

4) Algoritmy řazení

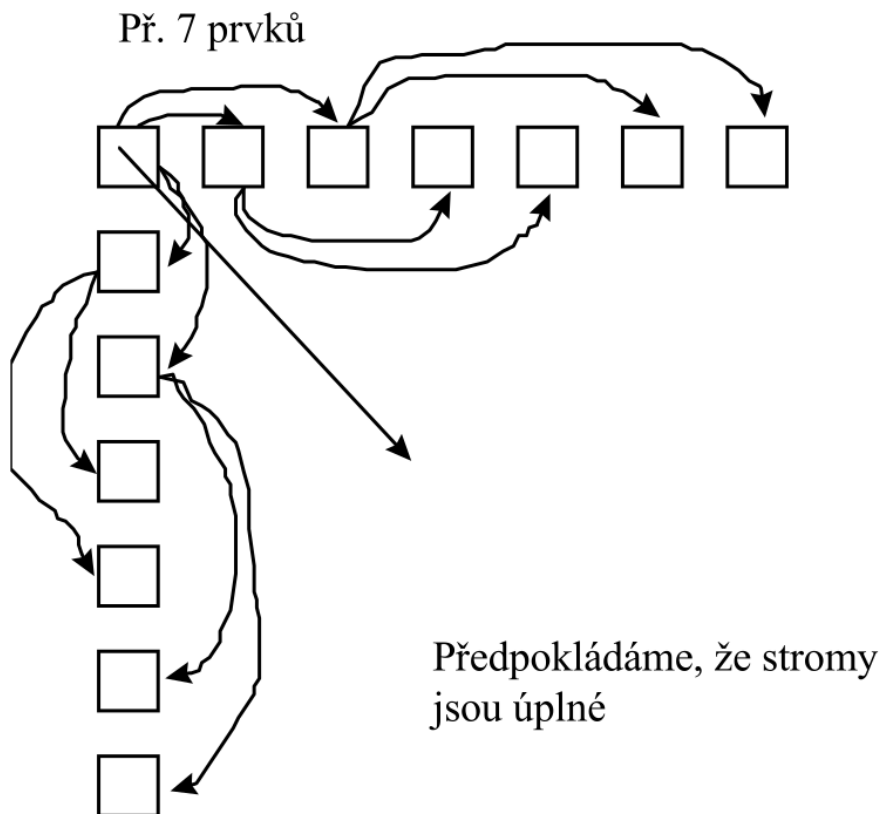
- Optimální sekvenční:
 - $p(n) = 1$
 - $t(n) = O(n * \log(n))$
 - $c(n) = O(n * \log(n))$

Enumeration sort (mřížka)

- Správná pozice každého prvku ve výstupní seřazené posloupnosti je dána počtem prvků, které jsou menší než tento prvek.
- Ideální algoritmus pro paralelní zpracování.

Topologie

- Na mřížce
- $p(n) = n^2$
- Extrémní případ, dosáhneme nejlepší časové složitosti
- Mřížka, kde procesory jsou vodorovně propojeny do stromu a svisle taky



Vlastnosti procesoru

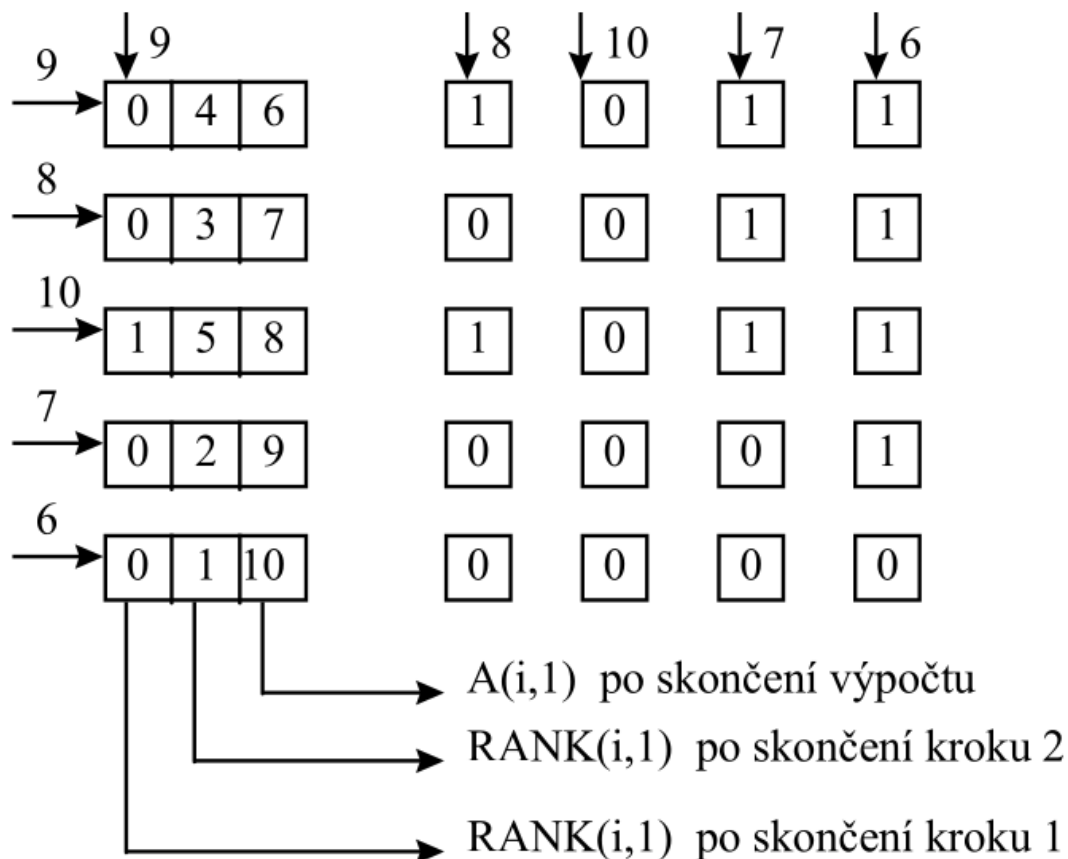
- Může uložit dva prvky do svých registrů A, B
- Může porovnat A a B a uložit výsledek do registru RANK
- Pomocí stromového propojení může předat obsah kteréhokoliv registru jinému procesoru
- Může přičítat k registru RANK

Algoritmus

1. Každý prvek je porovnán se všemi ostatními pomocí jedné řady procesorů
2. Správná pozice prvku je $RANK(x_i) = 1 + \text{počet menších prvků}$
3. Každý prvek je zadán na správné místo

Příklad

Př. $x = \{9, 8, 10, 7, 6\}$, u prvního sloupce pro všechny registry, u ostatních jen RANK



Analýza

- $t(n) = O(\log(n))$
- $c(n) = O(n^2 \cdot \log(n))$
- $p(n) = n^2$
- Není optimální

Diskuze

- Extrémně rychlý algoritmus $O(\log(n))$, což znamená zrychlení $O(n)$ oproti optimálnímu sekvenčnímu algoritmu.
 - Nic není rychlejší
- Spotřebovává příliš mnoho procesorů, na hranici přijatelnosti

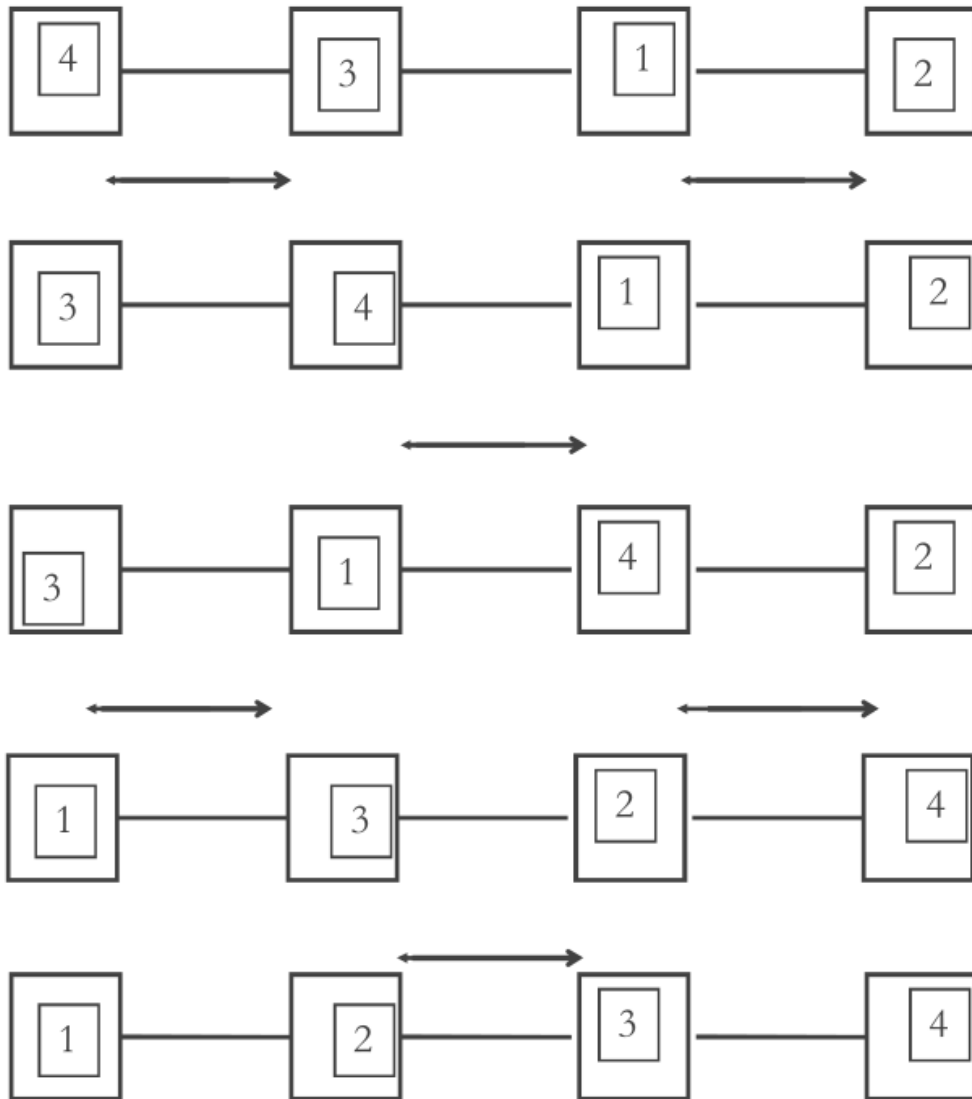
Odd-even transposition sort

- Paralelní bubble sort
- Lineární pole n procesorů $p(n) = n$

Algoritmus

- Na počátku každý procesor p_i obsahuje jednu z řazených hodnot y_i
- V prvním kroku se každý lichý procesor p_i spojí se svým sousedem p_{i+1} a porovnájí své hodnoty je-li $y_i > y_{i+1}$, procesory vymění své hodnoty
- V druhém kroku každý sudý procesor udělá totéž
- Po n krocích (maximálně) jsou hodnoty seřazeny

Příklad



Analýza

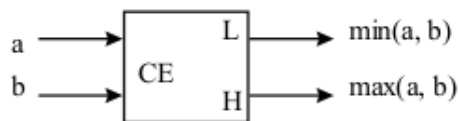
- Každý z kroků (1) a (2) provádí jedno porovnání a dva přenosy – konstantní čas
- Složitost: $t(n) = O(n)$
- Cena: $c(n) = t(n) * p(n) = O(n) * n = O(n^2)$
 - což není optimální
- Algoritmus má lineární časovou složitost, což je to nejlepší, čeho lze při lineární topologii dosáhnout

Odd even merge sort

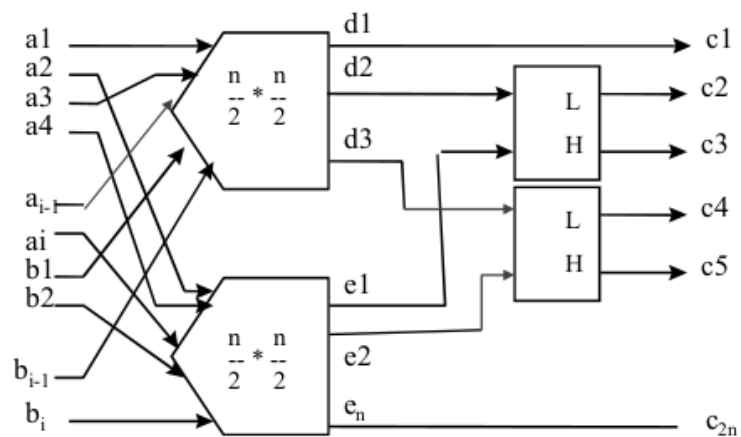
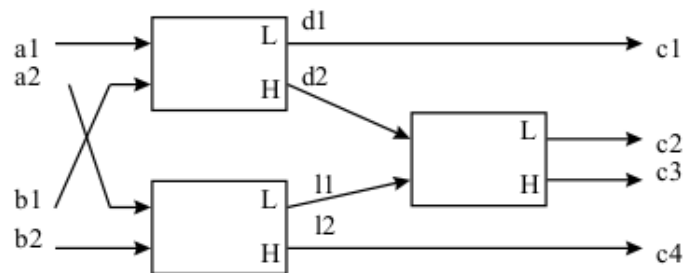
- Řadí se speciální sítí procesorů
- Každý procesor má dva vstupní a dva výstupní kanály
- Každý procesor umí porovnat hodnoty na svých vstupech, menší dá na výstup L (low), a větší dá výstup H (high).

Topologie

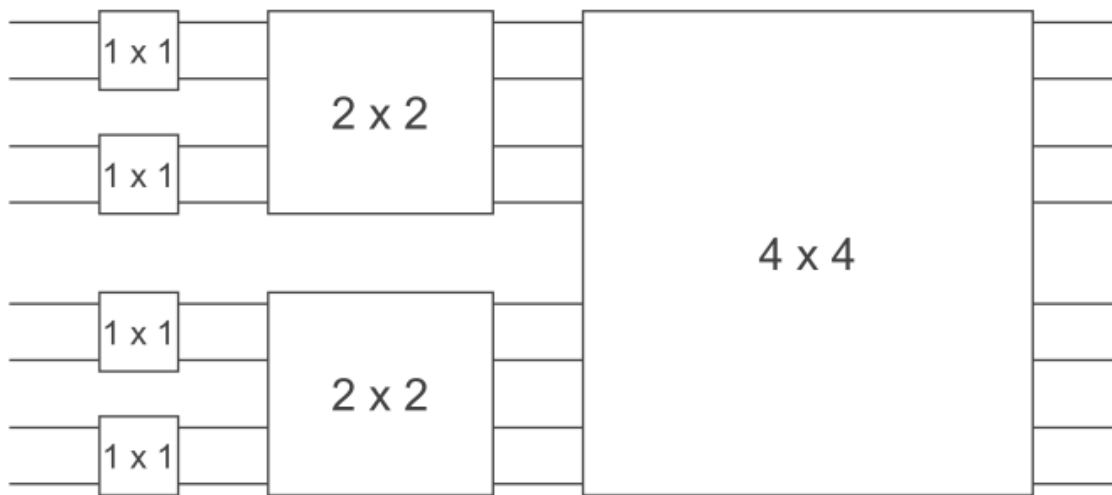
Sít' 1x1



2x2



Příklad



Analýza

- Řadíme posloupnost o délce $n = 2^m$
- 1.fáze potřebuje 2^{m-1} CE
- 2.fáze potřebuje 2^{m-2} sítí 2×2 po 3 procesorech
- 3.fáze 2^{m-3} sítí 4×4 po 9 procesorech
- 4.fáze 2^{m-4} sítí po 25 procesorech
- atd.

Časová složitost

- $t(n) = O(m^2) = O(\log^2(n))$

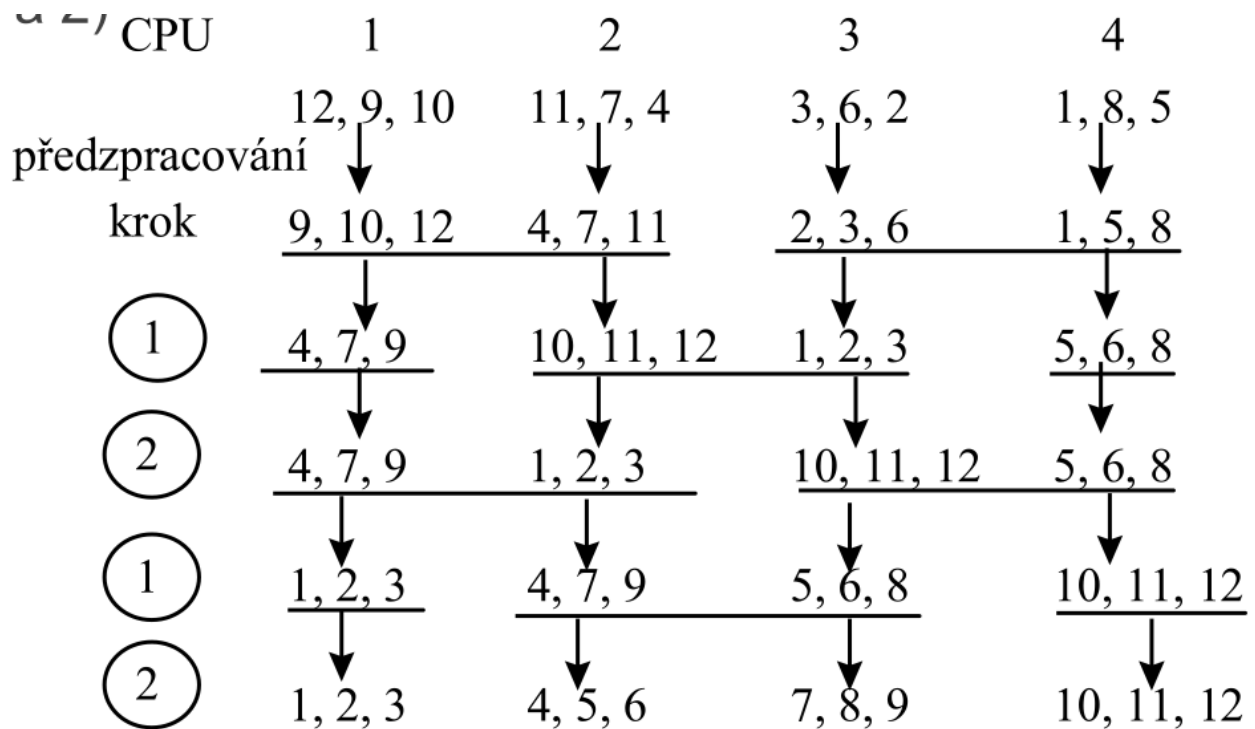
Cena

- $c(n) = O(n * \log^4(n))$
- Což není optimální

Merge-splitting sort

- Lineární pole procesorů $p(n) < n$
- Je variantou algoritmů lichý-sudý, kde každý procesor obsahuje několik čísel
- Porovnání a výměna je nahrazena operacemi merge-split
- Každý CPU se stará o více prvků
- Pomocí nastavení počtu procesorů jsme schopni nastavovat vlastnosti algoritmu

Příklad



Analýza

- Předzpracování optimálním alg.
 - $O((n/p) * \log(n/p))$
- Přenos S_{i+1} do P_i
 - $O(n/p)$
- Spojení S_i a S_{i+1} do S'_i optimálním alg.
 - $2 * n/p$
- Přenos S_{i+1} do P_{i+1}
 - $O(n/p)$
- Krok 1 nebo 2
 - $O(n/p)$

$$t(n) = O[(n/p) * \log(n/p)] + O(n) = O((n * \log(n))/p) + O(n)$$

$$c(n) = t(n) * p = O(n * \log n) + O(n * p)$$

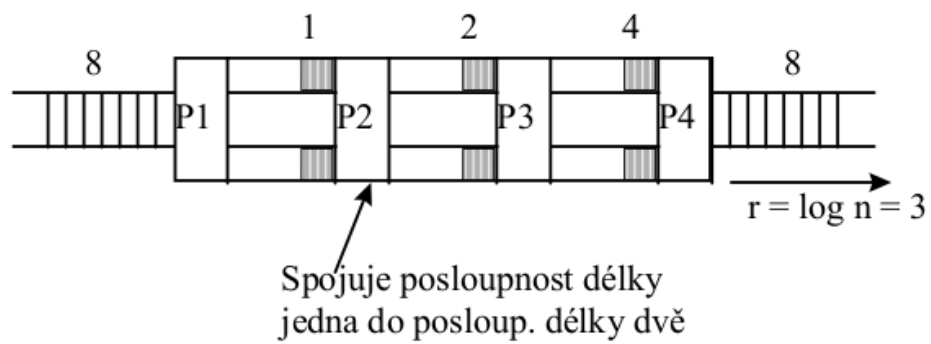
což je optimální pro $p \leq \log n$

Pipeline Merge sort

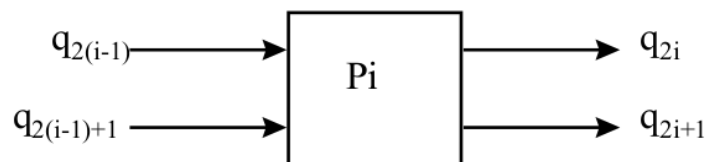
- Lineární pole procesorů $p(n) = \log n + 1$

- Data nejsou uložena v procesorech, ale postupně do nich vstupují
- Každý procesor spojuje dvě seřazené posloupnosti délky 2^{i-2}

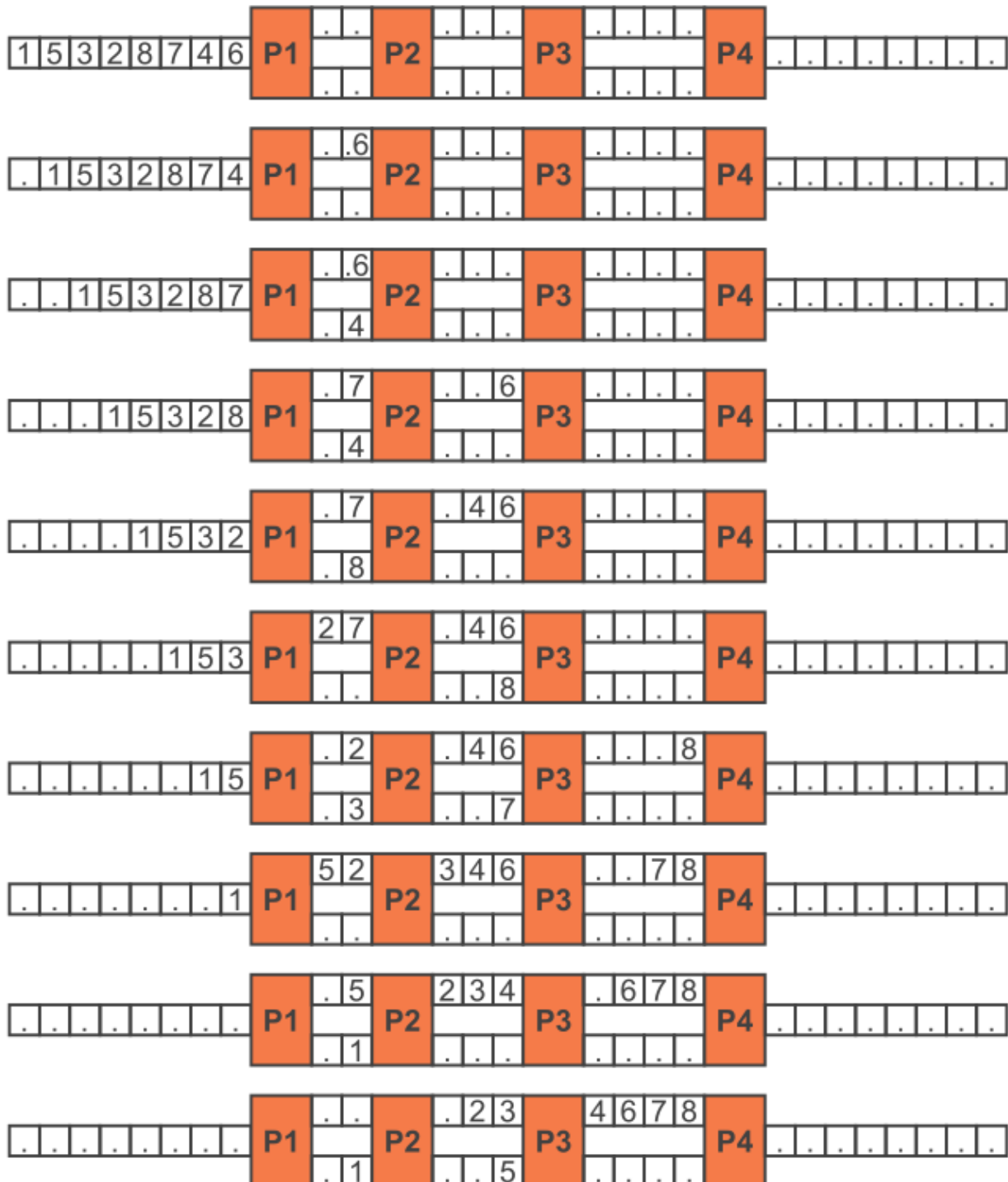
Topologie



Označení front :



Příklad



Analýza

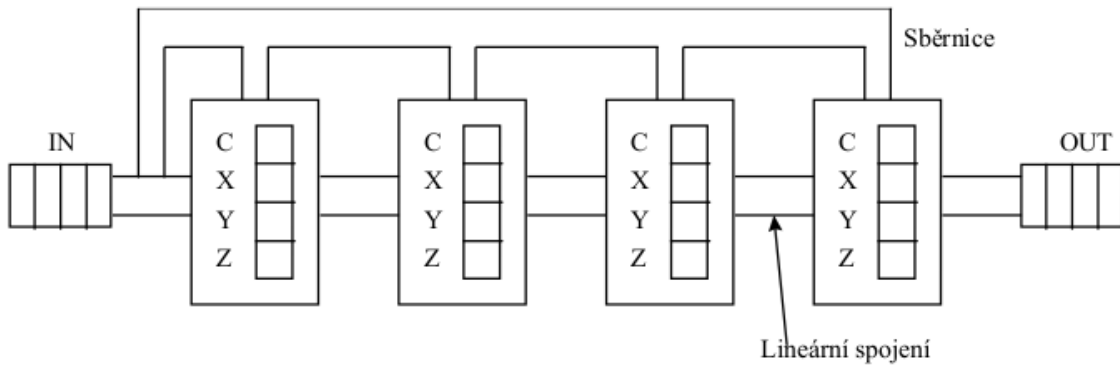
- $c(n) = t(n) * p(n) = O(n) * (\log n + 1) = O(n * \log n)$

- což je optimální

Enumeration sort (lineární pole)

- Lineární pole n procesorů, doplněných společnou sběrnicí, schopnou přenést v každém kroku jednu hodnotu.

Topologie



Příklad



Příklad

- Krok 1) je v konstantním čase, krok 2) trvá $2n$ cyklů, krok 3) trvá n cyklů
- $t(n) = O(n)$
- $c(n) = t(n) * p(n) = O(n) * n = O(n^2)$
- což není optimální

Minimum Extraction Sort

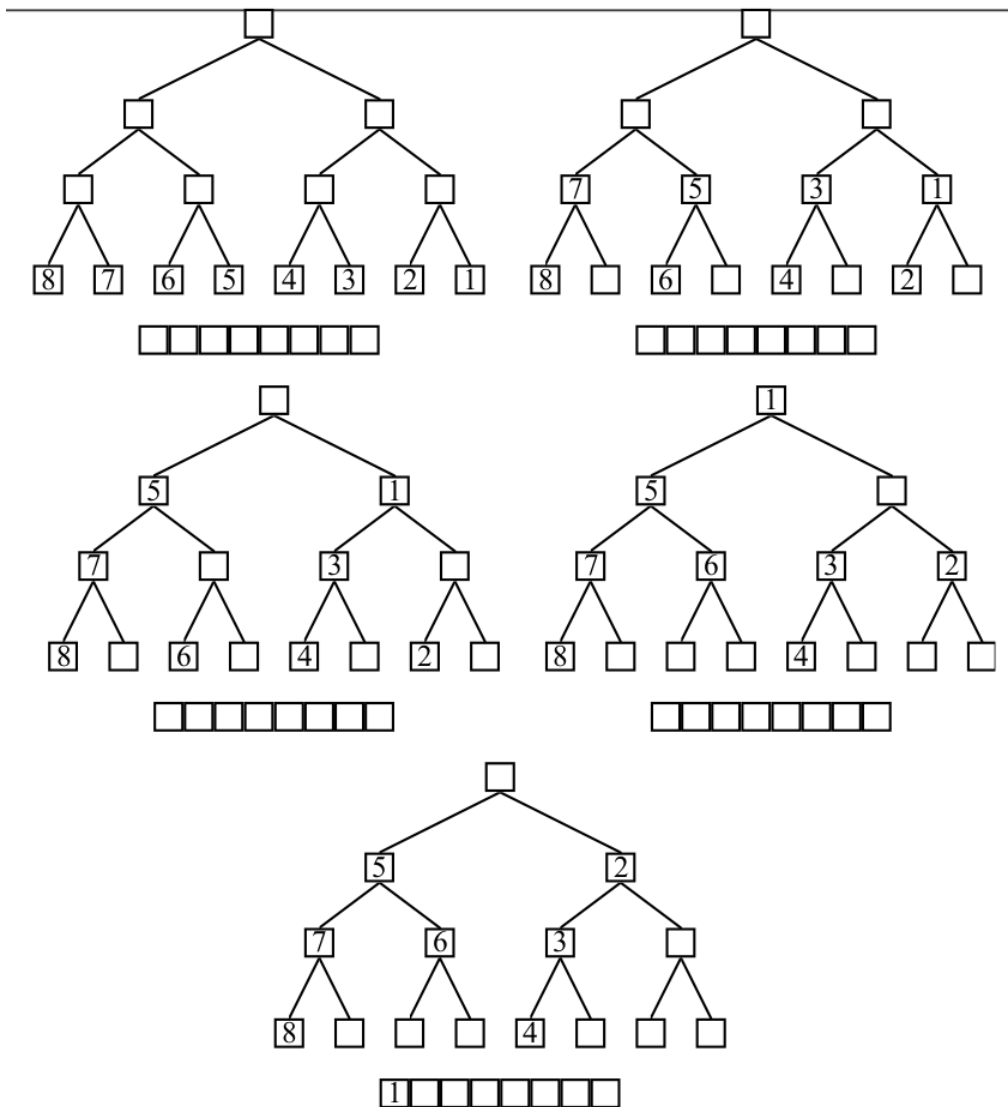
- Stromová architektura
- Každý procesor se stará o jeden prvek

Algorithmus

- Každý list obsahuje jeden prvek

- Každý nelistový procesor porovná hodnoty svých dvou synů a menší z nich pošle svému otci po $(\log n) + 1$ krocích se minimální prvek dostane do kořenového procesoru
- Každým dalším krokem se získá další nejmenší prvek

Příklad



Analýza

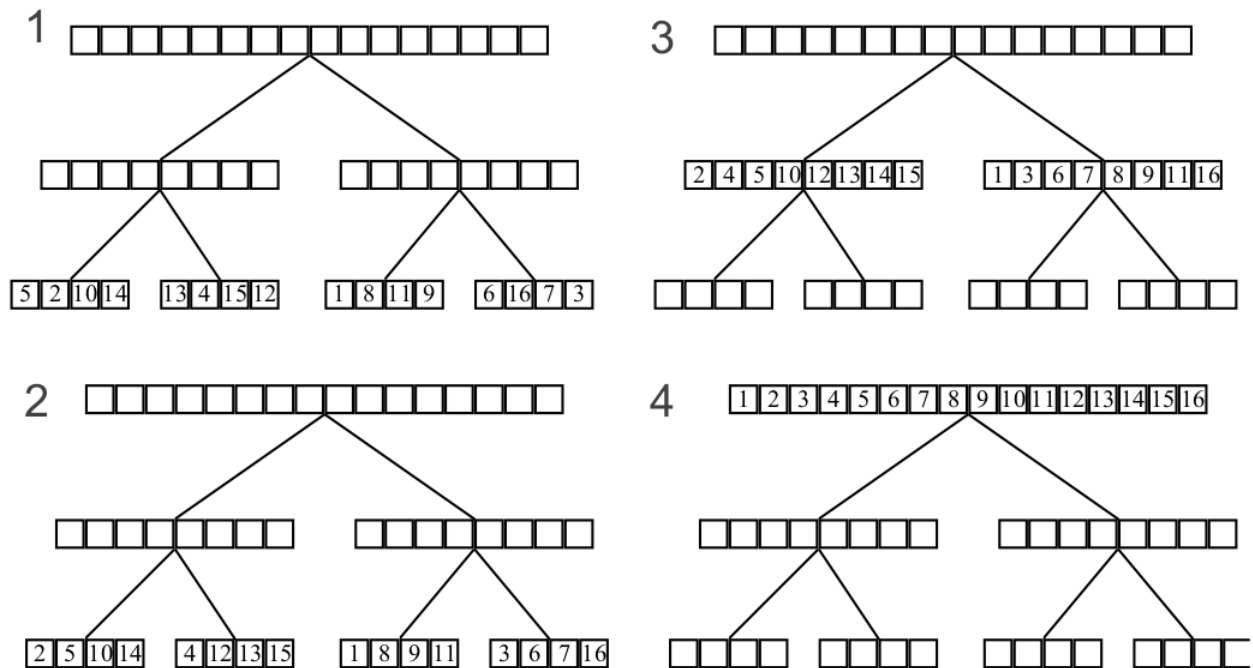
- Jelikož strom má $(\log n) + 1$ úrovní, první prvek se získá po $(\log n) + 1$ krocích. Kořenový procesor potřebuje jeden krok na porovnání a jeden na uložení výsledku do paměti. Každý ze zbylých $n - 1$ prvků spotřebuje 2 kroky.
- $t(n) = 2n + (\log n) - 1 = O(n)$

- $p(n) = 2n - 1$
- $c(n) = t(n) * p(n) = O(n^2)$
- což není optimální

Bucket Sort

- Stromová architektura
- Jeden procesor se stará o více než jeden prvek
- Fáze předpřípravy, seřazení optimálním sekvenčním algoritmem
 - viz merge sort

Příklad



Analýza

- $t(n) = O(n)$
- $p(n) = O(\log n)$
- $c(n) = O(n * \log n)$
- optimální

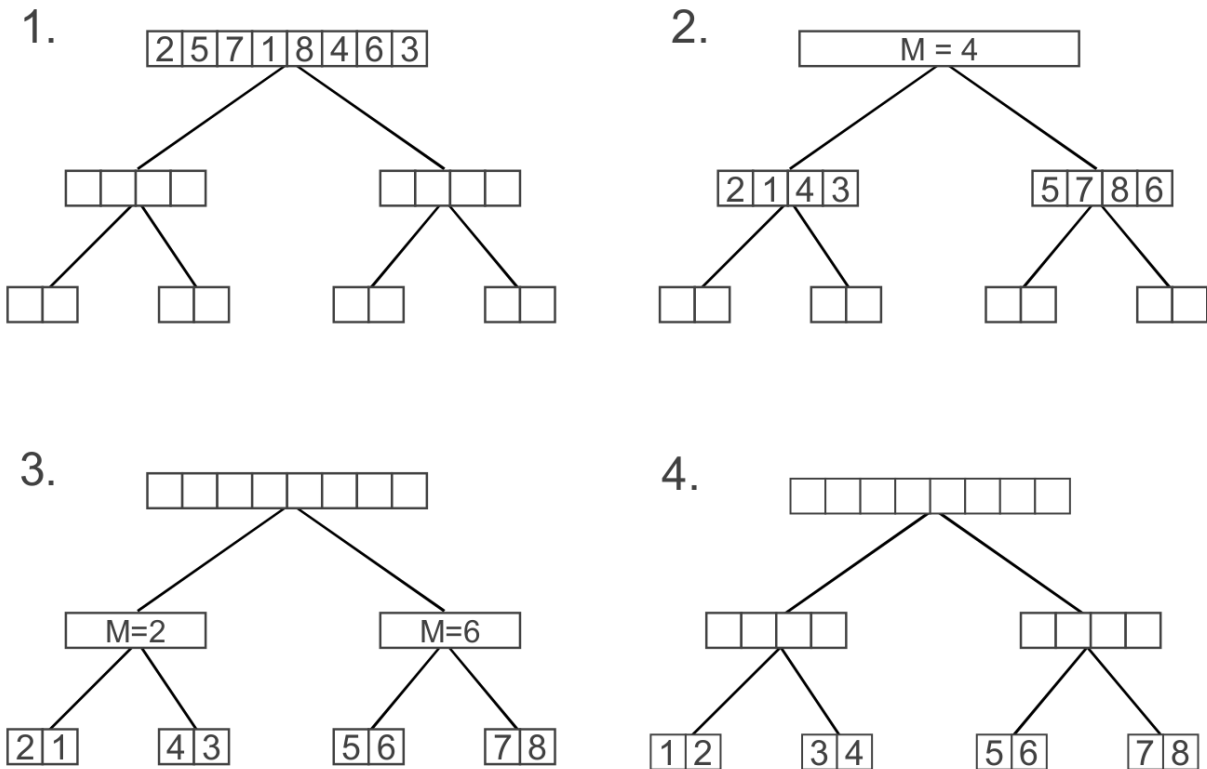
Median Finding and Splitting

- Stromová architektura
- Jeden procesor se stará o více než jeden prvek

- Jdeme opačným směrem, od kořene k listům

Algoritmus

- Uzel najde medián a všechny menší prvky pošle levému synovi a všechny větší pravému.
- Pokračuju do nějakého počtu, poté všichni seřadí zbytek optimálním sekvenčním algoritmem



Algoritmus nalezení mediánu není úplně triviální.

Analýza

- $t(n) = O(n)$
- $p(n) = O(\log n)$
- $c(n) = O(n \cdot \log n)$
- což je optimální

Shrnutí

Řazení na SIMD bez společné paměti

	$t(n)$	cena optimální?
<u>Speciální topologie</u>		
Enumeration Sort	$O(\log n)$	N
Odd-even Merge Sort	$\log^2 n$	N
<u>Lineární pole procesorů</u>		
Odd-Even Transposition	n	N
Merge-splitting Sort (agregovaná verze předchozího)		A
Pipeline Merge Sort	n	A
Enumeration Sort	n	N
<u>Mřížka (mesh)</u>		
Mesh Sort	$n^{1/2}$	N
Agregable Mesh Sort		A
<u>Strom</u>		
Minimum Extraction	n	N
Bucket Sort agregovaná verze předchozího		A
Median Finding and Splitting	n	A
<u>Hyperkostka</u>		
Cube Sort		
$t(n) = O(\log n) \dots O(\log^2 n)$		
$p(n) = n^2 \dots 2n$		

4) Algoritmy mediánu (select)

Sequential select

- Hledá k-tý nejmenší prvek v posloupnosti S
- Je-li $k = |S| / 2$, jde o medián

Algoritmus

```
procedure SEQUENTIAL_SELECT(S, k)
(1) if  $|S| \leq Q$  then seřaď S a odpočítej
    else rozděl S na  $|S|/Q$  posloupností  $S_i$  o délce Q prvků
(2) // Seřaď každou posloupnost  $S_i$  a najdi její medián  $M[i]$ 
    for  $i=1$  to  $|S|/Q$  do
         $M[i] = \text{SEQUENTIAL\_SELECT}(S_i, |S_i|/2)$ 
    end for
(3) // Najdi "medián mediánů" m
     $m = \text{SEQUENTIAL\_SELECT}(M, |M|/2)$ 
(4)  $L = \{s_i \in S: s_i < m\}$ 
     $E = \{s_i \in S: s_i = m\}$ 
     $G = \{s_i \in S: s_i > m\}$ 
(5) if  $|L| > k$  then SEQUENTIAL_SELECT(L, k) // prvek musí být v L
    else if  $|L| + |E| > k$  then return m // prvek musí být v E
    else SEQUENTIAL_SELECT(G,  $k - |L| - |E|$ ) // prvek musí být v G
```

- Pro $Q \geq 5$ $t(n) = O(n)$

Příklad

18 35 21 24 29 13 33 17 31 27 15 28 11 22 19 25 34 32 16 12 23 30 26 14 20

18 35 21 24 29

13 33 17 31 27

15 28 11 22 19

25 34 32 16 12

23 30 26 14 20

step 1

M 24 27 19 25 23

step 2

| \leftarrow s_1 \rightarrow $|s_2|$ \leftarrow s_3 \rightarrow |

18 21 13 17 15 11 22 19 16 12 23 14 20 24 35 29 33 31 27 28 25 34 32 30 26

step 4

18 21 13 17

15 11 22 19

16 12 23 14 20

step 1

17 15 16

| \leftarrow s_1 \rightarrow $|s_2|$ \leftarrow s_3 \rightarrow |

13 15 11 12 14 16 18 21 17 22 19 23 20

Parallel select

- Hledá k-tý nejmenší prvek v posloupnosti S
- EREW PRAM s N procesory $P_1 \dots P_N$
- Používá sdílené pole M o N prvcích

Algoritmus

```
procedure PARALLEL_SELECT(S, k)
(1) if |S| ≤ 4 then přímo nalezni k-tý prvek
    else rozděl S na N posloupností  $S_i$  o délce  $n/N$  a každou přiřaď
        jednomu procesoru  $P_i$ 
(2) for i=1 to N do in parallel
    M[i] = SEQUENTIAL_SELECT( $S_i$ , | $S_i$ |/2)
    end for
(3) m = PARALLEL_SELECT(M, |M|/2) ← s menším počtem procesorů
(4) L = {  $s_i \in S$ :  $s_i < m$  }
    E = {  $s_i \in S$ :  $s_i = m$  }
    G = {  $s_i \in S$ :  $s_i > m$  }
(5) if |L| > k then PARALLEL_SELECT(L, k)
    else if |L| + |E| > k then return m
    else PARALLEL_SELECT(G, k - |L| - |E|)
```

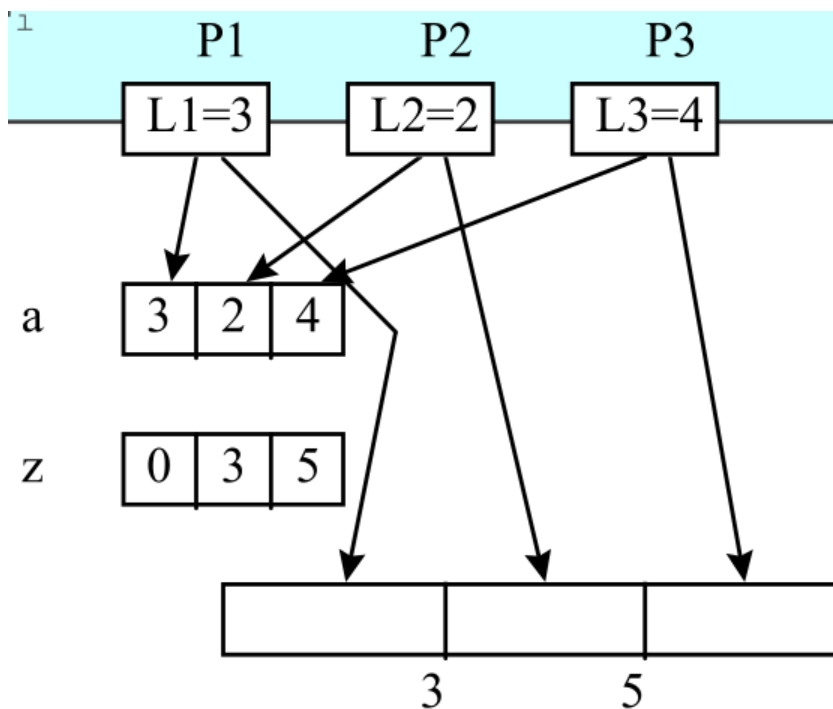
Analýza

- $t(n) = O(n/N)$ pro $n > 4$, $N < n/\log n$
- $p(n) = N$
- $c(n) = t(n) * p(n) = O(n)$
- což je optimální

Parallel splitting

- Krok 4 algoritmu Parallel select
- Úloha: Je dána posloupnost S a číslo \underline{m}
Mají se vytvořit tři posloupnosti:
 $L = \{s_i \in S: s_i < m\}$
 $E = \{s_i \in S: s_i = m\}$
 $G = \{s_i \in S: s_i > m\}$
- Složitost sekvenčního algoritmu je $O(n)$
- Paralelní řešení - máme N procesorů, které si sekvenci S rozdělí na podposloupnosti S_i o délce n/N

Příklad



Analýza

- $t(n) = O(\log N + n/N) = O(n/N)$ pro dostatečně malé N
- Cena: $c(n) = O(n/N) * N = O(n)$
- což je optimální

5) Algoritmy vyhledávání

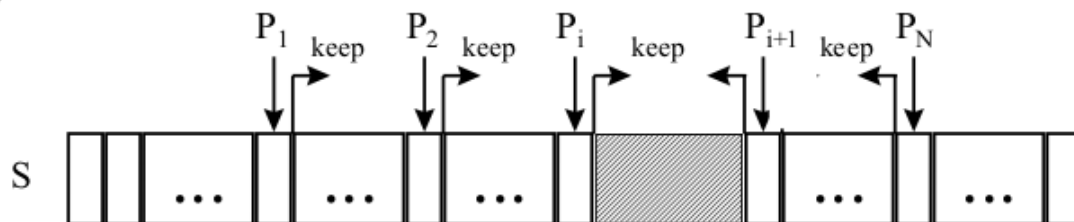
- Zjišťujeme zda prvek je v posloupnosti, případně na jakém indexu.
- Varianty
 - Posloupnost je seřazená
 - Posloupnost není seřazená

N-ární search

- Vyhledává se seřazené posloupnosti

Princip

- Při binárním vyhledávání zjistíme, ve které polovině se prvek nachází
- Při n-árním vyhledávání zjistíme, ve které z $N+1$ částí, se prvek nachází, kde N je počet procesorů



- Každý procesor se podívá na svoji "vlaječku" a na "vlaječku" svého pravého souseda
- Ten který detekuje změnu, zapíše svoji pozici a pozici svého pravého souseda – získáme blok je kterém je prvek
- Celý proces opakuju, dokud vyhledávací část není "akorát" malá, pak sekvenčně najdu

Analýza

- $t(n) = O(\log(n+1) / \log(N+1)) = O(\log_{N+1}(n+1))$
- $c(n) = O(N * \log_{N+1}(n+1))$
- což není optimální

Unsorted search

- Vyžaduje architekturu se sdílenou pamětí (PRAM)

- Vyhledává prvek v neseřazené posloupnosti

Princip

- Mám N procesorů, n prvků a hledaný prvek x
- Každý procesor si přečte hledaný prvek
- n prvků je rozděleno mezi N procesorů a provede se sekvenční search
- Do sdílené proměnné pak ten kdo našel zapíše index prvku, případně nedefinovaná hodnota

Analýza

Architektura EREW – Exclusive read exclusive write

- 1. krok = $O(\log n)$
- 2. krok = $O(n/N)$
- 3.krok = $O(\log N)$
- $t(n) = O(\log N + n/N)$
- $c(n) = O(N * \log N + n)$

Architektura CREW – Concurrent read exclusive write

- 1. krok = $O(1)$
- 2. krok = $O(n/N)$
- 3.krok = $O(\log N)$
- $t(n) = O(\log N + n/N)$
- $c(n) = O(N * \log N + n)$

Architektura CRCW – Concurrent read concurrent write

- 1. krok = $O(1)$
- 2. krok = $O(n/N)$
- 3.krok = $O(1)$
- $t(n) = O(n/N)$
- $c(n) = O(n)$ což je optimální

Tree search

- Vyhledává v neseřazené posloupnosti
- Stromová architektura s $2n-1$ procesorů

Algoritmus

1. Kořen načte hledanou hodnotu x a předá ji synům ... až se dostane ke všem listům
2. Listy obsahují seznam prvků, ve kterých se vyhledává (každý list jeden).
Všechny listy paralelně porovnávají x a x_i , výsledek je 0 nebo 1.
3. Hodnoty všech listů se předají kořenu každý ze list spočte logické or svých synů a výsledek zašle otcí. Kořen dostane 0 - nenalezeno, 1 - nalezeno

Analýza

- Krok (1) má složitost $O(\log n)$, krok (2) má konstantní složitost, krok (3) má $O(\log n)$.
- $t(n) = O(\log n)$
- $p(n) = 2 \cdot n - 1$
- $c(n) = t(n) \cdot p(n) = O(n \cdot \log n)$
 - což není optimální

