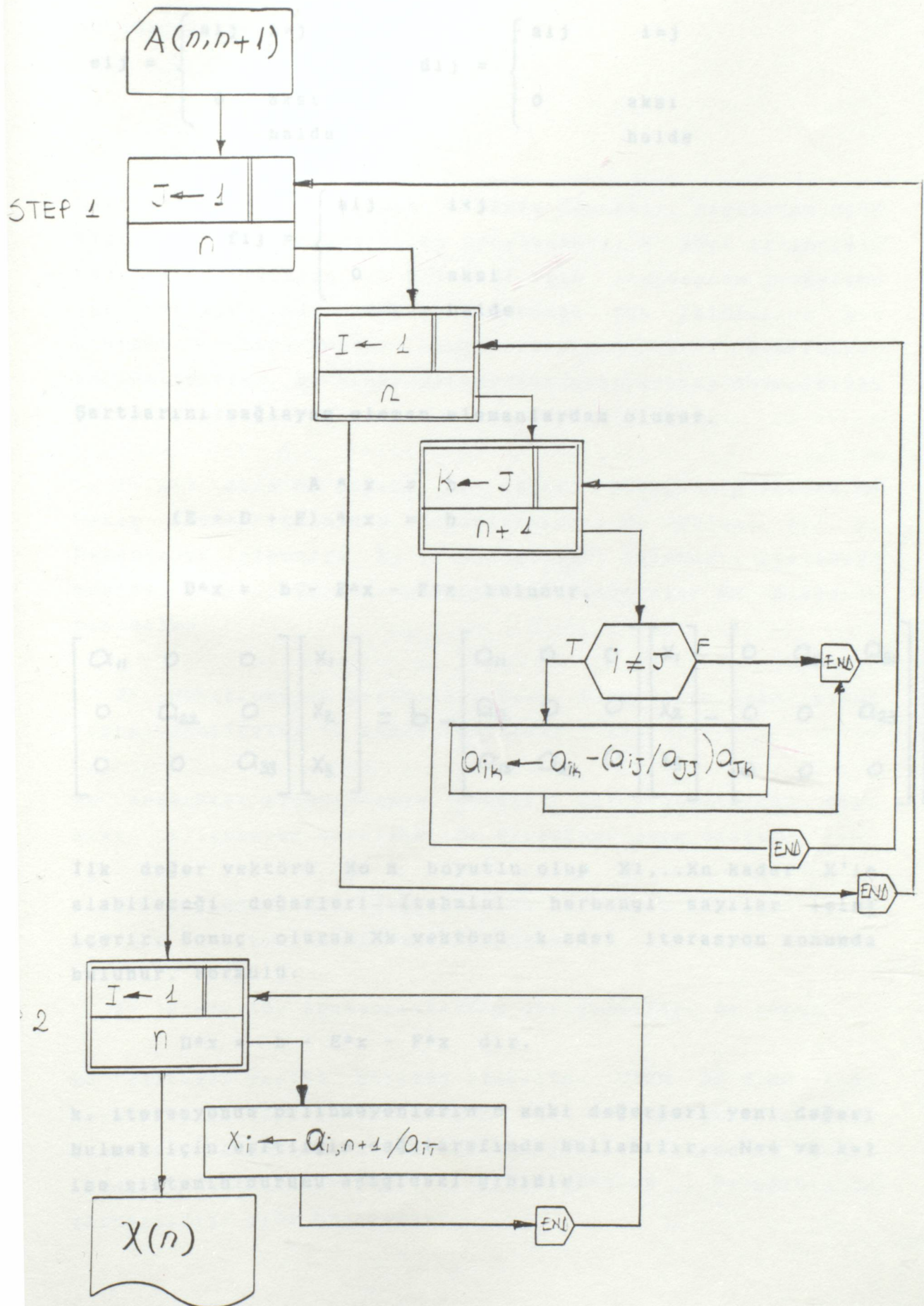
Procedüre de $n \times n$ boyutlu A matrisinde her satırın $(n+1)$. elemanı olarak b vektörünün elemanları saklanmaktadır. Yani $b(i) = A(i, n+1)$ dir. Birden fazla işlemci gerek duyduklarında değerleri paylaşılan bellekten eşzamanlı okuyabilirler.

11.3 PARALEL SEIDEL METODU (GAUSS İTERASYONU)

Bu bölümde $Ax=b$ denklem sisteminin Gauss İterasyonu ile çözümü incelenecektir.

A matrisinin $A=E+D+F$ toplamı şeklinde yazıldığını düşünelim. D, E, ve F $n \times n$ boyutlu matrisler olup elemanları d_{ij} , e_{ij} ve f_{ij} olsun.

- SIMD GAUSS-JORDAN METODU -



$$e_{ij} = \begin{cases} a_{ij} & i > j \\ 0 & \text{aksi} \\ & \text{halde} \end{cases} \quad d_{ij} = \begin{cases} a_{ij} & i = j \\ 0 & \text{aksi} \\ & \text{halde} \end{cases}$$

Bu durumda $f_{ij} = \begin{cases} a_{ij} & i < j \\ 0 & \text{aksi} \\ & \text{halde} \end{cases}$ denektir. Algoritma SIMD tipi bir f_{ij} matrisine kolay uyarlansın. N adet işlemciye vares e vektörünün $0/N$ bir aksisi için iterasyonu hesaplamak işi verilir. Her iterasyonda tüm işlemciler bir sonraki iterasyona başlanmadan önce senkronize edilmelidirler. Bu senkronizasyonun maliyeti şu nedenlerden Şartlarını sağlayan eleman elemanlardan oluşur.

1- $A * x = b$ değeri hesaplanıp kullanıma hazır $(E + D + F) * x = b$ denektir. Bu durumda X_1 'yi hesaplayan işlemciyi X_2 'yi hesaplayan işlemciye bekletmeye zorlar $D * x = b - E * x - F * x$ bulunur. İşlemci X_n hesabını bekler.

$$\begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = b - \begin{bmatrix} a_{11} & 0 & 0 \\ a_{12} & 0 & 0 \\ a_{13} & a_{32} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} - \begin{bmatrix} 0 & a_{21} & a_{31} \\ 0 & 0 & a_{23} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

İlk değer vektörü X_0 n boyutlu olup $X_1, \dots, X_n$ kadar X 'in alabileceği değerleri (tahmini herhangi sayılar için) içerir. Sonuç olarak X_k vektörü k adet iterasyon sonunda bulunur. Formülü:

2- İşlemciler senkronizasyona verilmektedir. Bu çabur

$$D * x = b - E * x - F * x \text{ dir.}$$

Bu şartları yerine getiren algoritma CREW SN SIMD tipi k. iterasyonda bilinmeyenlerin o anki değerleri yeni değeri bulmak için eşitliğin sağ tarafında kullanılır. $N=4$ ve $k=1$ ise sistemin durumu aşağıdaki gibidir.

Bu durumda metod;

$$\sum_{i=1}^n |x_i^{k+1} - x_i^k| < C$$

hata toleransı ile sonuca ulaşmış demektir. Algoritma SIMD tipi bir bilgisayara kolay uyarlanamaz. N adet işlemcimiz varsa a vektörünün n/N bileşeni için iterasyonu hesaplama işi verilir. Her iterasyon sonunda tüm işlemciler bir sonraki iterasyona başlamadan önce senkronize edilmelidirler. Bu senkronizasyonun maliyeti şu nedenlerden dolayı yüksek olabilir.

1- i < j için Xi değeri, Xj değeri hesaplanıp kullanıma hazır duruma gelmeden hesaplanamaz. Bu durum, Xi' yi hesaplayan işlemciyi Xj'yi hesaplayan işlemciyi beklemeye zorlar (j < i). Ve en sonunda tüm işlemciler Xn hesabını beklerler.

2- Hesaplamalar sırasında bazı bileşenler daha çabuk hesaplanabilirler (0 değer içerenler v.b).

Bu zorluklar algoritmanın ardışıl bir algoritmadan daha hızlı çalışmasını engeller. Bu darboğaza çare olarak:

1- En yeni hazır değerler Xi hesabında kullanılırlar böylece j < i için Xj yi beklemek gerekmez.

2- İşlemciler senkronizasyona zorlanmazlar. ha çabuk

Bu şartları yerine getiren algoritma CREW SM MIMD tipi paralel bilgisayarda çalışır. N adet işlemci kullanılır ve N <= n dir. Algoritma n adet işlem yaratır ve bunlar N adet işlemci tarafından aşağıda MIMDMODGS Procedüründe gösterildiği gibi hesaplanır.

procedure MIMD MODIFIED GS (A, x, b, c)

Step 1: for $i = 1$ to n do

(1.1) $old_i \leftarrow x_i^0$

(1.2) $new_i \leftarrow x_i^0$

(1.3) create process i

end for.

Step 2: Process i

(2.1) repeat

(i) $old_i \leftarrow new_i$

(ii) $new_i \leftarrow \left(b_i - \sum_{k=1}^{i-1} (a_{ik} \times old_k) - \sum_{k=i+1}^n (a_{ik} \times old_k) \right) / a_{ii}$

until $\sum_{i=1}^n \text{abs}(new_i - old_i) < c$

(2.2) $x_i \leftarrow new_i$. $\square$

X_i , ilk değer olarak verilmiştir. old_i değişkeni X_i 'nin bir önceki değerini saklamak için kullanılır. new_i değişkeninde X_i 'nin yeni hesaplanan değeri bulunur. 'c' ise köklerin bulunmak istendiği hassasiyet derecesidir. Algoritmada new_i değerine birden fazla işlemci ihtiyaç duyabileceği için, bu değişken paylaşılan bellekte saklanmaktadır.

Yukarıda verilen procedürede dikkat edilmesi gereken bir nokta vardır. Bir işlemci bir değeri güncelleştirirken başka bir işlemcinin bu değeri okumasını engellemek gerekir. Bu durumu engellemek için bir semaphore (bayrak) kullanmak gerklidir. Daha önce de belirtildiği gibi MIMD algoritmalarının asenkron tabiatından dolayı teorik olarak analiz edilmeleri zordur. Semaphoreların değerlerine göre hangi değişkenin ne zaman hangi işlemci tarafından hesaplandığını ortaya çıkarmak gerekir. Programın kesin çalışması ancak deneysel bir çalışma sonunda görülebilir.



XII. PARALEL BİLGİSAYARLARDA PERFORMANS

Çoklu işlemcili bir sistemde asenkron iteratif bir algoritmanın efektifliği algoritmanın performans imkanlarına bağlıdır. Gevşek bağlı bir sistemde işlemciler arası iletişim performansı etkileyen baskın bir faktördür. Bu bölümde asenkron iteratif bir algorithmada senkronizasyonun ve işlemciler arası iletişim performansı nasıl etkelediği anlatılacaktır. Tanıtılacak modelde iteratif algoritma Vektörel Ayırıştırma metodu ile tamamen eşit işlere (Tasks) ayrılmıştır. Her iterasyon içinde tamamen ardışıl bir faz olabilir. Algoritma üzerinde çalışacağı mimariye bağımlı olarak bir takım performans özellikleri ile karakterize edilir. Bunlar $(f_1, f_2, \dots, f_n)$ dir. Bu durumda 'F' (Feathure Space) olarak tanımlanır ve

$F = f_1 * f_2 * \dots * f_n$ şeklinde gösterilebilir. Verilen mimari için Performans İndeksi gerçel bir fonksiyondur. Bu indeks içindeki bölgesel minimum noktası F üzerinde mimari ve algoritmanın özellikle iyi uyduğu noktayı gösterir. Simetrik bir çoklu işlemcili sistem birbirinin tamamen aynı olan birçok işlemciden oluşur. Bir hesaplama sırasında bunların herbirinin üzerinde değişken sayıda komut işlenir ve bu komutların her biri de değişken sayıda makina çevriminden oluşur. Bir bellek erişimi sırasında bellek erişimi zamanında belirsizlikler (Memory access time fluctuations), bir komutun yürütme çevrimi sırasında ise yürütme zamanında belirsizlikler (Execution time fluctuations) oluşur.

Paralel algoritma tasarısında bir başka ilginç problem ise Aşağıdaki şekilde gevşek bağlanmış bir sistemde efektiflik faktörü görülmektedir. Şekilden de açıkça görüldüğü gibi efektifliği en çok etkileyen iletişimin hesaplama zamanına oranı (communication to computation time ratio) (t'_{cb}) dir.

Bu oran gevşek bağlı sistemlerde senkronize iteratif algoritmanın uygulanabilirliğini sınırlar. Bu oranın düşük olduğu algoritmalar gevşek bağlı sistemlerde rahatça kullanılabilir. Bu oran çok az bir yüzde mertebesinde artsa

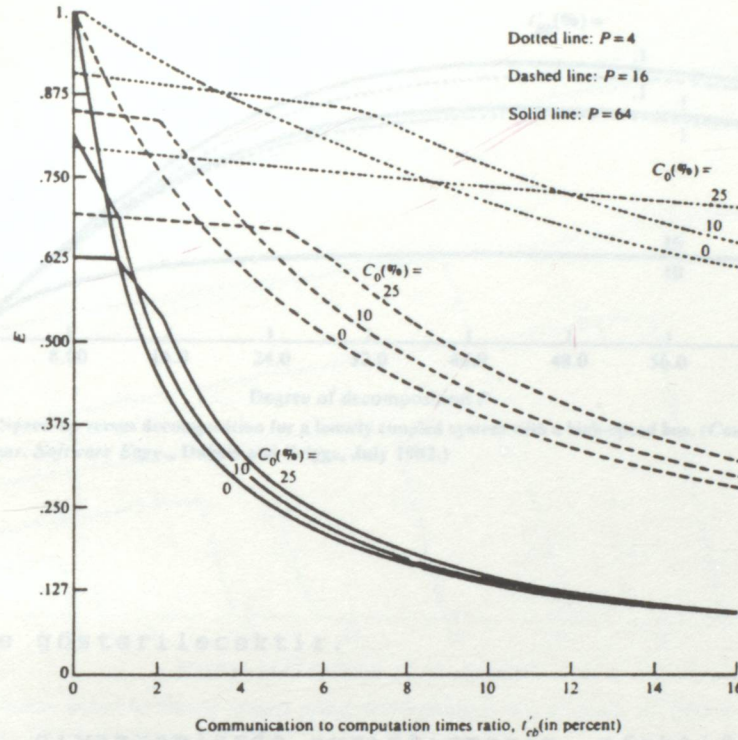


Figure 8.33 Efficiency versus feature t_{cb} for a loosely coupled system with a high speed. (Courtesy IEEE Trans. Software Engg., Dubois and Briggs, July 1982.)

bile algoritmanın efektifliği birden ve çok fazla düşer. Aşağıda performans ile ilgili bir başka diyagram görülmektedir. Dikkat edilirse sistemdeki işlemci sayısı arttıkça efektiflik faktörü küçülmektedir. Bu diyagram sayesinde mimari ve algoritma arasındaki etkileşim daha da bellidir.

Ayrıştırmanın Performans Üzerindeki Etkileri

Paralel algoritma tasarımı bir başka ilginç problem ise ayrıştırmanın algoritmanın efektifliğini nasıl etkilediğini bulmak ve algoritmanın maksimum hız ile çalışmasını sağlamak için optimum ayrıştırmaya karar vermektir. Ayrıştırma derecesi 'P' ile maksimum ayrıştırma derecesi

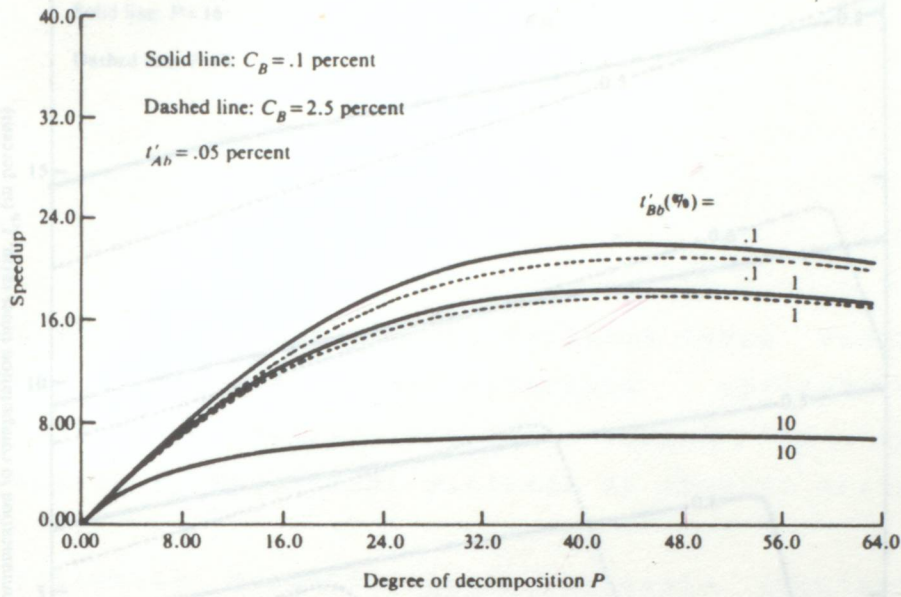
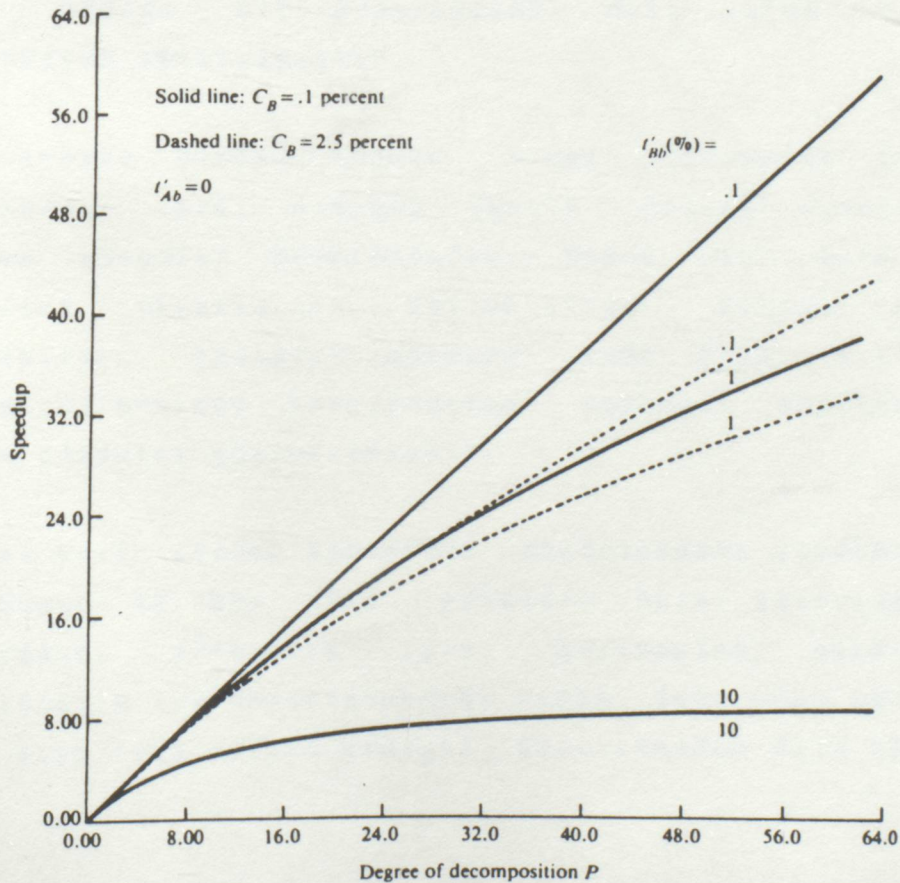


Figure 8.36 Speed-up versus decomposition for a loosely coupled system with a high-speed bus. (Courtesy of IEEE Trans. Software Engg., Dubois and Briggs, July 1982.)

'PMAX' ile gösterilecektir.

Aşağıdaki diyagramlarda ayrıştırmanın efektiflik üzerine etkisi gösterilmektedir.



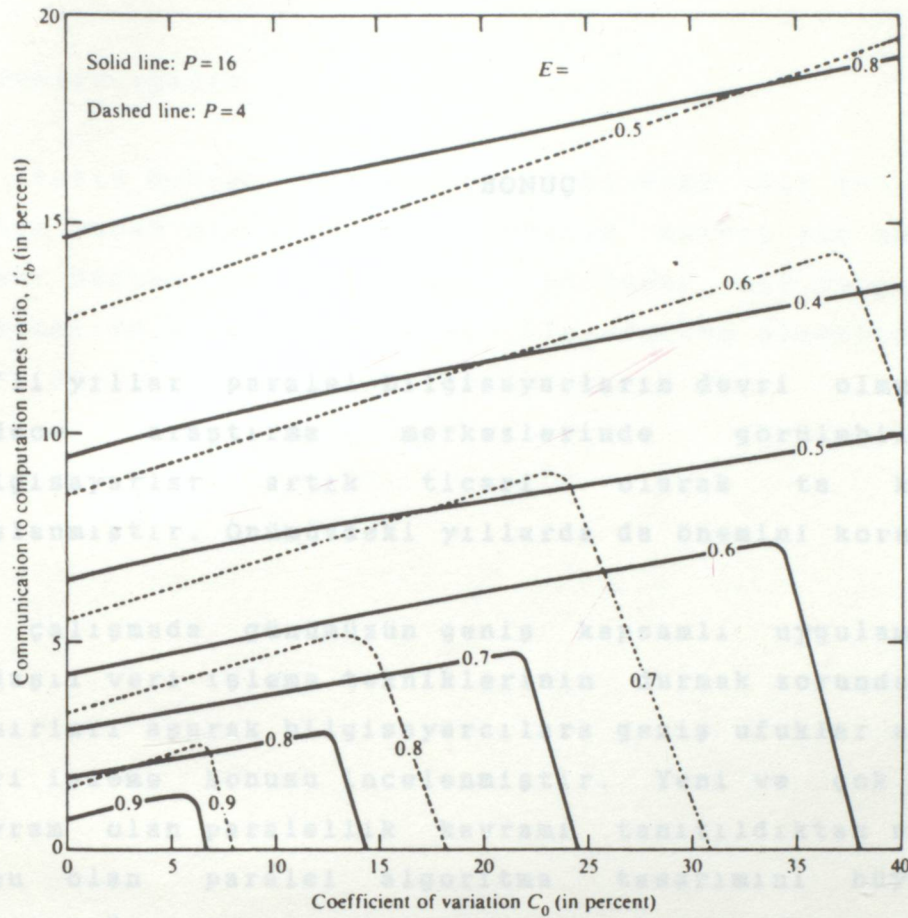


Figure 8.34 Feature planes for loosely coupled system with high-speed bus. (Courtesy of *IEEE Trans. Software Engg.*, Dubois and Briggs, July 1982.)

verebilmektedir.

Bu tesin davası olarak SONUÇ bilcek bir çalışmada seri işlen yapan bir bilgisayar donanım ekibi ile paralel işlen yapar duruma getirilebilir. seçilecek bir paralel işletim sistemi ve programlama dili ile seçilen algoritmalar gerçek

80'li yıllar paralel bilgisayarların devri olmuş, önceleri sadece araştırma merkezlerinde görülebilen süper bilgisayarlar artık ticari olarak ta kullanılmaya başlanmıştır. Önümüzdeki yıllarda da önemini koruyacaktır.

Bu çalışmada günümüzün geniş kapsamlı uygulamaların da, ardaşıl veri işleme tekniklerinin durmak zorunda kaldıkları sınırları aşarak bilgisayarlılara geniş ufuklar açan paralel veri işleme konusu incelenmiştir. Yeni ve çok önemli bir kavram olan paralellik kavramı tanıtıldıktan sonra, esas konu olan paralel algoritma tasarımı büyük ölçüde etkilediği için, paralel bilgisayarların donanım ve yazılım (programlama dilleri, derleyiciler, işletim sistemleri) özellikleri, geniş bir perspektifte değerlendirilmiştir. Başlı başına bir programlama dalı olan Concurrent Programlama anlatılmıştır.

Bilgisayarda çözümü mümkün temel problemler paralellik prensibinin ışığı altında teorik olarak incelenmiş ve optimum çözümler önerilmiştir. Bunun yanı sıra paralel algoritma tasarımının zorluklarına dikkat çekilmiş, problemlerin ayrıştırılmasında veya buna zıt olarak birleştirilmesinde karşılaşılan sorunlar açıklanmış ve mümkün çözümler gösterilmiştir.

Paralel veri işleme konusunda unutulmaması gereken önemli bir husus ta her işin paralell hale getirilemeye ve performansı arttırmak için kullanılan paralel bir algoritmanın iyi tasarlanmadığı zaman daha önce performansı düşük diye terk edilen ardaşıl algoritmadan bile kötü sonuç

verebileceğidir.

REFERANSLAR

Bu tezin devamı olarak yapılabilecek bir çalışmada seri işlem yapan bir bilgisayar donanım ekleri ile paralel işlem yapar duruma getirilebilir, seçilecek bir paralel işletim sistemi ve programlama dilisi ile seçilen algoritmalar gerçek uygulamalar için kullanılabilirler.

2. Quinn, M. J., 'Designing Efficient Algorithms For Parallel Computers', Mc Graw Hill, 1987.

3. Akl, Selim, 'Parallel Sorting Algorithms' Academic Press, Inc., 1985.

4. Erensoy, E., 'Computational Complexity of Sequential and Parallel Algorithms' John Wiley & Sons, 1985.

5. Jones, G., 'Programming in UCBAR', Prentice Hall, 1987.

6. Rockney, E. W., Jesshope, C. R., 'Parallel Computers Architecture, Programming and Algorithms' Adam Hilger Ltd., 1981.

7. 'Parallel Algorithms and Architectures' Proceedings of the International Workshop on Parallel Algorithms and Architectures, Centre National de Recherches Mathématiques, Luminy France, 14-18 April 1986 Elsevier Science Publishers B.V., 1986.

8. Stone, H. S., 'High Performance Computer Architecture', Addison Wesley, 1983.

9. Cohen, N., McGettrick, A. D., 'Concurrent Programming', International Computer Science Series, 1986.

10. Hopewich, R. J., 'Algorithms', Addison Wesley, 1986.

11. Hwang, S. 'Computer Architecture and Parallel
Analysis' Addison-Wesley, 1985

12. Hwang, S., Briggs, F. A. 'Computer Architecture and Parallel
Computation Numerical Methods', Prentice Hall, 1989

13. Ari, M. B. 'Principles of Concurrent Programming' Prentice
Hall, 1987

14. BYTE -

3. Akl, Selim, 'Parallel Sorting Algorithms' Academic
Press, Inc., 1985

4. Kronsjö, L. 'Computational Complexity of Sequential and
Parallel Algorithms' John Wiley & Sons, 1985

5. Jones, G., 'Programming in OCCAM', Prentice Hall, 1987

6. Hockney, R. W., Jesshope, C. R. 'Parallel Computers
Architecture, Programming and Algorithms' Adam Hilger Ltd.,
1981

7. 'Parallel Algorithms and Architectures' Proceedings of
the International Workshop on Parallel Algorithms and
Architectures' Centre National de Recherches Mathématiques,
Luminy France, 14-18 April 1986 Elsevier Science Publishers
B. V., 1986

8. Stone, H. S., 'High Performance Computer Architecture',
Addison Wesley, 1983

9. Gehani, N., McGettrick, A. D., 'Concurrent Programming',
International Computer Science Series, 1988

10. Sedgewick, R., 'Algorithms' Addison Wesley, 1988

11. Baase, S. 'Computer Algorithms' Introduction to Design and Analysis' Addison Wesley, 1988

12. Hwang, K. Briggs, F.A. 'Computer Architecture and Parallel Processing' Mc. Graw Hill, 1986

13. Ari M.B. 'Principles of Concurrent Programming' Prentice Hall, 1982

14. BYTE -

- November 1988

- December 1989

15. IEEE Transactions On Computers

- Vol. C-31 no:4 April 1982 'Experience With Multiprocessor Algorithms'

- Vol. C-26 no:2 February 1977 'Parallel Processors and Processing An Introduction'

- Vol. c-22 no:8 August 1973 'A Parallel Algorithm For The Efficient Solution Of General Class Of Recurrence Equations'

- Vol. C-28 no:5 May 1979 'Very Fast Fourier Transform Algorithms Hardware For Implementation'

- Vol. c-29 no:7 July 1980 'Fast Sorting Algorithms On Uniform Loaders'

- Vol. c-29 no:7 July 1980 'A Computation Model Of Parallel Solution Of Linear Equations'

- Vol. c-34 no:11 November 1985 'Parallel Computer Architectures and Problem Solving Strategies For The Consistent Labeling Problem'

ÖZGEÇMİŞ

FÜSUN YAMAN 1967' de İstanbul'da doğdu. Liseyi Özel Kadıköy Kız Lisesi'nde bitirdi.1987 yılında Yıldız Üniversitesi Bilgisayar Bilimleri Mühendisliği Bölümünden mezun oldu. Kütüphane Otomasyonu ve veri tabanı yönetim sistemleri konusunda uzmandır. Kültür Bakanlığı Süleymaniye Kütüphanesi'nin bilgisayar sisteminin kurulması projesini gerçekleştirmiştir. Ocak 1987'den bu yana Y.Ü Mühendislik Fakültesi, Bilgisayar Bilimleri Mühendisliği Bölümde Araştırma Görevlisi olarak çalışmaktadır.



