Paralel Programlamanın Gerçeklenebileceği Düzeyler

1. Program düzeyinde
2. Alt program düzeyinde (proses, thread)
3. Deyim düzeyinde (parallel for, while, ...)
4. İşlem düzeyinde ($y = a^i + b^j + c^k$)
5. Mikro kod düzeyinde (pipeline -> fetch, decode, execute)

Paralel Bilgisayarların Sınıflandırılması

1. İşlemci türü / sayısı
    a. Massively parallel (binlerce işlemci : CM-5)
    b. İri tanecikli (Crouse grained: işlemci sayısı az ama güçlü)
2. İşlemci ara bağlaşımı
    a. Paylaşılan belleğe sahip
    b. Mesaj aktarımlı yapılar
3. Global kontrol (komut ve veri elemanları göz önüne alınarak sınıflandırılır)
    a. SISD (single instruction single data): evimizdeki tek işlemcili PC ler
    b. SIMD: tüm işlemciler aynı programın aynı kod satırını farklı data ile yürütürler (parallel for)
    c. MIMD: asenkron çalışan işlemciler için
    d. MISD: uygulanması söz konusu değil
4. Eşzamanlı / asenkron çalışma

Farklı Mimariler

1. Dizi ($\ldots - P_{i-1} - P_i - P_{i+1} - \cdots$)
2. İki boyutlu dizi (Dört bir tarafla ara bağlantılı)
3. Ağaç yapısı şeklinde (d düzeyli bir ağaçta $2^d - 1$ düğüm bulunur)
4. Perfect shuffle ($d = 2i,\ 0 \le i \le n/2$ ve $d = 2i + 1 - n,\ n/2 \le i < n$ şeklinde)
5. Küp bağlantılı mimari (bağlantılar küpün ayrıtları şeklinde)

ÖR: n elemanlı u_i dizisinin elemanlar toplamını hesaplamak için, ağaç yapısı kullanılırsa, $n/2$ yapraklı ağaç yeterli

Biz genel olarak PRAM (Parallel Random Access machine) modeline uygun olarak çalışacağız

Özellikleri:
- Çok sayıda işlemci + bir ortak bellek
- Ortak bir saat üzerinden eşzamanlı çalışma
- Ortak bellek üzerinden haberleşme

Ortak Belleğe Erişim

1. Exclusive (dışlamalı): bir anda yalnızca bir işlemci erişebilir
2. Concurrent (çoklu): çok sayıda işlemci aynı anda erişebilir

Okuma ve yazma türlerine göre sınıflandırırsak: EREW (tekli okuma, tekli yazma), CREW, ERCW, CRCW.

Çoklu yazma çelişkilerini giderme yöntemleri

1. ECR (equality conflict resolution): aynı değer yazılacaksa izin ver
2. PCR (priority conflict resolution): işlemciler arası indislerine göre öncelik
3. ACR (arbitrary conflict resolution): rasgele birini seç, yazsın

Paralel Algoritma Analizi

1. Yürütme Zamanı: Çalışma-sonlanma arası geçen süre
    a. İşlem süresi
        i. Hesaplama adımları
        ii. İletişim (yönlendirme) adımları
    b. Alt ve üst sınır
    c. Hızlanma: en hızlı ardışıl çalışma süresi / paralel çalışma süresi
2. Gerek Duyulan İşlemci Sayısı ($P_{iş}$)
3. Maliyet: $C_{par} = O_{par}(N).P_{iş}$
4. Verimlilik (efficiency): $E_p = T_{seri}(n)/C_{par}$

SEÇME (Selection): n elemanlı sırasız bir dizi, k. en küçük elemanı bul

```
procedure SEQUENTIAL_SELECT(S, k)
        // Küçük bir Q değeri belirle – örneğin 5
1       if |S| < Q,
                then S'i sırala ve k. elemanı dön
        else
                S'i her birinde Q adet elemanı olan |S|/Q adet alt diziye böl
        end if
2       Her alt dizinin medyanını bul
3       Medyanların medyanı olan M değerini bul
4       S'i, M'den küçük, M'e eşit ve M'den büyük olmak üzere 3 alt diziye böl → S1, S2, S3
5       if |S1| < k,
                then return SEQUENTIAL_SELECT(S1, k)
        else if |S1| + |S2| < k,
                then return M
        else
                then return SEQUENTIAL_SELECT(S3, k - |S1| - |S2|)
        end if
end
```

Yukarıda anlatılan algoritma kullanılarak, dizinin tamamının sıralanmasıyla elde edilebilecek $O(n \log n)$'lik çalışma süresi $O(n \log \frac{n}{Q})$'ya indirilebilir.

Aşağıda seçme işlemini SIMD-EREW ortamında paralel olarak gerçekleyelim. İşlemci sayısı $P_{iş} = N = n^x,\ 0 < x < 1$
Amacımız: Yürütme zamanını küçültüp, ardışıl çözümden daha iyi bir çözüm elde etmek (maliyet optimal olmalı)

Öncelikle, bir D verisinin, A ortak bellek alanı kullanılarak N adet işlemciye nasıl yayınlanabileceğini inceleyelim

```
procedure BROADCAST(D, N, A)
1       İşlemci P1:
        i. D adresinde yer alan veriyi okur
        ii. Bir kopyasını kendi yerel belleğine alır,
        iii. Bu değeri A[1] gözüne yazar
2       for i=0 to log(N-1) do
                for j=2^i+1 to 2^(i+1) do in parallel
                        İşlemci Pj:
                        i. A[j - 2^i] alanındaki veriyi okur
                        ii. Bu veriyi kendi yerel belleğine yazar
                        iii. Bu bilgiyi A[j] alanına yazar
                end for
        end for
end
```

Yukarıda açıklanan veri yayma yöntemi her adımda 2 kat daha fazla işlemciye veriyi iletmektedir, $O(\log N)$. Yine bir dizideki sayıların, $1 \le k \le n$ olan tüm k değerleri için, ilk k toplamın tamamını bulan prosedür aşağıdaki gibi yazılabilir.

```
procedure ALLSUMS(ai)
        for j=0 to log(N-1) do
                for i=2^j+1 to N do in parallel
                        İşlemci Pi:
                        i. Ortak bellek üzerinden P(i-2^j) işlemcisinden a(i-2^j) verisini al
                        ii. ai = ai + a(i-2^j)
                end for
        end for
end
```

Yukarıda verilen yöntem $O(\log N)$ karmaşıklıkta, 1'den olası tüm k'lara kadar olan toplamları bulabilmektedir.

Paralel seçme algoritmasını gerçeklerken, EREW-SIMD bir yapıda çalışacağız. Tüm varsayımlarımız aşağıdadır:

- Sıralanmamış olan n elemanlı $S = \{s_1, s_2, \dots, s_n\}$ dizisinin; $1 \le k \le n$ olmak üzere k. elemanını bulacağız.
- N adet işlemci mevcut ($N = n^{1-x},\ 0 < x < 1$)
- Her işlemci n bilgisine sahip (BROADCAST ile bu bilgiyi edinmiş)
- Her işlemcinin belleğinde $n/N = n^x$ elemanlı bir diziyi taşıyacak yerel bellek mevcut
- Her işlemci SEQUENTIAL SELECT , BROADCAST ve ALLSUMS prosedürlerini yürütebiliyor
- Paylaşılan bellekte N elemanlı bir M dizisi için alan var

```
procedure PARALLEL_SELECT(S, k)
1       if |S| ≤ 4
              then P1 işlemcisi k. elemanı belirler ve geri döner
        else
              i. S, her biri |S|^x elemanlı |S|^(1-x) adet Si alt dizisine bölünür
              ii. Her Si alt dizisi bir Pi işlemcisine atanır
        end if
2       for i=1 to |S|^(1-x) do in parallel
2.1           mi <-- SEQUENTIAL_SELECT(Si, ceil(|Si| / 2))  // kendi dizisinin medyanı
2.2           Pi işlemcisi, mi değerini ortak bellekteki M[i] gözüne yazar
        end for
3       m <-- PARALLEL_SELECT(M, ceil(|M|/2))   // medyan dizisinin medyanı
4       S dizisi 3 alt diziye bölünür:
              L = {si ε S: si < m}
              E = {si ε S: si = m}
              G = {si ε S: si > m}
5       if |L| ≤ k then PARALLEL_SELECT(L, k)
        else if |L| + |E| ≤ k then return m
        else PARALLEL_SELECT(G, k - |L| - |E|))
        end if
end
```

Gerçekleme Ayrıntıları:

1. S dizisinin ortak bellekteki adresi A, $|S|$ boyut bilgisi ve k değerleri BROADCAST ile dağıtılır
2. Her işlemci x'i hesaplar
3. Her işlemci kendi alt dizisinin ilk-son değerlerini hesaplar: $P_i \rightarrow A + (i-1)n^x : A + in^x - 1$
4. Her P_i işlemcisi, m değerini BROADCAST ile öğrenmiştir
   Tüm aralıklar, örneğin L için, $L = L_1 + L_2 + \cdots + L_N$ şeklinde belirlenir (önce kendi L_i'leri hesaplanır)

ÖR: $n = 29$ için 21. elemanı bul, $S_i = [6,6,6,6,5]$ şeklinde paylaştırılır, medyanlar $m_i = [14,9,7,25,32]$ ve $m = 14$, $L_i = [2,4,5,0,0]$, ALLSUMS ile $z_i = \sum l_i = [0,2,6,11,11]$ hesaplanır. P_i işlemcisi L dizisinin $z_{i-1} + 1$ gözüne yazar.

KAYNAŞTIRMA

Sıralı ve r elemanlı $A = \{a_i\}$ dizisiyle, sıralı ve s elemanlı $B = \{b_j\}$ dizisi kaynaştırılarak, sıralı ve $r + s$ elemanlı $C = \{c_k\}$ dizisi elde edilecektir. Ardışıl çözüm aşağıda verilmiştir.

```
procedure SEQUENTIAL_MERGE(A, B, C)
1.1     i <-- 1
1.2     j <-- 1
2       for j=0 to r+s do
              if ai < bj then
                    ck = ai
                    i = i + 1
              else
                    ck = bj
                    j = j + 1
              end if
        end for
end
```

Paralel kaynaştırma algoritmasını vermeden önce, sıralı dizide kullanılabilecek ikili arama yöntemini verelim.

```
procedure BINARY_SEARCH(S, x, h) // x elemanı var mı? Varsa yeri, yoksa 0 dön
1.1     i = 1
1.2     h = n
1.3     k = 0
2       while i≤h do
2.1           m = floor((i+h)/2)
2.2           if x = m then
                    k = m
                    i = h + 1 // while'dan çık
              else if x < Sm then
                    h = m - 1
              else
                    i = m + 1
              end if
        end while
end
```

```
procedure CREW_MERGE(A, B, C) // paralel okumalara izin var
1       for i=1 to N-1 do in parallel
                // Pi işlemcisi ai' ve bi', kendi bölgelerinin başlangıç noktalarını belirler
                ai' = ai . ceil(r / N) // A'
                bi' = bi . ceil(s / N) // B'
        end for
2       V = {v1, v2, ..., v2N-2} <-- A' ve B' dizisini kaynaştır
        vi = (ai / bi) --> A' / B' dizi indisi
2.1     for i=1 to N-1 do in parallel
                Pi işlemcisi B' üzerine BINARY_SEARCH uygulayararak ai' < bj' olacak en küçük j
                if j bulundu then                                          değerini bulacak
                        Vi+j-1 = (ai', i, A)
                else
                        Vi+N-1 = (ai', i, A)
                end if
        end for
2.3     for i=1 to N-1 do in parallel
                Pi işlemcisi A' üzerine BINARY_SEARCH uygulayararak bi' < aj' olacak en küçük j
                if j bulundu then                                          değerini bulacak
                        Vi+j-1 = (bi', i, B)
                else
                        Vi+N-1 = (bi', i, B)
                end if
        end for
3       // A ve B kaynaştırılıp, C oluşturulacak. Önce hangi indislerden başlanacağını gösteren Q
3.1     Q(1) = (1.1)                                                       dizisini oluştur
3.2     for i=2 to N do in parallel
                if V2i-2 = (ak', k, A) then Pi işlemcisi
                        B'e BINARY_SEARCH uygulayarak bj > ak' olacak en küçük j'yi bulur
                        Q(i) = (k.ceil(r/N), j)
                else
                        A'ya BINARY_SEARCH uygulayarak aj > bk' olacak en küçük j'yi bulur
                        Q(i) = (j, k.ceil(s/N))
                end if
        end for
3.3     for i=1 to N do in parallel
                // Pi işlemcisi SEQUENTIAL_MERGE ile Q(i) = (x,y) bilgisini kullanarak, ax ve by
                // noktalarından başlayan alt dizileri kaynaştırır ve sonucu C dizisinin x+y-1
                // adresinden başlayarak yazar. Sonlanma koşulları:
                i. i ≤ N-1 -> PN dışındaki tüm işlemciler için V2i elemanından daha büyük veya eşit
                   olan bir elemana A veya B içinde rastlanır.
                ii. Son işlemci için: i = N durumunda, A veya B'de hiç eleman kalmaz
        end for
end
```

CREW Gerçekleme Ayrıntıları (oldukça kuvvetli bir model)

Paralel okumayı nasıl gerçekleyebiliriz:

Çözüm 1: Aynı göze erişmek isteyen işlemcilerin isteklerini bir kuyruğa yerleştir ve sırayla isteği karşıla. En kötü durumda N işlemci için $O(N)$ yük getirir

Çözüm 2: Broadcast ile veriyi tüm işlemcilere dağıt

Çözüm 3: Multicast: Sadece veriyi okumak isteyen işlemcilere dağıt

Multicast: N yapraklı bir ağaç oluştur, ağacın kökü erişilmek istenen bellek gözü. Her bir bellek gözü için $2N-1$ bellek gözü daha ayırmalıyız. Ayrıca her işlemci iki yerel değişken içermeli [$level(i)$: işlemci isteğinin ulaştığı ağaç düzeyi ; $loc(i)$: isteğin ulaştığı ağaç düğümü]

```
procedure MULTIPLE_BROADCAST(d1, d2, ...)
1       for i=1 to N do in parallel // sadece veriye ulaşmak isteyen işlemciler
                level(i) = 0
                loc(i) = N+i-2
                d(i) + loc(i) <-- [i] yaz
        end for
2       for v=0 to log(N-2) do // köke kadar taşıma işlemi
2.1             for i=1 to N do in parallel
                        x = floor((loc(i)-1)/2) // anne düğümün konumu
                        if loc(i) % 2 = 1 and level(i) = v then
                                loc(i) = x
                                d(i) + loc(i) <-- [i] yaz // isteğini üst seviyeye taşı
                                level(i) = level(i) + 1
                        end if
                end for
```

```
2.2           for i=1 to N do in parallel // sıradaki sağ çocuk
                    if d(i) + x içinde bir [j] yer almıyor ise (1 ≤ j ≤ N) then
                          loc(i) = x
                          d(i) + loc(i) <-- [i] yaz // isteğini üst seviyeye taşı
                          level(i) = level(i) + 1
                    end if
              end for
       end for
3      for v=log(N-1) down to 0 do
3.1           for i=1 to N do in parallel
                    x = floor((loc(i)-1)/2) // anne düğümün konumu
                    y = 2.loc(i) + 1 // sol çocuğum
                    if loc(i) tek and level(i) = v ise then
                          d(i) + x içeriğini oku
                          d(i) + loc(i) <-- okunan içeriği yaz
                          level(i) = level(i) - 1
                          if d(i) + y'de [i] var ise then
                                loc(i) = y // soldan istek gelmiş
                                loc(i) = y + 1 // sağdan istek gelmiş
                          end if
                    end if
              end for
3.2           for i=1 to N do in parallel // sıradaki sağ çocuk
                    if loc(i) çift and level(i) = v ise then
                          d(i) + x içeriğini oku
                          d(i) + loc(i) <-- okunan içeriği yaz
                          level(i) = level(i) - 1
                          if d(i) + y'de [i] var ise then
                                loc(i) = y // soldan istek gelmiş
                                loc(i) = y + 1 // sağdan istek gelmiş
                          end if
                    end if
              end for
       end for
end
```

Verilen bu yöntem $O(\log N)$'lik karmaşıklıktadır. Ayrıca, $n(2N-1)$ bellek gereksinimine ihtiyaç duymaktadır.

EREW Kaynaştırma algoritması:

Yardımcı prosedür: iki sıralı diziyi kaynaştırmadan medyanı bul

```
procedure TWO_SEQUENCE_MEDIAN(A, B, x, y)
1      low_A = 1
       low_B = 1
       high_A = r
       high_B = s
       n_A = r
       n_B = s
2      while n_A > 1 and n_B > 1 do
              u = low_A + ceil((high_A - low_A - 1)/2) // medyanın konumu
              v = low_B + ceil((high_B - low_B - 1)/2) // medyanın konumu
              w = min(floor(n_A/2), floor(n_B/2)) // küçük olan dizi
              n_A = n_A - w
              n_B = n_B - w
              if a_u ≥ b_v then
                    high_A = high_A - w
                    low_B = low_B + w
              else
                    high_B = high_B - w
                    low_A = low_A + w
              end if
       end while
3      {a_u-1, a_u, a_u+1} x {b_v-1, b_v, b_v+1} // u ve v elimizde
       medyan olma koşullarını doğrulayan çiftin x ve y indislerini geri getir
end
```

Artık EREW MERGE prosedürünü aşağıdaki gibi gerçekleyebiliriz.

```
procedure EREW_MERGE(A, B, C)
1       P1 işlemcisi (1, r, 1, s) bilgisini elde eder
1.2     for j=1 to log(N) do
              for i=1 to i=2^(j-1) do in parallel // kaç işlemci çalışacak
                     (e, f, g, h) dörtlüsünü elde eden Pi işlemcisi
1.2.1                TWO_SEQUENCE_MEDIAN(A[e,f], B[g,h], x, y) // iki alt dizinin medyanı
1.2.2                {p1, p2, q1, q2} indislerini hesaplar
                     if ax medyan ise then
                            p1 = x
                            q1 = x + 1
                            if by ≤ ax then
                                   p2 = y
                                   q2 = y + 1
                            else
                                   p2 = y - 1
                                   q2 = y
                            end if
                     else
                            p2 = y
                            q2 = y + 1
                            if ax ≤ by then
                                   p1 = x
                                   q1 = x + 1
                            else
                                   p1 = x - 1
                                   q1 = x
                            end if
                     end if
                     Pi işlemcisi P2i-1 (e, p1, g, p2) dörtlüsünü gönderir
                     Pi işlemcisi P2i (q1, f, q2, h) dörtlüsünü gönderir
2       // Kaynaştırma
        for i=1 to N do in parallel
              (a, b, c, d) dörtlüsüne sahiplen // alt dizi indisleri
              w = 1 + ceil((i-1)(r+s)/N)
              z = min(ceil(i(r+s)/N), r+s)
              SEQUENTIAL_MERGE(A[a,b], B[c,d], C[w,z])
        end for
end
```

SIRALAMA

Sıralanmamış durumdaki n elemanlı bir S dizisi sıralanacak, en kötü çözüm $O(n \log n)$ ile guick sort

```
procedure QUICK_SORT(S)
        if |S| = 2 and s2 < s1 then
              s1 <--> s2 // swap
        else if |S| > 2
              m = SEQUENTIAL_SELECT(S, ceil(|S| / 2)) // S'in medyanını bul
              S1 = {si : si ≤ m} and |S1| = ceil(|S| / 2) // S'i iki alt diziye ayır
              S2 = {si : si ≥ m} and |S2| = floor(|S| / 2)
              QUICK_SORT(S1)
              QUICK_SORT(S2)
        end if
end
```

Lineer dizi üzerinde sıralama, işlemciler sadece yanlarındaki işlemcilerle iletişimde bulunabilir:
$(\dots - P_{i-1} - P_i - P_{i+1} - \cdots)$

```
procedure ODD_EVEN_TRANSPOSITION(S)
        for j=1 to ceil(n / 2) do
              for i=1,3,...,2.floor(n / 2)-1 do in parallel
                     if xi > xi+1 then
                            xi <--> xi+1
                     end if
              end for
              for i=2,4,...,2.floor((n-1) / 2) do in parallel
                     if xi > xi+1 then
                            xi <--> xi+1
                     end if
              end for
        end for
end
```

Bu yöntemin karmaşıklığı $O(n)$ olmakla birlikte, n işlemciye gerek duyduğundan, toplam malyeti $O(n^2)$ mertebesinde

n işlemci yerine daha anlamlı olabilecek $N \ll n$ işlemci kullanalım. Sırasız alt diziyi n/N 'lik alt dizilere bölelim. Her birini sıralayıp, odd-even yöntemindeki gibi kaynaştıralım.

```
procedure MERGE_SPLIT(S)
1       for i=1 to N do in parallel
              QUICK_SORT(Si)
        end for
2       for j=1 to ceil(N/2) do
2.1           for i=1,3,...,2.floor(n / 2)-1 do in parallel
                    SEQUENTIAL_MERGE(Si, Si+1, Si')
                    // Si <-- {s1', s2', ..., sn/N' }
                    // Si+1 <-- {sn/N+1', sn/N+2', ..., s2n/N'}
              end for
2.2           for i=2,4,...,2.floor((n-1) / 2) do in parallel
                    SEQUENTIAL_MERGE(Si, Si+1, Si')
                    // Si <-- {s1', s2', ..., sn/N' }
                    // Si+1 <-- {sn/N+1', sn/N+2', ..., s2n/N'}
              end for
        end for
end
```

Yöntemin karmaşıklığı $O(\frac{n}{N}\log\frac{n}{N} + N)$

CRCW Modeli için çözümü inceleyeceğiz. CW çelişkilerini gidermek için, 1'den fazla sayıda işlemci aynı bellek gözüne yazmak isterse, toplamlarını yazacağız. Elimizde n^2 adet işlemci var. Yöntemin adı "Sorting by enumeration". Eğer $s_i = s_j$ ve $i > j$ ise $s_i > s_j$ kabul et. Yöntem, her s_i elemanı için, dizideki s_i'den küçük eleman sayısını bularak ilerler.

```
procedure CRCW_SORT(S)
1       for i=1 to n do in parallel
              for j=1 to n do in parallel // tüm Pi,j işlemcilerini seç
                    if si > sj or (si = sj and i > j) // her elemanın sırası belirlenir
                          then Pi,j işlemcisi C(i) gözüne 1 yazar
                    else
                          0 yaz // gerekli değil
                    end if
              end for
        end for
2       for i=1 to n do in parallel // tüm Pi,j işlemcilerini seç
              Pi,1 işlemcisi Sc(i)+1 gözüne Si yazar
        end for
end
```

$O(1)$ karmaşıklıkta çalışabilecek, ancak çok güçlü bir donanım isteyen bir algoritma.

CREW modelinde sıralama yapmak istediğimizde, $N \ll n$ adet işlemci kullanacağız. Kaynaştır – Böl mantığıyla ilerler.

```
procedure CREW_SORT(S)
1       for i=1 to N do in parallel
              n/N boyutundaki Si alt dizisini oku
              QUICK_SORT(Si)
              Si' <-- Si
              Pi' <-- {Pi} // kaynaştırma adımında etkili olacak işlemci kümesi
        end for
2       u = 1
        V = N
        while v > 1 do
              for m=1 to floor(v/2) do in parallel
                    Pm u+1 <-- P2m-1 u U P2m u
              end for
              CREW_MERGE(S2m-1 u, S2m u, Sm u+1)
              if v tek sayı then
                    Pceil(v/2) u+1 <-- Pv u
                    Sceil(v/2) u+1 <-- Sv u
              end if
              u <-- u + 1
              v <-- ceil(v/2)
        end while
end
```

EREW modelini incelersek, aynı bellek gözüne birden fazla sayıda işlemci aynı anda erişemediğini varsayıyoruz. QUICKSORT benzeri bir çalışma kullanacağız. 1 adet medyan kullanmak yerine, N adet kullanacağız.

```
procedure EREW_SORT(S)
        if |S| ≤ K
                QUICK_SORT(S)
        else
1               for i=1 to K-1 do
                        PARALLEL_SELECT(S, sqrt(i), |S|/k) // m_i medyanları belirlenir
                end for
2               S_1 <-- {s ε S : s ≤ m_1} // ilk alt dizi
3               for i=2 to K-1 do
                        S_i <-- {s ε S : m_i-1 ≤ s ≤ m_i}
                end for
4               S_k <-- {s ε S : s ≥ m_k-1}
5               for i=1 to K/2 do in parallel
                        EREW_SORT(S_i)
                end for
6               for i=K/2+1 to K do in parallel
                        EREW_SORT(S_i)
                end for
        end if
end
```

ARAMA

$S = \{s_1, s_2, \dots, s_n\}$ içinde $x = s_k$ değeri aranıyor.

```
procedure SEQUENTIAL_SEARCH(S, x, k)
1       i = 1
        k = 0
2       while i ≤ n AND k = 0 do
                if s_i = x then
                        k = i
                end if
                i = i+1
        end while
```

Sırasız dizi için $O(n)$, sıralı dizi için $O(\log n)$

Sıralı Dizide Arama

$S = \{s_1, s_2, \dots, s_n\}$ için $s_i \leq s_j$ $(i < j)$ geçerli

EREW modeli (paralel erişim söz konusu değil)

- $N < n$ işlemci var, x'i arıyoruz (tüm işlemciler öğrenmeli, dolayısıyla BROADCAST ile dağıt $O(\log N)$)
- Her işlemciye S'in n/N'lik bir parçası düşer $P_i \leftarrow \{s_{(i-1)\frac{n}{N}+1}, \dots, s_{(i)\frac{n}{N}}\}$

Her işlemci kendi bölgesine binary search uygular, $O(\log \frac{n}{N})$

Toplam maliyet $O(\log N) + O\left(\log \frac{n}{N}\right) = O(\log n)$, gereksiz!

CREW model

- Broadcast'e gerek yok, CR ile 1 adımda tüm işlemciler okur

1. Her işlemci kendi alt dizisini oluşturur
2. Her işlemci binary search uygular $O(\log \frac{n}{N})$

Paralel okuma ile binary search hızlandırılabilir mi?

$N = 1$, binary search işleminde dizi ikiye bölünür,
$N > 1$ için, dizi $N + 1$ parçaya bölünsün

- Her işlemci kendi alt dizisinin sağ sınırında kalan alan ile x'i karşılaştırır. İşlem sonucuna göre, $N + 1$ dizinin N adeti elenir. Kalan alt dizi üzerinde işlem tekrar edilir.
- Binary search $2^d - 1$ adımda, $N + 1$ search $(N + 1)^d - 1$ adımda sonuca ulaşır
- d sayıda işlem ile sonuca ulaşılır, $d = \left\lceil \frac{\log(n+1)}{\log(N+1)} \right\rceil$. Böylece $(\log n \rightarrow \log N)$

```
procedure CREW_SEARCH(S, x, k)
1       q = 1
        r = n
        k = 0
        g = ceil(log(n+1) / log(N+1))
        while q ≤ n AND k = 0 do
                j0 = q-1
                for i=1 to N do in parallel
                        ji = (q - 1) + i(N + 1)^(g-1)
                        if sji = x then
                                k = ji
                        else if sji > x then
                                ci = left
                        else
                                ci = right
                        end if
                        if ci ≠ ci-1 then
                                q = ji-1 + 1
                                r = ji - 1
                        end if
                        if i = N AND ci ≠ ci-1 then
                                q = ji + 1     // sadece son işlemci yürütür
                        end if
                end for
                g = g - 1 // düzey sayısı
        end while
```

Sırasız Dizide Arama

```
procedure SM_SEARCH(S, x, k)
1       for i=1 to N do in parallel
                Read x
        end for
2       for i=1 to N do in parallel
                Kendi dizinin sırasını belirle
                SEQUENTIAL_SEARCH(Si, x, ki)
        end for
3       for i=1 to N do in parallel
                if ki > 0 then
                        k = ki  // çelişki giderme yöntemi
                end if
        end for
```

EREW

1. BROADCAST → $O(\log N)$
2. $S_i \leftarrow n/N$ → $O(n/N)$
3. Herkes aynı bellek gözüne $O(\log N)$
   $T \rightarrow O(\log N) + O(n/N)$
   N işlemcinin toplam maliyeti $O(N \log N) + O(n)$

ERCW

1. ve 2. aynı
3. sabit sürede
   $O(\log N) + O(n/N)$

CREW

1. sabit sürede
   gerisi aynı
   $O(\log N) + O(n/N)$

CRCW

1. ve 3. Sabit
   $O(n/N)$
   N işlemcinin toplam maliyeti $O(n)$

AĞAÇ MİMARİSİ

Ağaçta $n = 8$ eleman varsa (sadece yapraklarda), ağaçta $2n - 1$ işlemci

Düğümler: Kök (dış dünyayla bağlantı), ara, yaprak (veriler)

T adet arama işlemini, $\log n + T - 1$ adımda hesaplar

MESH MİMARİ

n elemanlı S dizisi, $N = n$ işlemci, $n^{1/2} \times n^{1/2}$ 'lik bir matris halinde bağlı

```
procedure MESH_SEARCH(S, x, sonuc)
1       P1,1, x'i okur
2       if x = s1,1 then
                b1,1 = 1
        else
                b1,1 = 0
        end if
3       for i=1 to n^1/2-1 do   // dağıtma = öğrenme
                for j=1 to i do in parallel
                        Pj,i, (bj,i ve x) değerlerini Pj,i+1'e aktarır
                        if x = si,j+1 OR bj,i = 1 then
                                bj,i+1 = 1
                        else
                                bj,i+1 = 0
                        end if
                end for
                for j=1 to i do in parallel
                        Pj,i, (bj,i ve x) değerlerini Pj+1,i'e aktarır
                        if x = si+1,j OR bj,i = 1 then
                                bj+1,i = 1
                        else
                                bj+1,i = 0
                        end if
                end for
        end for
4       for n^1/2 down to 2 do   // geri toplama
                for j=1 to i do in parallel
                        Pj,i, (bj,i) değerini Pj,i-1'e aktarır
                end for
                for j=1 to i-1 do in parallel
                        bj,i-1 = bj,i
                end for
                if bi,i-1 = 1 OR bi,i = 1 then
                        bi,i-1 = 1
                else
                        bi,i-1 = 0
                end if
                for j=1 to i-1 do in parallel
                        Pi,j, (bi,j) değerini Pi-1,j'e aktarır
                end for
                for j=1 to i-2 do in parallel
                        bi-1,j = bi,j
                end for
                if bi-1,i-1 = 1 OR bi,i-1 = 1 then
                        bi-1,i-1 = 1
                else
                        bi-1,i-1 = 0
                end if
        end for
5       P1,1 sonuç üretir
        if b1,1 = 1 then
                sonuc = BULUNDU
        else
                sonuc = BULUNAMADI
        end
```

MATRİS İŞLEMLERİ

Transpozunu Bulma $O(n^2)$

```
procedure TRANSPOSE(A)
1       for i=2 to N do
                for j=1 to i-1 do
                        swap(aij, aji)
                end for
        end for
```

MESH MİMARİSİ

$n \times n$ matris için $n \times n$ elemanlı mesh mimarisi
- Başlangıçta P_{ij} işlemcisinde a_{ij}
- Algoritma sonlandığında, a_{ji} bulunacak
- Her P_{ij} işlemcisinde a_{ij}, b_{ij} ($P_{i,j+1}$ sağ + üst komşudan gelen), c_{ij} (sol + alt komşudan gelen) değişkenleri tutuluyor

```
procedure MESH_TRANSPOSE(A)
1       (1.1) ve (1.2) paralel yürütülür (bilgilerin yüklenme aşaması)
1.1     for i=2 to n do in parallel
              for j=1 to i-1 do in parallel
                    veri = (a_i,j) ve hedef = (j, i) verilerini üst komşuya gönder c_i-1,j
              end for
        end for
1.2     for i=1 to n-1 do in parallel
              for j=i+1 to n do in parallel
                    veri = (a_i,j) ve hedef = (j, i) verilerini sol komşuya gönder b_i,j-1
              end for
        end for
2       (2.1) ve (2.2) ve (2.3) paralel yürütülür
2.1     for i=2 to n do in parallel
              for j=1 to i-1 do in parallel
                    while P_i,j'nin komşulardan veri girişi oldukça do
                          if (a_km,m,k) üçlüsü P_i+1,j'den gelmişse then
                                Bu bilgiyi P_i-1,j'ye gönder
                          end if
                          if (a_km,m,k) üçlüsü P_i-1,j'den gelmişse then
                                if i=m AND j=k then
                                      a_i,j = a_k,m  // hedefe ulaştı
                                else
                                      Bu bilgiyi P_i+1,j'ye gönder
                                end if
                          end if
                    end while
              end for
        end for
2.2     for i=1 to n do in parallel
              while P_i,i'nin komşularından veri girişi olduğu sürece do
                    if (a_km,m,k) üçlüsü P_i+1,i'den gelmişse (alttan) then
                          Bu bilgiyi P_i,i+1'e gönder (sağa)
                    end if
                    if (a_km,m,k) üçlüsü P_i,i+1'den gelmişse (sağdan) then
                          Bu bilgiyi P_i+1,i'e gönder (aşağı)
                    end if
              end while
        end for
2.3     for i=1 to n-1 do in parallel
              for j=i+1 to n do in parallel
                    while P_i,j'nin komşulardan veri girişi oldukça do
                          if (a_km,m,k) üçlüsü P_i,j+1'den gelmişse then
                                Bu bilgiyi P_i,j-1'e gönder
                          end if
                          if (a_km,m,k) üçlüsü P_i,j-1'den gelmişse then
                                if i=m AND j=k then
                                      a_i,j = a_k,m // hedefe ulaştı
                                else
                                      Bu bilgiyi P_i,j+1'e gönder
                                end if
                          end if
                    end while
              end for
        end for
```

Matris Çarpımı

$A[m,n] \times B[n,k] = C[m,k]$ // alt sınır $O(n^2)$

$c_{i,j} = \sum_{s=1}^{n} a_{i,s} * b_{s,j}$ şeklinde hesaplanacaktır. Bilinen en iyi çözüm $O(n^x)$, $(2 < x < 3)$

```
procedure MATRIX_MULTIPLICATION(A, B, C)
1       for i=1 to m do
              for j=1 to k do
                    c_i,j = 0
                    for s=1 to n do
                          c_i,j = c_i,j + (a_i,s * b_s,j)
                    end for
              end for
        end for
```

MESH MİMARİSİ MATRİS ÇARPIMI

A ve B matrisleri mesh'e düzgün sırada yanlardan verilir, her düğüm çarpımları hesaplayarak toplar
P_{ij} işlemcisi c_{ij} elemanını taşır ve ilk değeri sıfırdır
- iki giriş bilgisi (a, b) hazır olunca
1. $a * b$ hesaplanır
2. c_{ij}'ye toplam eklenir
3. $j \neq k$ (işlemci son sütun elemanı değilse) a değerini sağa gönderir
4. $i \neq m$ (son satır değilse) b değerini aşağı gönderir

```
procedure MESH_MATRIX_MUL (A, B, C)
1       for i=1 to m do in parallel
              for j=1 to k do in parallel
                    c_{i,j} = 0
                    while P_{i,j}, a ve b girişleri hazır olduğunda do
                          c_{i,j} = c_{i,j} + (a_{i,s} * b_{s,j})
                          if i < m then b → P_{i+1,j} end if
                          if j < k then a → P_{i,j+1} end if
                    end while
              end for
       end for
```

Ancak, n^2 işlemci kullandık, toplam maliyet yine $O(n^3)$

CRCW

$m \times n \times k$ adet işlemciden oluşan $3B$ bir mimari; yazma çelişkileri: yazmak işleyen istemcilerin değerleri toplamı

```
procedure CRCW_MATRIX_MUL (A, B, C)
1       for i=1 to m do in parallel
              for j=1 to k do in parallel
                    for s=1 to n do in parallel
                          c_{i,j} = 0
                          c_{i,j} <-- (a_{i,s} * b_{s,j})
                    end for
              end for
       end for
```

$O(1)$ sürede çözüm bulunur, ancak toplam maliyet $O(n^3)$

SINAV

- Dizi ters çevrilecek, her işlemci kendine düşen kısmı belirleyip swap uygulamalı
- Ağaç yapısında toplamlar: değeri anneye gönder, gelenleri topla ve anneye gönder, soldan gelen bilgiyi sağa aktar

Matris Dizi Çarpımı

$A[m,n] \times U[n,1] = V[m,1]$ $v_{i,j} = \sum_{j=1}^{n} a_{i,j} * u_j$ şeklinde hesaplanacak

Lineer Dizi Üzerinde Çarpma

İşlemci dizisine; alttan dizi, yandan matris verilecek yeni dizi hesaplanacak
- başlangıç $v_i = 0$ Her P_i'de a, u ve v değişkenleri mevcut

Veri geldikçe

- $a = a_{i,j}$ ve $u = u_j$ ardından $a * u$ hesaplanır
- Çarpım sonucu v_i'ye eklenecek ardından $i \neq 1$ ise, u_j değerini P_{i-1}'e gönder

```
procedure LINEAR_MUL (A, U, V)
1       for i=1 to m do in parallel
              v_i = 0
              while P_i, a ve u girişleri hazır olduğunda do
                    v_i = v_i + (a * u)
                    if i > 1 then u → P_{i-1} end if
              end while
       end for
```

$n + m - 1$ adımda gerçekleşir, $n > m$ için $O(n)$ karmaşıklıkta, oysa ağaç $O(\log n)$

toplam $2n-1$ işlemci, ağaçta n yaprak, m adımda A matrisi ağacın n yaprağına verilir, toplam kökten dışarı verilir

```
procedure TREE_MV_MUL (A, U, V)
1       do 1.1 and 1.2 in paralel
1.1     for i=1 to n do in parallel  // yaprak işlemciler
              for j=1 to m do in parallel
                    u_i * a_ji çarpımını hesapla
                    çarpım sonucunu anneye gönder
              end for
        end for
1.2     for i=n+1 to 2n-1 do in parallel
              while P_i'nin her iki girişinde de veri olduğu sürece do
                    iki girişin toplamını hesapla
                    if i < 2n-1 then
                          sonucu anneye gönder
                    else
                          sonucu çıkışa gönder
                    end if
              end while
        end for
```

Paralel Graf Algoritmaları

$G=(V,E)$ grafı için Bitişiklik(Adjacency) matrisi A ($n \times n$'lik bir matris, 0 ve 1 değerlerinden oluşuyor veya ağırlıklar)

Bir v_i düğümünden başka bir v_j düğümüne ulaşılabiliyorsa, $v_i - v_j$ arasında yol vardır (her düğümden bir kez geç)

Yolun uzunluğu: Geçilen kenar sayısı

Bağlılık Matrisinin Hesaplanması (Connectivity)

Elemanları 0 ve 1'lerden oluşan, $c_{i,j}$ değerleri bulunacak

Boolean Matris çarpımının aynısı: çarpma yerine lojik AND, toplama yerine lojik OR

A bitişiklik matrisinden yararlanarak, B başlangıç matrisi oluşturulur ($b_{jk}=a_{jk}$ ve $b_{ii}=1$)

$B.B=B^2$ ve en uzun yol $n-1$ kenarlı olduğundan $C=B^{n-1}$, dolayısıyla $\log n$ çarpma işlemiyle sonuç hesaplanır

3 boyutlu matris çarpma işlemi kullanılabilir, küpün her elemanında A, B ve C saklayıcıları mevcut
İlk aşamada küpün $(0,j,k)$ yüzüne A matrisi yerleştirilir

```
procedure CUBE_CON (A, C)
1       Bitişiklik Matrisi köşegen <-- T
        for j=0 to n-1 do in parallel
              A_0,j,j = 1 // sadece P_0,j,j işlemcileri çalışır
        end for
2       A saklayıcısının içeriğini B'ye taşı
        for j=0 to n-1 do in parallel
              for k=0 to n-1 do in parallel
                    B_0,j,k = A_0,j,k
              end for
        end for
3       for i=1 to ceil(log(n-1)) do
              CUBE_MATRIX_MULTIP(A, B, C)
              for j=0 to n-1 do in parallel
                    for k=0 to n-1 do in parallel
                          A_0,j,k = C_0,j,k
                          B_0,j,k = C_0,j,k
                    end for
              end for
        end for
```

Küp matris çarpımı $\log n$'de çalıştığından, toplam karmaşıklık $O(\log^2 n)$

Bağlı Bileşenlerin Bulunması

Her $v_i - v_j$ çifti arasında bir yol varsa, bu tür graflara bağlı graflar denir.
Önce grafın bağlılık matrisi C oluşturulur. C'den yararlanılarak $n \times n$ 'lik D matrisi oluşturulur.
$d_{j,k}$ gözü için, eğer $c_{j,k}=1$ ise değeri v_k dır. Aksi durumda değeri 0.
En küçük numaralı sütundan başla (madem bağlılar), bileşen üyelerini bul.

Toplam $N = n^3$ adet işlemci kullanılır.

- Her işlemcide A, B ve C saklayıcıları mevcut
- Başlangıçta $A_{0,j,k} = a_{j,k}$ bitişiklik matrisi
- Algoritma sonlandığında $C_{0,j,0} = v_j$ bileşen numarası

```
procedure CUBE_COMPONENETS (A, C)
1       Bağlılık matrisini oluştur
        CUBE_CONNECTIVITY(A, C)
2       D matrisini oluştur
        for j=0 to n-1 do in parallel
                for k=0 to n-1 do in parallel
                        if c0jk = 1 then d0jk = vk end if
                end for
        end for
3       Bakalım satırın en küçüğünü nasıl bulacak
        for j=0 to n-1 do in parallel
                j satırındaki n işlemci, c0jl ≠ 0 olan en küçük l'yi belirler   // log n
                c0j0 = l
        end for
```

Tüm düğüm çiftleri arasındaki en kısa yol

- Yönlü ve ağırlıklandırılmış $G = (V, E)$, W $n \times n$'lik bir ağırlık matrisi
- Her $v_i - v_j$ çifti arasındaki en kısa yolu istiyoruz
- d_{ij}^k en fazla k adet kenardan oluşan i ve j düğümü arasındaki en kısa yol
- $d_{ij}^1 = w_{ij}$, v_i ve v_j düğümleri arasında tek kenardan oluşan yol
- Eğer aralarında kenar yoksa $w_{ij} = \infty$ $(w_{ii} = 0)$
- n düğümlü bir graf G, en fazla $n-1$ kenara sahip bir yol içerebilir: d_{ij}^{n-1}'e kadar hesaplamalıyız
- Diğer taraftan, $d_{ij}^k = \min\{d_{il}^{k/2} + d_{lj}^{k/2}\}$ şeklindeki toplam küçültülür
- Çarpım yerine toplama, toplam yerine min fonksiyonu kullanan matris çarpımıyla çözülebilir

Paralel çarpma: Cube Matrix multiplication ($N = n^3$)

- Her işlemcide A, B, C saklayıcıları mevcut
- İlk durumda ağırlık matrisi atanıyor: $A_{0,j,k} = w_{j,k}$
- Sonlanma durumu, $A_{0,j,k} = v_i$ ve v_i düğümleri arasındaki en kısa yolun uzunluğu

```
procedure CUBE_SHORTEST_PATH (A, C)
1       D1 matrisini oluştur
        for j=0 to n-1 do in parallel
                for k=0 to n-1 do in parallel
                        if j ≠ k AND A0,j,k ≠ 0 then
                                A0,j,k = ∞
                        end if
                        B0,j,k = A0,j,k   // B'ye O(1) sürede aynısını kopyala
                end for
        end for
2       for i=1 to ceil(log(n-1)) do
                CUBE_MATRIX_MULTIP(A, B, C)   // çarpma -> toplam, toplama -> min
                for j=0 to n-1 do in parallel
                        for k=0 to n-1 do in parallel   // O(1) sürede çözüm bulunur
                                A0,j,k = C0,j,k
                                B0,j,k = C0,j,k
                        end for
                end for
        end for
```

En Küçük Kapsama Ağacı (Min Spanning Tree)

- $G = (V, E)$ grafının en küçük kapsayan ağacı $G' = (V', E')$
- ağırlıklar toplamı en küçük, ağaç olduğundan bir çevrim yok, G'nin bir alt grafı
- önce, rasgele bir düğümü ağaca ekle
- ardından, c_i'nin komşuları arasından en yakın olanı ekle
- en yakın olanları güncelle
- elimizde bir grup var, o gruba en yakın olanı seçerek ilerle

```
procedure SEQUENTIAL_MST (W, TREE)
1       v0 ekle
        cvi = v0
2       Tüm düğümler ağaca eklenene kadar tekrarla
2.1             Ağaca en yakın düğümü ekle
                min w(vi, cvi)olanı bul
2.2             Ağaçta yer almayan her vi düğümü için cvi değerini güncelle
                cvi = min(wvi,cvi, wvj,vj)   // eskisi mi yeni eklenen mi daha yakın
```

EREW paralel EKKA

- N adet işlemci
- İşlemci sayısı düğüm sayısından bağımsız ($N = n^{1-x},\ 0 < x < 1$)
- Her işlemciye V düğüm kümesinin bir alt kümesi düşer (her birine n^x adet eleman)
- Her işlemci kendi V_p (düğüm kümesi) için c_{v_p} oluşturur
- G'nin ağırlık matrisi W ortak bellekte yer alır
- Herkes kendi en küçük önerisini bulsun, BROADCAST ile paylaşsın

```
procedure EREW_MST (W, TREE)
1       v0 düğümünü ağaca ekle
        for i=0 to N-1 do in parallel
                for Pi'nin sahip olduğu Vi'de yer alan tüm vj düğümleri için do
                        cvj = v0
                end for
        end for
2       for i=1 to n-1
                for j=0 to N-1 do in parallel
2.1                     Vj içinde ağaçta yer almayan Vp düğümleri içinden min dist(vp, cvp)'yi bul
                        Pi işlemcisi, dist(vr, ct) için (dj=dist,aj=vr,bj=vt) üçlüsünü oluşturur
2.2                     MINIMUM prosedürü kullanılarak en küçük dj (ayrıca aj ve bj) bulunur
                                (0 ≤ j ≤ N-1). as ağaca yeni eklenecek, bs ise önceden yer alıyordu
2.3                     P0 işlemcisi (as, bs) kenarını ağaca ekler
2.4                     BROADCAST(as) bilgisi tüm işlemcilere duyrulur
2.5                     bu bilgiyi elde eden tüm işlemciler için
                        for k=0 to N-1 do in parallel
i                               if vk, Vj içerisinde yer alıyorsa (benim elemanım) then
                                        vk = t   // ağaçta yer alıyor şeklinde işaretle
                                end if
ii                              for Vj'nin ağaçta yer almayan her vp düğümü için do
                                        if dist(vp, vk) < dist(vp, cvp) then
                                                cvp = vk
                                        end if
                                end for
                        end for
                end for
        end for
```