Optimálne zlučovanie

- **PRAM** = **Parallel Random Access Machine**
  - model s (neobmedzenou) zdieľanou pamäťou
  - **synchrónny model** – spoločné hodiny
  - **SIMD** – v každom kroku sa vykonáva rovnaká inštrukcia
  - vstup aj výstup v zdieľanej pamäti
  - každý procesor má ID a pozná celkový počet procesorov
  - pamäťové modely: EREW, CREW, CRCW (common, arbitrary, priority)
  - zrýchlenie: $S_p(n) = T^*(n) / T_p(n)$
  - cena: $C_p(n) = p.T_p(n)$

Work-time zápis algoritmov

- WT presentation framework:
  - zápis **abstrahujúci** od detailov PRAM implementácie
  - **neviaže sa na fixný počet** použitých procesorov
  - **postupnosť časových jednotiek**, kde každá môže obsahovať ľubovoľne veľa konkurentných operácií
  - akoby sekvenčný algoritmus s konštrukciou „paralelný for“, ktorá vykoná zadaný príkaz konkurentne pre všetky hodnoty **i** (i je identifikátor príslušného procesu)

```
for i := 1 to N pardo
    príkaz;
```

WT-charakteristiky

- **T(n)** – počet časových jednotiek, ktoré potrebuje algoritmus na vstup veľkosti n
- **W(n)** – celkový počet operácií, ktoré vykoná algoritmus na vstupe veľkosti n (práca algoritmu)

```
for i:=1 to n pardo
    A[i] := i;
```

Počet časových jednotiek: **1**
Počet vykonaných operácií: **n**

- **WT-rozvrhovanie** – každý WT-program vieme „skompilovať“ na program pre p-procesorový PRAM s časom behu: $T_p(n) <= W(n)/p + T(n)$

WT a optimalita

- **T(n)** – počet časových jednotiek (synchronizovaného času), ktoré potrebuje algoritmus na vstup veľkosti **n** (bez ohľadu na počet procesorov)
- **W(n)** – celkový počet operácií, ktoré vykoná algoritmus na vstupe veľkosti **n** (práca algoritmu)
- **T*(n)** – čas optimálneho sekvenčného algoritmu
  - optimálny algoritmus: $W(n) \in O(T^*(n))$
  - WT-optimálny algoritmus:
    $W(n) \in O(T^*(n))$ a $T(n)$ sa nedá zlepšiť
- ak je algoritmus optimálny, potom $S_p(n) \in \Theta(p)$ pre $p \in O(T^*(n)/T(n))$

Paralelné sčítanie vo WT

```
for i := 1 to n pardo
  B[i] := A[i];
// 1 krok, n operácií
for h := 1 to log n do
  for i := 1 to n/2^h pardo
    B[i] := B[2*i-1] + B[2*i];
// log n krokov, ∑_{h=1}^{log n} n/2^h operácií
S := B[1];
```

EREW
T(n) ∈ Θ(log n)
W(n) ∈ Θ(n)
$\mathbf{C_p(n)}$ ∈ Θ(n + p.log n)

T*(n) ∈ Θ(n)
W(n) ∈ Θ(T*(n)) optimálny

optimálne zrýchlenie
pre p ∈ Θ(n/log n)

pre CREW PRAM
T(n) ∈ Ω(log n), teda
WT-optimálny pre CREW

Prefixové súčty rekurzívne

```
if n = 1 then
  return {A[1]};
for i := 1 to n/2 pardo
  Y[i] := A[2*i-1]+A[2*i]

Rekurzívne vypočítaj prefixovú sumu
   pre pole Y a ulož ju do poľa Z

for i := 1 to n pardo
  if i = 1 then S[i] := A[i]
           else if i mod 2 = 0 then S[i] := Z[i/2]
                               else S[i] := Z[i/2] + A[i];
```

EREW
T(n) = O(1) + T(n/2)
W(n) = O(n) + W(n/2)
…
T(n) ϵ O(log(n))
W(n) ϵ O(n)

W(n) ϵ Θ(T*(n)) → optimálny algoritmus

Prefixové súčty nerekurzívne bez pomocných polí

```
for i := 1 to n pardo S[i] := A[i];
for h:=1 to log(n) do
   for i:=1 to n/2^h pardo
       S[i*2^h] := S[i*2^h] + S[i*2^h-2^(h-1)];
for h:=log(n)-1 downto 1 do
   for i:=1 to (n/2^h)-1 pardo
       S[i*2^h+2^(h-1)] := S[i*2^h+2^(h-1)] + S[i*2^h]
```

Zlučovanie usporiadaných postupností

- **Vstup:**

  usporiadané (neklesajúce) postupnosti **A** a **B**, |A| = n, |B| = m

  zjednodušenie: všetky prvky sú rôzne

- **Výstup:**

  usporiadaná postupnosť **C** = Merge(A, B) z prvkov postupností **A** a **B**

- Sekvenčný algoritmus:

  O(n+m) – (použitý v rekurzii MergeSort)

Rank – terminológia

- **rank(x:A)** – počet prvkov usporiadanej postupnosti **A** menších (alebo rovných) ako **x**
- **rank(A:B)** – postupnosť hodnôt (r_1, r_2, …, r_n) takých, že r_i = rank(A[i]:B)
- Ak poznáme rank(A:B) a rank(B:A), potom **zlučovanie** usporiadaných postupností je jednoduché:
  - C[rank(A[i]:B) + rank(A[i]:A)] := A[i];
    keďže A je utriedená: C[rank(A[i]:B) + i] := A[i];
  - C[rank(B[i]:A) + i] := B[i];

Paralelný rank

- Problém rank(A: B)
  - A aj B sú usporiadané
- Idea:
  - pre každý prvok ***a*** z ***A*** paralelne spustíme **binárne vyhľadávanie**
  - paralelný čas: O(log(|B|))
  - práca: O(|A|.log(|B|))
    - nie je optimálny – chceme prácu len O(|A|+|B|)

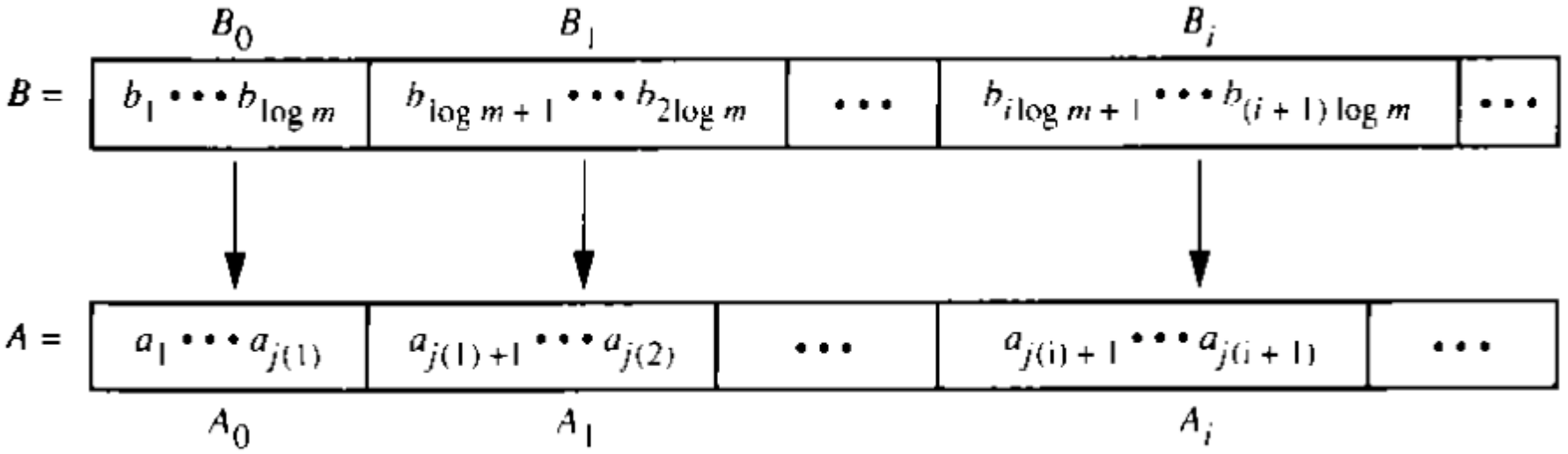
Paralelné zlučovanie

- Usporiadané postupnosti A, B také, že |A|= n,|B|=m
- **Idea:** rozdeľme si problém Merge(A, B) na menšie podproblémy Merge(A_i, B_i) také, že
  - A je zjednotením disjunktných A_i
  - B je zjednotením disjunktných B_i
  - každý prvok z Merge(A_{i-1},B_{i-1}) je menší ako každý prvok z Merge(A_i, B_i)
  - paralelne vyriešime Merge(A_i, B_i)

Paralelné zlučovanie (1)

- Konštrukcia (A_i, B_i):
  - rozdeľme B na k = m/log(m) súvislých blokov veľkosti log(m)
    $B_0, B_1, B_2, \ldots, B_{k-1}$, kde
    $B_i = B[i.\log(m)+1], \ldots, B[(i+1).\log(m)]$
  - j[i] := rank(B[i.log(m)]: A) (pre posledné prvky z množín B_i)
    - vypočítame paralelne sekvenčným binárnym vyhľadávaním
    - paralelný čas: O(log(n))
    - celková práca: m/log(m) blokov x O(log(n)) ϵ O(n+m)
  - $A_i = A[j[i]+1], \ldots, A[j[i+1]]$
    - j[0]=0; ak j[i] = j[i+1], potom A_i je prázdna postupnosť
    - potenciálny zvyšok A[j[k]+1], …, A[n] už bude usporiadaný

Paralelné zlučovanie (2)



- $a_{j(i)} < b_{i.\log m} < a_{j(i)+1}$ $\quad\quad$ $b_{i.\log m} < b_{i.\log m+1}$
- ak $|A_i| \in O(\log(n))$, potom **Merge(A_i, B_i)** môžeme vypočítať sekvenčne v čase $O(|A_i|+|B_i|) = O(\log(n)+\log(m))$ (m/log(m) blokov)
  (sekvenčný čas = paralelný čas aj paralelná práca)
- čo s blokmi, kde $|A_i| >> \log(n)$

Paralelné zlučovanie (3)

- ak $|A_i| > \log(n)$:
  - **Merge(A_i, B_i)** rozdelíme na menšie problémy tak, ako sme pôvodne rozdeľovali Merge(A, B) (vymeníme úlohu A a B)
  - A_i rozdelíme na bloky veľkosti log(n) : $A_{i,0}$, $A_{i,1}$, … pre koncový prvok každého bloku vypočítame jeho rank v B_i
    - $|B_i| = \log(m)$ -> ranky vieme vypočítať paralelným spustením sekvenčných binárnych vyhľadávaní v čase loglog(m) s prácou $|A_i|/\log(n) \times O(\text{loglog}(m)) \in O(|A_i| + |B_i|)$
  - ranky definujú rozdelenie B_i na príslušné podbloky
    - $(A_i, B_i) \rightarrow (A_{i,0}, B_{i,0}), (A_{i,1}, B_{i,1}), \ldots$ pre každý platí, že $A_{i,j} \in O(\log(n))$ a $B_{i,j} \in O(\log(m))$, teda môžeme riešiť Merge(A_i, B_i) ako v predchádzajúcom prípade (sekvenčným vyhľadávaním)

Paralelné zlučovanie - zhrnutie

Zlučovanie usporiadaných postupností **A, B** (|A| = |B| = n)

- **Merge(A, B)** rozbijeme na podproblémy **Merge(A_i, B_i)**, kde $|A_i|=O(\log(n))$ a $|B_i| = O(\log(n))$
  - Paralelný čas: $O(\log(n))$
  - Paralelná práca: $O(n)$ n/log(n) blokov po log(n) operácií
- Jednotlivé **Merge(A_i, B_i)** riešime paralelne sekvenčným algoritmom:
  - Merge(A_i, B_i) – sekvenčný čas zlúčenia $O(|A_i|+|B_i|) = O(\log(n))$
  - Spolu paralelný čas: $O(\log(n))$
  - Paralelná práca pre n/log(n) blokov: $O(n)$ - optimálny algoritmus
  - CREW PRAM (CR pri súbežných binárnych vyhľadávaniach)

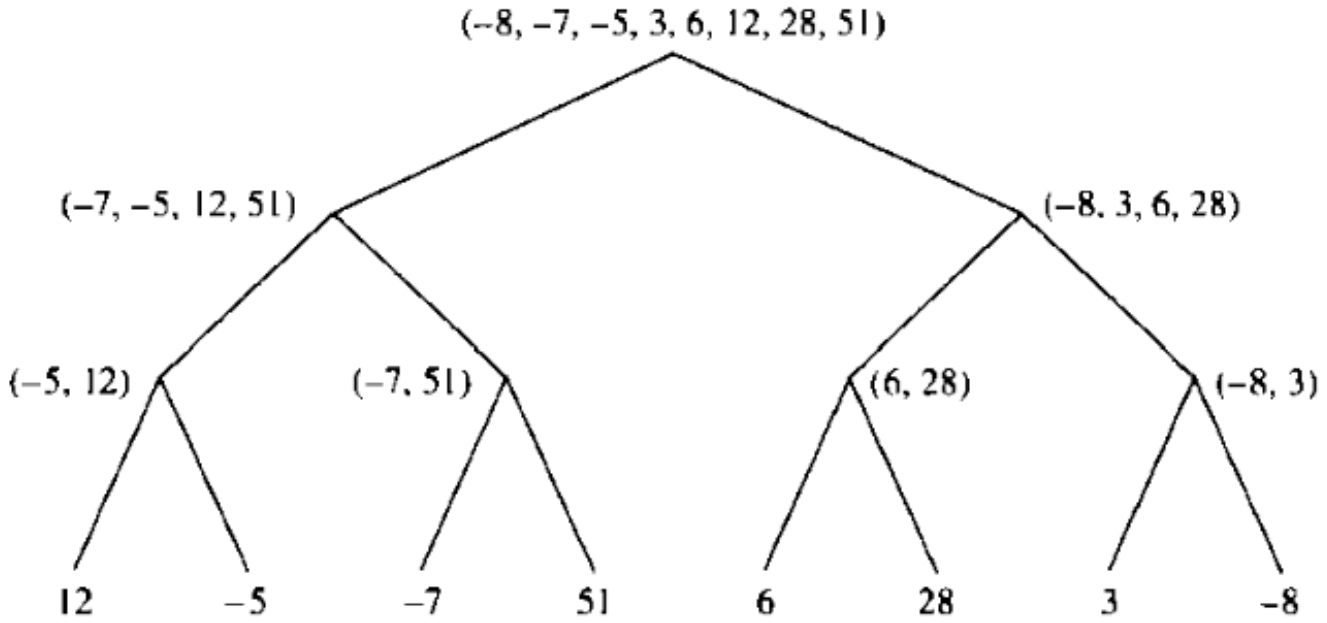
MergeSort rekurzívne a paralelne

```
MergeSort  (A[], n) {                          pre n = 2^k
if n = 1 then
  return {A[1]};

Rekurzívne usporiadaj obidve polovice poľa – každú pomocou
   n/2 procesorov – spolu paralelne pomocou n procesorov
  pardo ( MergeSort (A[1..n/2], n/2);
        MergeSort (A[n/2+1..n], n/2) );

Paralelne zlúč usporiadané polovice v čase O(log(n))
  return {Merge(A[1..n/2], A[n/2+1..n])};
}
```

Paralelný MergeSort



- Bottom-up pohľad:
  - začíname mergovaním postupností dĺžky 1
  - jednotlivé mergovania na rovnakej úrovni vieme realizovať paralelne

MergeSort paralelne bez rekurzie

```
MergeSort  (A[], n) {                          pre n = 2^k

if n = 1 then  return {A[1]};
for i := 1 to n/2 pardo
    if A[2*i-1] > A[2*i] then A[2*i-1] :=: A[2*i];
if n > 2 then  for h := 2 to log(n) do
   { paralelne zlúč n/2^h blokov veľkosti 2^h }
    for i := 0 to (n/2^h)-1 pardo
       Merge(A[i*2^h+1..i*2^h+2^h-1], A[i*2^h+2^h-1+1..i*2^h+2^h]);
return {A[]};
}
```

Paralelný MergeSort

- usporiadanie s použitím optimálneho zlučovania v paralelnom čase **O(log(n))**:
  - log(n) úrovní
  - na i-tej úrovni zlučujeme postupnosti dĺžky 2^i v čase $O(\log(2^i)) = O(i)$
  - paralelný čas: $1 + 2 + 3 + \ldots + \log(n) \in$ **$O(\log^2(n))$**
  - na každej úrovni máme optimálnu prácu **O(n)**
  - **$T(n) \in O(\log^2(n))$**
  - **$W(n) \in O(n.\log(n))$**
  - optimálny algoritmus pre $T(n) \in O(\log^2(n))$, CREW PRAM

Rýchlejšie optimálne usporiadania

- použitím techniky paralelného vyhľadávania a dvojlogaritmických stromov možno zrealizovať zlučovanie optimálne s prácou **O(n)** v čase **O(loglog(n))**
- potom paralelný MergeSort dosiahneme optimálne v čase O(log(n).loglog(n))

- zreťazením (pipelining) možno usporiadať postupnosť optimálne v čase O(log(n)) na CREW PRAM (JáJá 4.3.2)

- je možné usporiadať postupnosť v konštantnom čase ?

Ďakujem za pozornosť !