5. prednáška (14.3.2011)

Paralelné triedenie

PRAM - zopakovanie (1)

- PRAM = Parallel Random Access Machine
  - teoretický model
  - očíslované procesory pristupujúce k spoločnej zdieľanej pamäti
  - spoločné hodiny => **synchronizované vykonávanie**
  - každý (synchronizovaný) krok procesora má 3 fázy:
    - načítanie obsahu **nanajvýš jednej** pamäťovej bunky zo zdieľanej pamäte
    - lokálny výpočet
    - zapísanie obsahu do **nanajvýš jednej** pamäťovej bunky v zdieľanej pamäti

PRAM - zopakovanie (2)

- PRAM = Parallel Random Access Machine
  - rôzne varianty prístupu k pamäti: **EREW**, **CREW**, **CRCW** (common, arbitrary, priority)
- **Miery efektívnosti**:
  - p – počet procesorov
  - $T^*(n)$ – sekvenčný čas
  - $T_p(n)$ – paralelný čas pri p procesoroch
  - speed-up $S_p(n) = T^*(n) / T_p(n)$
  - cost $C_p(n) = p*T_p(n)$

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**

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

WT-miery efektívnosti

- **T(n)** – počet časových jednotiek, ktoré vykoná algoritmus na vstupe veľkosti n
- **W(n)** – celkový počet operácií, ktoré vykoná algoritmus na vstupe veľkosti n (práca algoritmu)
- **Optimalita:**
  - optimálny algoritmus: $W(n) = T^*(n)$
  - WT-optimálny algoritmus:
    $W(n)=T^*(n)$ a $T(n)$ sa nedá zlepšiť

Paralelné zlučovanie (1)

- **Vstup:**
  - usporiadané 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)** – podprocedúra použitá v MergeSorte

Rank - terminológia

- **rank(x:A)** – počet prvkov postupnosti **A** menších ako **x**
- **rank(A:B)** – postupnosť hodnôt $(r_1, r_2, \ldots, r_n)$ 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|))

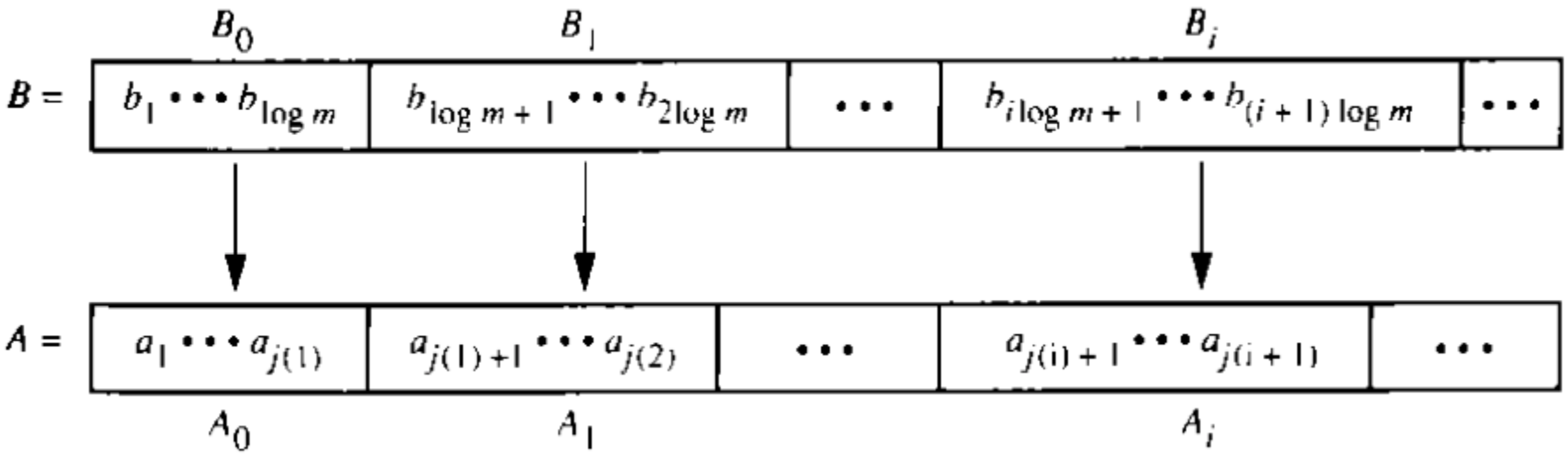
Paralelné zlučovanie (2)

- Usporiadané postupnosti A, B také, že $|A|=|B|=n$
- **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 (3)

- Konštrukcia (A_i, B_i):
  - Rozdeľme B na m=n/log(n) súvislých blokov veľkosti log(n): B_1, B_2, …, B_m, kde
    $B_i = B[i.\log(n)], \ldots, B[(i+1).\log(n)-1]$
  - j[i] := rank(B[i.log(n)]: A)
    - Vypočítame paralelne aplikovaním binárneho vyhľadávania
    - Paralelný čas: O(log(n))
    - Celková práca: n/log(n) blokov x O(log(n)) = O(n)
  - $A_i = A[j[i]+1], \ldots, A[j[i+1]]$
    - Ak j[i] = j[i+1], potom A_i je prázdna postupnosť

Paralelné zlučovanie (4)



- Ak $|A_i| = O(\log(n))$, potom **Merge(A_i, B_i)** vieme spraviť sekvenčne v čase $O(|A_i|+|B_i|) = O(\log(n))$. Čo s blokmi, kde $|A_i| > \log(n)$?
  - Sekvenčný čas = paralelný čas aj paralelná práca

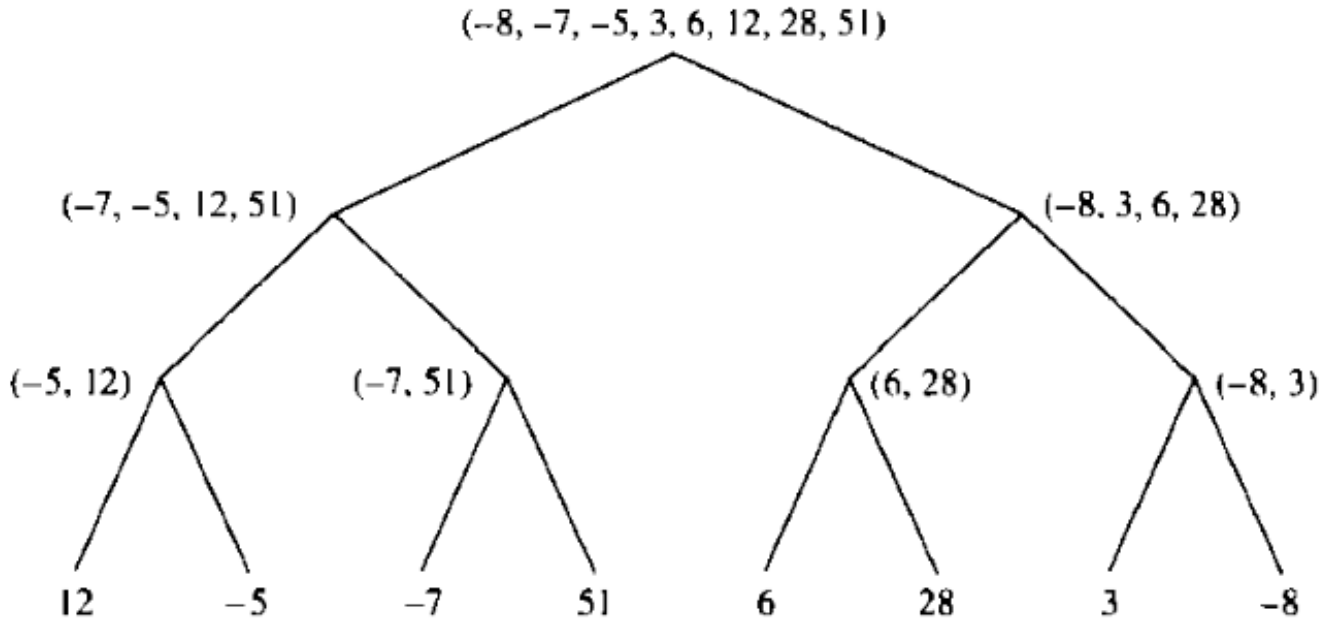
Paralelné zlučovanie (5)

- Ak $|A_i| > \log(n)$:
  - **Merge(A_i, B_i)** rozdelíme na menšie problémy tak, ako sme pôvodne rozdeľovali Merge(A, B)
  - A_i rozdelíme na bloky veľkosti log(n), $A_{i,0}$, $A_{i,1}$, ... pre začiatočný prvok každého bloku vypočítame jeho rank v B_i
    - $|B_i| = \log(n)$ -> ranky vieme vypočítať paralelným spustením binárnych vyhľadávaní v čase loglog(n) s prácou $|A_i|/\log(n) \times O(\text{loglog}(n)) = O(|A_i|)$
  - ranky definujú rozdelenie B_i na príslušné podbloky
    - (A_i, B_i) -> $(A_{i,0}, B_{i,0})$, $(A_{i,1}, B_{i,1})$, ... pre každý platí, že $A_{i,j}=O(\log(n))$ a $B_{i,j}=O(\log(n))$, t.j. máme predošlý prípad

Paralelné zlučovanie (6)

- Sumarizácia:
  - **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: $O(n)$
  - Jednotlivé Merge(A_i, B_i) riešime paralelne sekvenčným algoritmom:
    - Merge(A_i, B_i) – sekvenčný čas $O(|A_i|+|B_i|) = O(\log(n))$
    - Paralelný čas: $O(\log(n))$
    - Paralelná práca: $O(n)$ - optimálny algoritmus
    - CREW PRAM (CR pri súbežných bin. vyhľadávaniach)

Paralelný MergeSort (1)



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

Paralelný MergeSort (2)

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

Paralelné vyhľadávanie (1)

- **Vstup:**
  - utriedené pole prvkov $x[1] < x[2] < \ldots < x[n]$
  - $x[0]=-\infty$, $x[n]=\infty$
  - hodnota y
- **Výpočtový model:**
  - **p** PRAM procesorov (CREW)
- **Výstup:**
  - index **i** taký, že $x[i] <= y < x[i+1]$
- **Idea:** zovšeobecnenie binárneho vyhľadávania

Paralelné vyhľadávanie (2)

- Práca v krokoch (iniciálne l=0, r=n+1):
  - v každom kroku uvažujeme podpole x[l], …, x[r]
  - p procesormi rozdelíme podpole na p+1 segmentov, každému segmentu okrem prvého priradíme procesor: $[l=q_0, q_1-1], [q_1, q_2-1], \ldots, [q_p, r= q_{p+1}]$
  - procesor i pozrie hodnotu $x[q_i]$
  - ak procesor i identifikuje hodnotu ($x[q_i] = y$), tak končíme, inak pre ďalší krok uvažujeme segment i taký, že $x[q_i] < y < x[q_{i+1}]$
    - procesor spĺňajúci podmienku nastaví nové l a r pre ďalší krok (exclusive write), resp. P1, ak $y < x[q_1]$

Paralelné vyhľadávanie (3)

- Triviálny prípad:
  - ak r-l <= p, tak uvažujeme jednoprvkové segmenty
- Paralelný čas:

$$O(\log(n+1)/\log(p+1))$$

- Využijeme, že každým krokom zmenšíme problém p+1 násobne

Rank s použitím vyhľadávania

- Chceme spočítať **rank(A:B)**
  - **A** a **B** sú usporiadané, $|A|=O(n^s)$, $|B| = n$ a s je konštanta $0 < s < 1$
- Pre každý prvok z A hľadáme pozíciu v B paralelným vyhľadávaním s n^{1-s} procesormi
- Paralelný čas: $O(\log(n+1)/\log(n^{1-s}+1)) = O(1)$
- Paralelná práca:
  - $O(n^s)$ prvkov A x n^{1-s} procesorov x $O(1)$ krokov = $\mathbf{O(n)}$

loglog zlučovanie (1)

- Inšpirácia v dvojlogaritmickom strome pri paralelnom hľadaní maxima v čase $O(\log(\log(n))$ a paralelnom zlučovaní v čase $O(\log(n))$

- Usporiadané postupnosti A, B, $|A|=n$, $|B|=m$
- Chceme vypočítať **rank(B:A)**

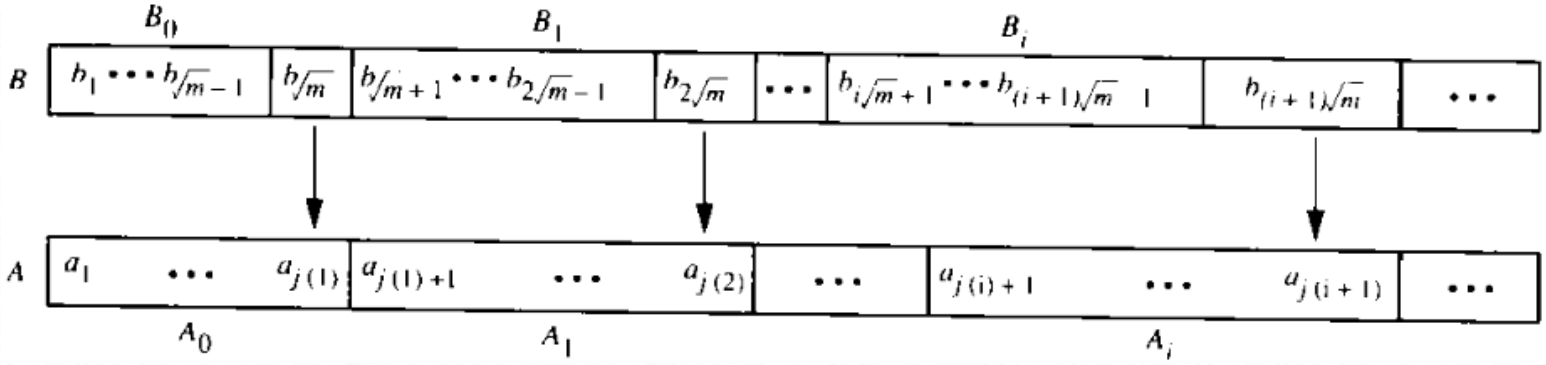
loglog zlučovanie (2.1)

- Rozdeľme **B** na $\sqrt{m}$ blokov veľkosti $\sqrt{m}$: B_0, B_1, …
  - $B_i = B[i\sqrt{m}+1], \ldots, B[(i+1)\sqrt{m}-1]$
- Spočítajme rank pre prvky $B[\sqrt{m}], B[2\sqrt{m}], \ldots, B[m])$, t.j., **rank(B[i√m]: *A*)**
  - aplikujeme paralelné vyhľadávanie s $p=\sqrt{n}$ procesormi na každý prvok
  - Čas: $O(\log(n+1)/\log(\sqrt{n}+1)) = \mathbf{O(1)}$
  - Práca: $\sqrt{m}$ prvkov x $O(\sqrt{n})$ práce na jedno hľadanie = $O(\sqrt{m}\sqrt{n}) = \mathbf{O(m+n)}$
    - $0 <= (\sqrt{m} - \sqrt{n})^2 = m - 2\sqrt{m}\sqrt{n} + n$
- k blokom B_i určíme prislúchajúce *A*-bloky:
  - $A_i = A[\text{rank}(B[i\sqrt{m}]:A)+1], \ldots, A[\text{rank}(B[(i+1)\sqrt{m}]:A)]$

loglog zlučovanie (2.2)

- Rekurzívne vypočítame rank(B_i: A_i)
- Ak b je z B_i, potom

$$\text{rank}(b: A) = \text{rank}(B[i\sqrt{m}]: A) + \text{rank}(b: A_i)$$



loglog zlučovanie (3)

- Paralelný čas:
  - $T(n, m) <= O(1) + \max_i T(n_i, \sqrt{m}) = O(1)+T(n, \sqrt{m})$
    - Po loglog(m) rekurzívnych vnoreniach majú B množiny veľkosť $O(1)$
  - **$T(n, m) = O(\log\log(m))$**
- Paralelná práca:
  - keďže práca jedného volania je $O(|A|+|B|)$, na jednej úrovni máme celkovú prácu $O(n+m)$
  - **$W(n, m) = O((n+m).\log\log(m))$**
  - CREW PRAM

loglog(n) optimálne zlučovanie

- postupnosti A, B také, že |A|=|B|=n
- A aj B rozdelíme do blokov veľkosti loglog(n)
  - A' a B' nech je postupnosť prvých prvkov v každom bloku: |A'|=|B'|=n/loglog(n)
- 1. Spočítame rank(A':B') a rank(B':A')
  - Aplikujeme neoptimálne O(loglog(n)) zlučovanie
  - Čas: O(loglog(n))
  - Práca: O((n/loglog(n)).loglog(n)) = O(n)

loglog(n) optimálne zlučovanie

- 2. Spočítame rank(A':B) a rank(B':A)
  - podľa rank(A':B') vieme, v ktorom B-bloku je každý prvok z A'
  - Každý B-blok má loglog(n) prvkov => sekvenčným algoritmom vieme nájsť rank(x: B) pre každý x z A'
  - analogicky rank(B':A)
  - Čas: O(loglog(n)) – sekvenčné hľadanie
  - Práca: n/loglog(n) prvkov x loglog(n) = O(n)

loglog(n) optimálne zlučovanie

- 3. Spočítame rank(A:B) a rank(B:A)
  - na základe A' a B' rozdelíme (A,B) na podbloky (A_i, B_i), kde $|A_i|=O(\log\log(n))$ a $|B_i|=O(\log\log(n))$
  - podobne ako pri zlučovaní v čase $O(\log(n))$
  - aplikovaním sekvenčného zlučovania dorátame rank
  - Čas: $O(\log\log(n))$ – bloky sú veľkosti $O(\log\log(n))$
  - Práca: $O(n)$ – rank cez sekvenčné zlučovanie
- Algoritmus celkom:
  - **$T(n) = O(\log\log(n))$, $W(n) = O(n)$**
  - WT-optimálny CREW

Sumarizácia

- Aplikovaním optimálneho algoritmu pre rank v čase **O(loglog(n))** dostávame optimálny paralelný MergeSort v čase O(log(n).loglog(n))

- Poznámka:
  - Aplikovaním pipe-liningu vieme skonštruovať optimálny triediaci algoritmus v čase O(log(n)) na CREW PRAM

Oblivious porovnávací algoritmus

- Nevšímavý (oblivious) porovnávací algoritmus nezávisí na konkrétnych hodnotách
  - v každom kroku má množinu dvojíc pamäťových miest
  - ak (x, y) je taká dvojica, potom ak x > y, tak obsah pamäťových miest sa vymení
- Zodpovedajú použitiu komparátorov („hardvérovo orientované porovnávanie")

Bitonická postupnosť

- x[1], x[2], …, x[n] je bitonická postupnosť práve vtedy, ak existujú i, l také, že:
  - x[i mod n] <= x[(i+1) mod n] <= …<=x[l mod n]
  - x[(l+1)mod n]>=x[(l+2)mod n]>= …>=x[(i+n-1) mod n]
- Inak:
  - existuje také cyklické posunutie postupnosti x, že postupnosť je najprv neklesajúca a potom nerastúca
- Príklad: 18 5 4 8 13 94 50 32 30

  4 8 13 94 50 32 30 18 5

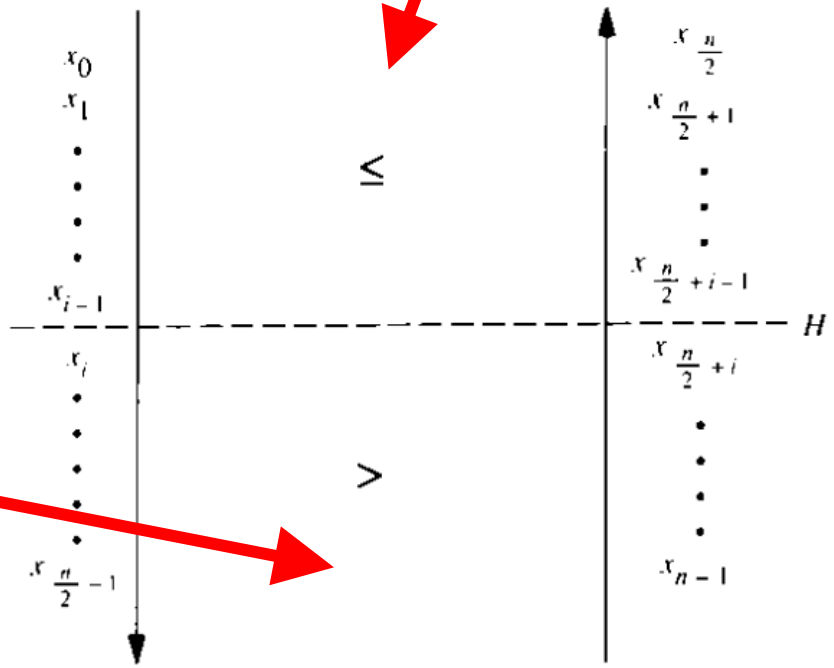
Jedinečné rozdelenie

- Ak x je bitonická postupnosť s párnym počtom prvkov taká, že
  - x[0] <= x[1] <=…<=x[n/2-1]
  - x[n/2] >= x[n/2+1] <=…<=x[n-1]

potom x možno rozdeliť na 2 oblasti:

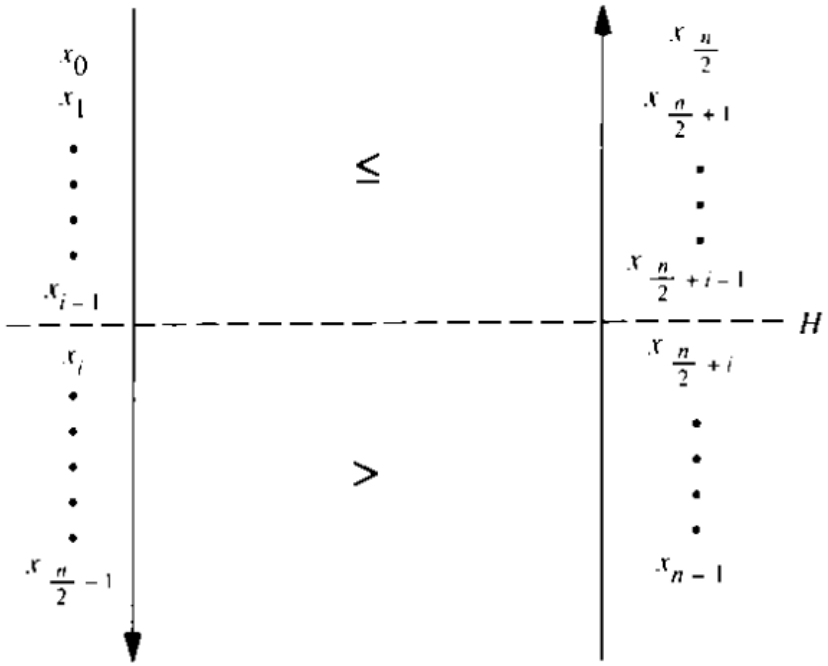
Každá hodnota vľavo je menšia alebo rovná ako každá hodnota vpravo.

Každá hodnota vľavo je väčšia ako každá hodnota vpravo.



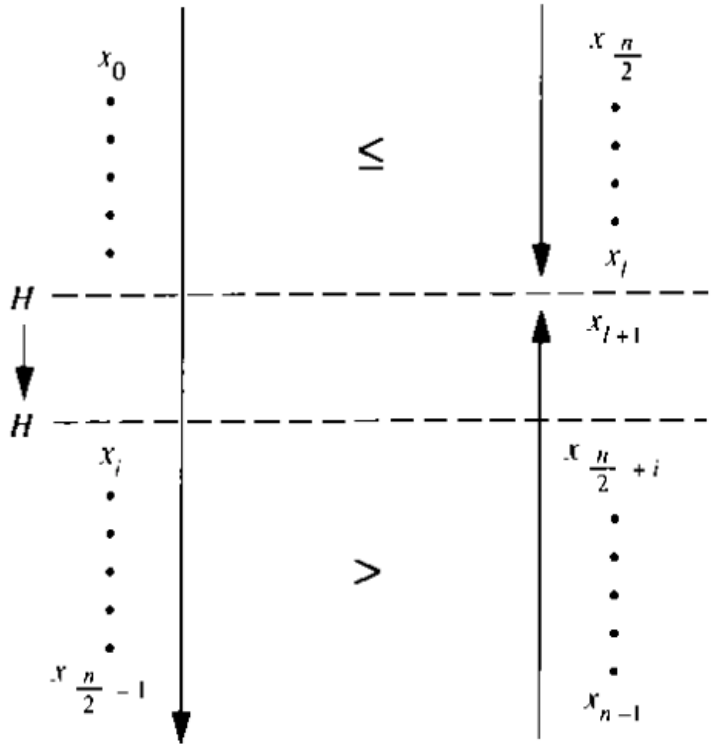
Jedinečné rozdelenie dôkaz

- Zoberme prvé i také, že x[i] > x[i+n/2], potom H je medzi x[i-1] a x[i]
- Všimnime si, že všetky požiadavky jedinečného rozdelenia sú splnené



Jedinečné rozdelenie všeobecne

- Každá bitonická postupnosť párnej dĺžky má vlastnosť jedinečného rozdelenia.
- Dôkaz:
  - predpoklad: i=0 a l>n/2
  - H je medzi x[l] a x[l+1], potom ju posúvame „dole" kým x[i] <= x[i+n/2]
  - cyklické posunutie postupnosti zachováva vlastnost jedinečného rozdelenia



Využitie rozdelenia

Nech x je bitonická postupnosť párnej dĺžky a

- L[i] = min(X[i], X[i+n/2]) pre i=0, …, n/2-1
- R[i] = max(X[i], X[i+n/2]) pre i=0, …, n/2-1

potom L aj R sú bitonické.

Dôkaz:

- Stačí si všimnúť, že obe postupnosti sú súvislé podpostupnosti originálnej postupnosti

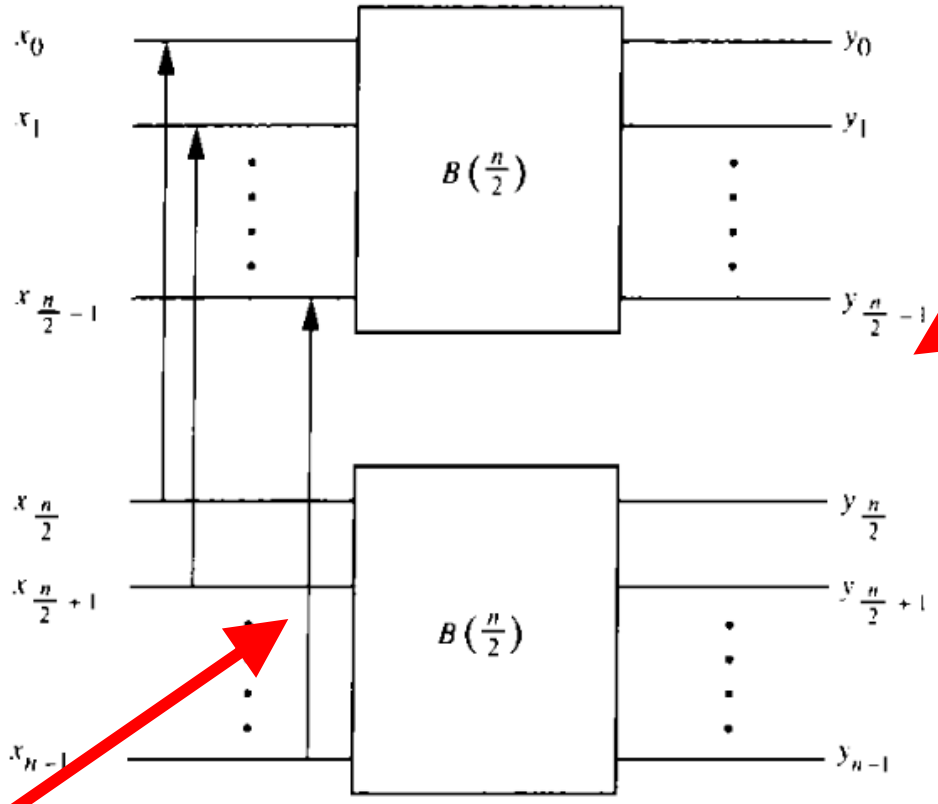
Bitonické triedenie

- **Vstup**: bitonická postupnosť X
- **Výstup**: usporiadaná bitonická postupnosť X
- ***Algoritmus***:

```
for i := 0 to n/2-1 pardo
begin
  L[i] := min(X[i], X[i+n/2]);
  R[i] := max(X[i], X[i+n/2]);
end;
```

rekurzívne (a paralelne) usporiadaj bitonické postupnosti L a R

Bitonické triedenie a komparátory



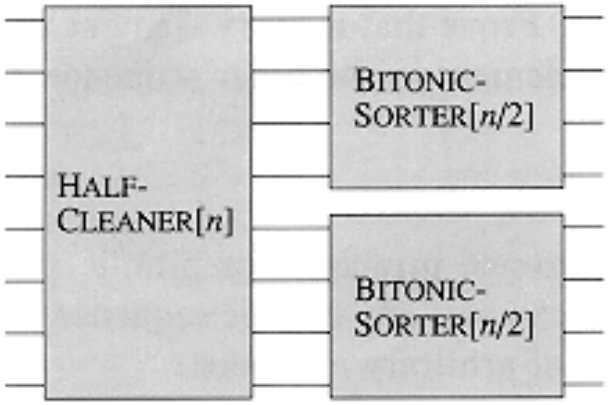
B(n) – triediaca sieť pre n-prvkovú bitonickú postupnosť

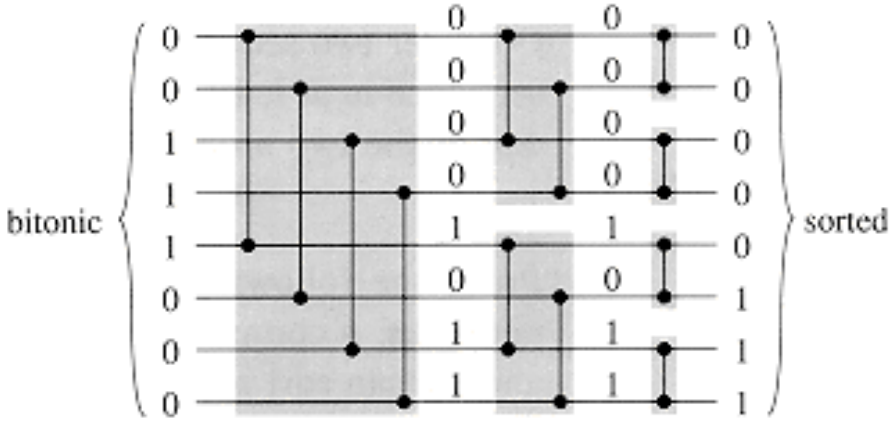
Komparátor vezme 2 hodnoty, porovná ich a ak treba, tak ich navzájom vymení (menšia v smere šípky)

B(n) sa skladá z dvoch **B(n/2)** a **n/2** komparátorov

Bitonické triedenie a komparátory

- Každý komparátor v triediacej sieti zodpovedá jednotke práce
- Komparátory porovnávajú hodnoty paralelne – každá úroveň komparátorov predstavuje časovú jednotku





Bitonické triedenie - analýza

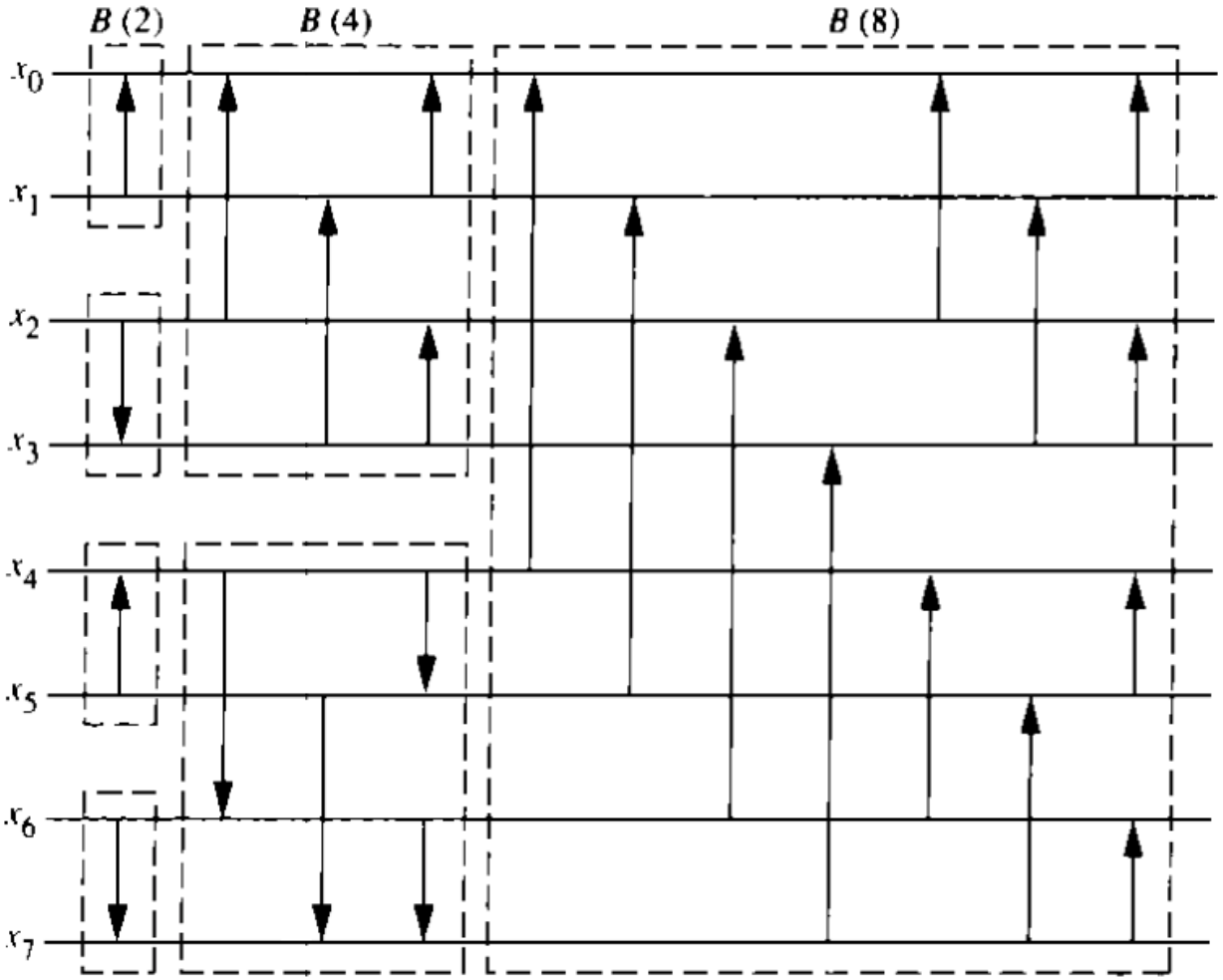
- Čas: $T(n) = O(\log(n))$
  - každým rekurzívnym volaním vzniknú polovičné problémy
- Práca: $W(n) = O(n.\log(n))$
  - na každej úrovni máme n/2 komparátorov, resp. porovnaní

- Ako triedit' nielen bitonické postupnosti?

Bitonický MergeSort (1)

- Idea: výsledkom rekurzívnych volaní v MergeSorte sú utriedené monotónne postupnosti
  - ich spojenie je bitonická postupnosť, ktorej usporiadaním dostaneme výsledok zlúčenia vstupných postupností
  - bitonické triedenie použijeme na zlučovanie
- $T(n) = O(\log^2(n))$
- $W(n) = O(n.\log^2(n))$
- EREW PRAM

Bitonický MergeSort (2)



0-1 princíp (1)

- Ak f je monotónna funkcia (x <= y => f(x) <= f(y)) potom:
  - min(f(x), f(y)) = f(min(x, y))
  - ak postupnosť $x_1, \ldots, x_n$ je vstup triediacej siete a $y_1, \ldots, y_n$ je výstup, potom pre vstup $f(x_1), \ldots, f(x_n)$ je výstup $f(y_1), \ldots, f(y_n)$

0-1 princíp (2)

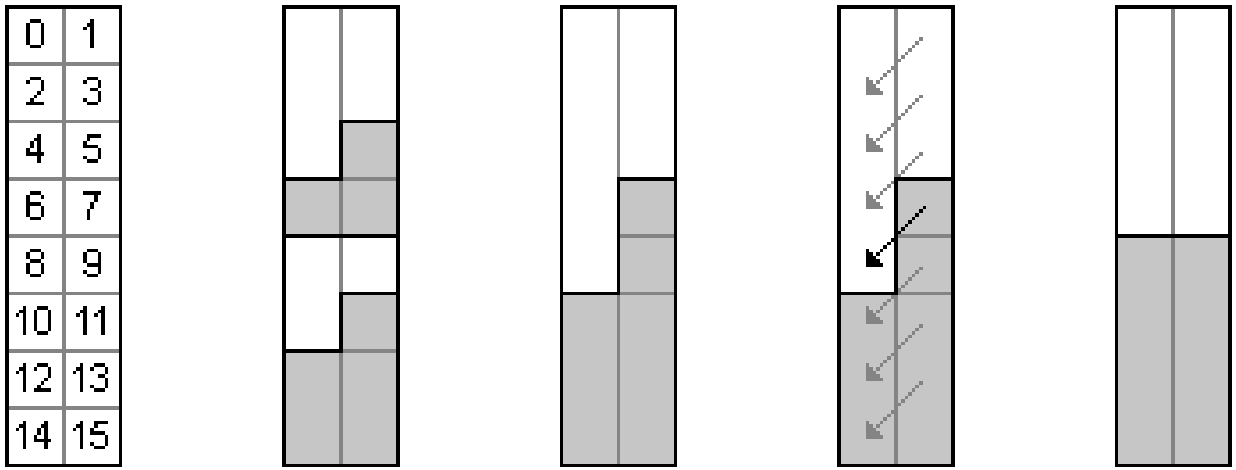
- Ak triediaca sieť triedi každú postupnosť núl a jednotiek, potom triedi každú postupnosť.
  - **Sporom**. Nech existuje postupnosť $x_1, x_2, \ldots, x_n$, ktorá nie je triediacou sieťou utriedená. Potom existuje výstup y_k taký, že $y_k > y_{k+1}$.
  - Definujme monotónnu funkciu f:
    - f(x) = 0, ak x < yk
    - f(x) = 1, ak x >= yk
  - Pre vstup $f(x_1),\ldots, f(x_n)$ je výstup $f(y_1), \ldots, f(y_n)$. Avšak $f(y_k) = 1$ a $f(y_{k+1})=0$, čo je spor s predpokladom, že sieť triedi všetky 0-1 postupnosti

Even-odd zlučovanie (1)

- **Vstup:** postupnosť *A* taká, že *A*[0], …, *A*[n/2-1] je utriedená a aj *A*[n/2], …, *A*[n] je utriedená
- **Výstup:** utriedená postupnosť *A*
- Algoritmus:
  - Rekurzívne aplikuj zlučovanie na prvky na párnych indexoch *A*[0], *A*[2], …
  - Rekurzívne aplikuj zlučovanie na prvky na nepárnych indexoch *A*[1], *A*[3], …
  - Compare(*A*[i], *A*[i+1]) pre i=1, 3, 5, …

Even-odd zlučovanie (2)

- Dôkaz indukciou na $n=2^i$ s využitím 0-1 princípu:



Mapovanie indexov

Vstup – šedé políčká sú 1

Po rekurzívnom aplikovaní zlučovania na párne a nepárne indexy – rozdiel vo „výškach“ je nanajvýš 2

Porovnanie za sebou idúcich dvojíc

Even-odd zlučovanie (3)

- Čas: **T(n) = O(log(n))**
  - T(n) = 1 + T(n/2)
- Práca: **W(n) = O(n.log(n))**
  - W(n) = n/2 + 2.W(n/2)

- Dôsledok: Even-odd MergeSort:
  - **Čas: $O(\log^2(n))$**
  - **Práca: $O(n.\log^2(n))$**

Ďakujem za pozornosť !