$f(n) = 1$ ⇒ sekvenční
$f(n) = c$ ⇒ konstantní
$f(n) = \log n$ ⇒ logaritmický
$f(n) = n$ ⇒ lineární

algoritmus s optimální cenou
$c(n)_{optim} = t_{seq}(n)$

55. Distribuované a paralelní algoritmy - algoritmy řazení, select, algoritmy vyhledávání.

V distribuovaných algoritmech kromě klasických metrik pro vyhodnocení kvality algoritmu (**časová složitost**, **paměťová složitost**) sledujeme i další metriky, konkrétně:

- **počet procesorů** potřebných k řešení úlohy v závislosti na velikosti instance n — $f(n)$
- **cena paralelního řešení** $c(n) = p(n) * t(n)$ (počet procesorů krát čas)
- **zrychlení**: čas sekvenčního algoritmu / čas paralelního algoritmu — $\frac{t_{seq}(n)}{t(n)}$
- **efektivnost**: čas sekvenčního algoritmu / cena paralelního algoritmu: $\frac{t_{seq}(n)}{c(n)}$
  - < 1 – neoptimální (způsobeno režií související s paralelizací)
  - = 1 – optimální

- čas - $t(n)$ - doba potřebná k řešení úlohy v počtu kroků

Algoritmy řazení

V algoritmech paralelního řazení typicky uvažujeme, že žádné dva prvky si nejsou rovny. Pokud bychom přesto měli takový soubor čísel, mohli bychom ke každému číslu přidat jeho pořadí a následně řadit podle dvojic (hodnota, pořadí), které už jsou jistě unikátní. Pro srovnání budeme využívat optimální sekvenční algoritmus (např. merge sort), jehož časová složitost je t(n) = O(n log n).

Enumeration sort

Pozice prvku v seřazené posloupnosti je dána počtem prvků, které jsou menší než tento prvek. Základním provedením je implementace na mřížce n x n, kde procesory v řádcích a sloupcích jsou propojeny do binárního stromu. Každý procesor se stará o porovnání dvojice prvků – tyto dva prvky má uloženy v registrech A a B. K tomu má ještě registr RANK, do kterého může přičítat. Princip algoritmu je následující:

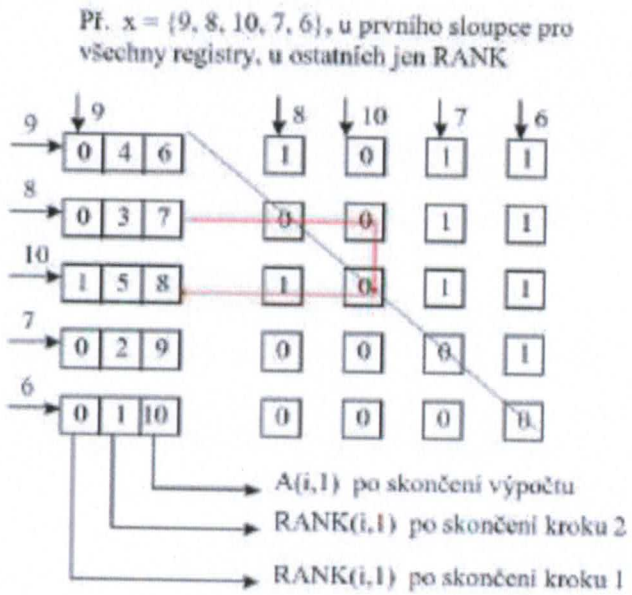
1. Každý prvek je porovnán se všemi ostatními pomocí jedné řady procesorů (pokud je prvek odpovídající danému řádku větší, nastavíme RANK na 1, jinak na 0)
2. RANK je pomocí stromové struktury sečten a v prvním sloupci v každém řádku je spočten celkový RANK, tzn. počet prvků, které jsou menší než prvek odpovídající danému řádku.
3. Prvky jsou přemístěny podle hodnoty RANK

$t(n) = O(\log n)$
$p(n) = n^2$
$c(n) = O(\log n \cdot n^2)$ – cena není optimální

```
1) for i=1 to n do in parallel
        1.1) každý procesor P(i, j) v řadě i získá x_i a x_j
                (i, j = 1...n) a uloží je do A(i, j) a B(i, j)

        1.2) if B(i, j) < A(i, j) then RANK(i, j) = 1
                                  else RANK(i, j) = 0
             endif
   endfor
2) for i = 1 to n do in parallel
        2.1) obsah registrů RANK všech procesorů v řadě i je sečten a
          uložen do RANK(i, 1)
        2.2) P(i, 1) spočte RANK(x_i) jako RANK(i, 1) += 1
   endfor
3) for i=1 to n do in parallel
        if  RANK(i, 1)=j then x_i je přesunuto z A(i, j) do A(j, 1)
        endif
   endfor
```

Složitost prvního kroku je O(log n) – šíření v řádích a sloupcích probíhá po binárním stromě. V kroku 2 probíhá sčítání na stromové struktuře, taktéž O(log n). Nakonec přesun hodnot podle RANK probíhá přes diagonálu, aby bylo zamezeno konfliktů. Vzhledem k posílání po stromové struktuře opět složitost O(log n). Celkem tedy složitost je O(log n) a cena O(n^2 log n), což není optimální. Znázornění kroku 3:

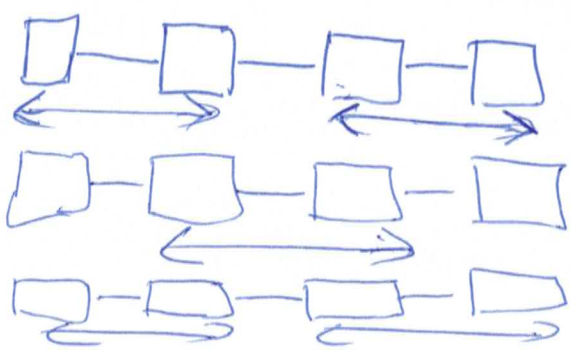


- žádný paralelní algoritmus pro rozumný výpočetní model není rychlejší, bez ohledu na počet procesorů

Odd-even transposition sort

Paralelní analogie bubble sort. Máme lineární pole n procesorů, každý obsahuje jednu z řazených hodnot. V prvním kroku se každý lichý procesor spojí se svým sousedem vpravo a porovnají své hodnoty. Je-li hodnota vlevo větší než hodnota vpravo, vymění si své hodnoty. V druhém kroku se stane to stejné, ale tentokrát se spojují sudé procesory. Po maximálně n krocích jsou hodnoty seřazeny. Časová složitost je tedy O(n), cena O(n^2), což není optimální.

na lineární topologii nelze dosáhnout lepší čas. složitosti

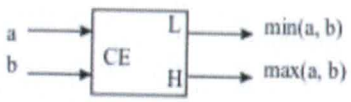


$p(n) = n$
$t(n) = O(n)$
$c(n) = O(n^2)$
- což není optimální

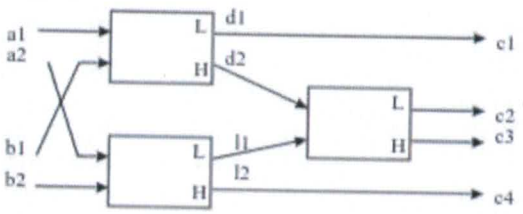
Odd-even merge sort

Řadí se sítí speciálních procesorů, každý má 2 vstupy a dva výstupy (Low, High). Procesor hodnoty na vstupu porovná a menší dá na výstup Low, větší na výstup High. Řazení pak probíhá kaskádou sítí, výstupem každé úrovně jsou pak seřazené pod-posloupnosti. Časová složitost je O(log^2 n), cena O(nlog^4 n), což není optimální.

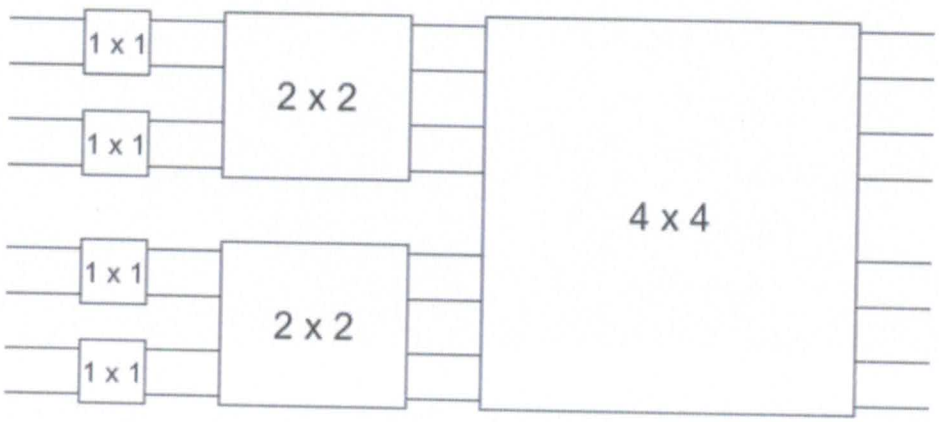
- Síť 1x1



2x2



$t(n) = O(\log^2 n)$
$c(n) = O(n \log^4 n)$
$p(n) = O(n \cdot \log^2 n)$



Merge-splitting sort

Opět lineární pole procesorů, funguje na podobném principu jako odd-even transposition sort, ovšem každý procesor uchovává několik čísel. Namísto jednoduchého porovnání a přehození pak provádíme operaci merge a následný split na seřazené podposloupnosti.

předzpracování (seřazení optim. sek. alg)

12,9,10 | 11,7,4 | 3,6,2 | 1,8,5
↓ | ↓ | ↓ | ↓
9,10,12 | 4,7,11 | 2,3,6 | 1,5,8

4,7,9 | 10,11,12 | 1,2,3 | 5,6,8

4,7,9 | 1,2,3 | 10,11,12 | 5,6,8

1,2,3 | 4,7,9 | 5,6,8 | 10,11,12

1,2,3 | 4,5,6 | 7,8,9 | 10,11,12

Pozor: Merge-splitting sort

```
Algoritmus
for i = 1 to p do in parallel
        procesor Pi seřadí svou posloupnost sekvenčním algoritmem
endfor
for k = 1 to ⌈p/2⌉ do
        1) for i = 1, 3, ..., 2.⌊p/2⌋ do in parallel
                spoj Si a Si+1 do setříděné sekvence Si'
                Si = první polovina Si'
                Si+1 = druhá polovina Si'
           endfor
        2) for = 2, 4, ... , 2.⌊p/2⌋ do in parallel
                ....
           endfor
endfor
```

- 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 $p \leq \log n$

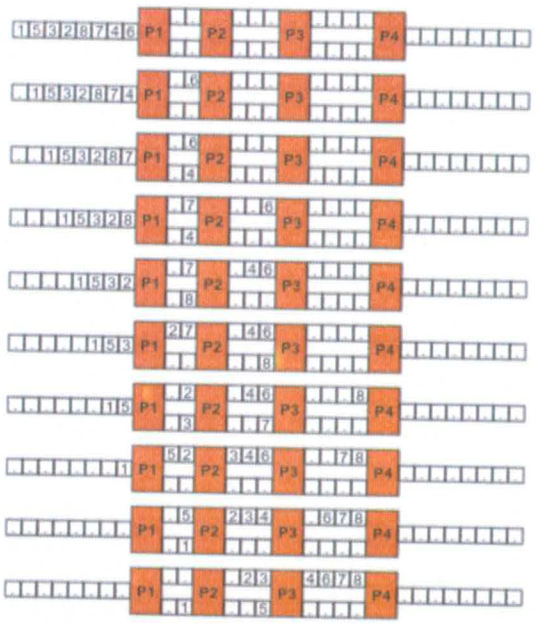
$c(n) = O(n \cdot \log n) + O(n \cdot p)$
je optimální pro $p \leq \log n$

Pipeline merge sort

Opět lineární pole procesorů, data tentokrát však nejsou uložena v procesorech, ale postupně do nich vstupují jako na zřetězené lince. Každý procesor spojuje dvě seřazené posloupnosti do delší seřazené posloupnosti (z počátku seřazené posloupnosti velikosti 1, pak velikosti 2, 4, ...). Časová složitost je O(n), přičemž využíváme log(n) procesorů, cena je tedy O(n log n), tedy algoritmus je optimální. (Algoritmus je navržen právě tak, aby používal právě počet procesorů nutný k dosažení optimální ceny.)

$A(n) = O(n)$
$p(n) = \log n$
$c(n) = O(n \cdot \log n)$, což je optimální

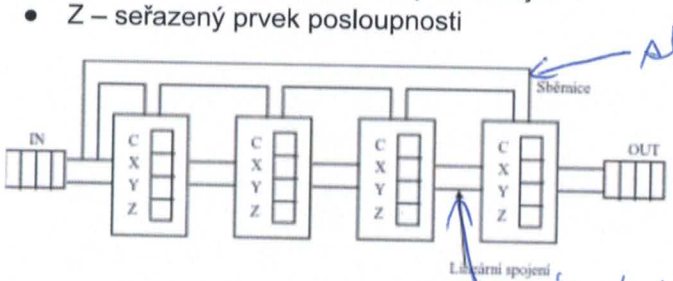
- střídá 1. a 2. výstup.
- procesor začne spojovat až nechválí, když má na 1. vstupu celou posloupnost (délky pro levého procesor) a na druhém vstupu alespoň jeden prvek



Enumeration sort na lineárním poli procesorů – lineární topologie

Máme n procesorů, které jsou propojeny společnou sběrnicí, která v každém kroku umožňuje přenést jednu hodnotu. Každý procesor má v sobě uloženo:

- X – jeden prvek z posloupnosti, fixní po celou dobu běhu
- Y – postupně všechny prvky z posloupnosti, probíhá posun doprava napříč procesory
- C – kolikrát bylo X větší než procházející Y
- Z – seřazený prvek posloupnosti



lineární propojení

Algoritmus funguje následovně:

1. Registr C se nastaví na 1
2. 2n-krát paralelně proběhne následující krok (iteraci označme k):
   a. Pokud vstup není vyčerpán, vloží se prvek i do registru X procesoru i (sběrnicí) a zároveň se vloží do prvního procesoru do registru Y a zároveň se obsah všech registrů Y ve všech registrech posune doprava
   b. Každý procesor s neprázdnými X a Y je porovná a inkrementuje C, pokud X > Y
   c. Je-li k > n (byl vyčerpán vstup), pak procesor (k-n) pošle sběrnicí obsah svého registru X procesoru (C - n), který jej uloží do svého registru Z. Tím umístíme jeden prvek na jeho finální místo.

$p(n) = n$

$t(n) = O(n)$

$c(n) = O(n^2)$, což není optimální

3. Nakonec posouvají procesory své registry Z doprava a tím je produkována výsledná posloupnosti

Časová složitost je O(n), cena je tedy O(n^2), což není optimální.

Minimum Extraction sort

Stromová hierarchie procesorů, každý listový procesor obsahuje jeden prvek. Každý nelistový procesor porovná hodnoty svých dvou synů a menší z nich pošle svému otci. Tímto způsobem minimální prvek bude v kořenovém procesoru po log n krocích a v každém dalším kroku se získá další nejmenší prvek. Časová složitost je tedy O(n), počet procesorů je lineární, tedy cena O(n^2).

$c(n) = O(n^2)$
$t(n) = O(n)$

$t(n) = O(\log n) + O(n)$

Bucket sort

Řazení stromem procesorů s m listovými procesory, přičemž n = 2^m. Každý listový procesor obsahuje n/m řazených prvků a umí je seřadit optimálním sekvenčním algoritmem (heap sort, merge sort). Každý nelistový procesor umí spojit dvě seřazené posloupnosti sekvenčním algoritmem. Časová složitost O(n), je využit logaritmický počet procesorů, celkém tedy cena O(n log n), tj. optimální.

$t(n) = O(\log n)$
$l(n) = O(n)$

$c(n) = O(n \log n) \Rightarrow$ optimální

Median Finding and Splitting

Stejná architektura jako bucket sort, tentokrát však nelistové procesory umí nalézt medián v optimálním čase (algoritmus select s lineární složitostí) – následně každý nelistový procesor najde medián a rozdělí prvky na prvky větší a menší než medián – menší dá levému procesoru a větší pravému procesoru. Jedná se tedy o paralelní verzi quicksortu. Časová složitost je O(n), cena je tedy O(n log n), je tedy také optimální.

Algoritmus Select

Algoritmus Select obecně hledá k-tý nejmenší prvek v posloupnpsti S. Speciálním případem je pak k = |S| / 2, tzn. medián. Algoritmus select funguje na principu rozděl a panuj (divide and conquer) – celý seznam je rozdělen na podseznamy malé délky Q (např. 5), ve kterých je rekurzivně nalezen medián (medián je v malém seznamu nalezen pomocí seřazení). Následně spočítáme medián mediánů, na základě kterého rozdělíme prvky na větší/rovno/menší a podle velikosti těchto seznamů rekurzivně hledáme medián už pouze v podmnožině prvků. Pro rozumné Q má algoritmus lineární časovou složitost.

```
Algoritmus
procedure SEQUENTIAL_SELECT(S, k)
(1) if |S| ≤ Q then seřaď S a odpočítej
       else rozděl S na |S|/Q posloupnosti S_i o délce Q prvků
(2) // Seřaď každou posloupnost S_i a nalezni její medián M[i]
    for i=1 to |S|/Q do
       M[i] = SEQUENTIAL_SELECT(S_i, |S_i|/2)
    end for
(3) // Nalezni "medián mediánů" m
    m = SEQUENTIAL_SELECT(M, |M|/2)
(4) L = {s_i ∈ S: s_i < m}
    E = {s_i ∈ S: s_i = m}
    G = { s_i ∈ 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
```

Paralelizace algoritmu select provedeme tak, že hledání mediánu v podposloupnostech provádíme paralelně:

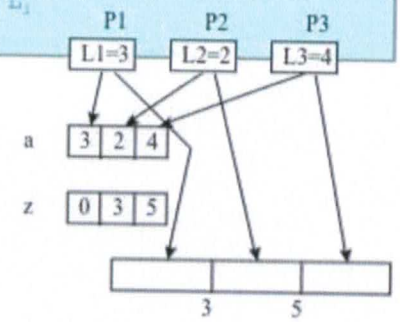
```
Algoritmus
procedure PARALLEL_SELECT(S, k)
(1) if |S| ≤ 4 then přímo nalezni k-tý prvek
    else rozděl S na N posloupnosti S_i o délce n/N a každou přiřaď
         jednomu procesoru P_i
(2) for i=1 to N do in parallel
       M[i] = SEQUENTIAL_SELECT(S_i, |S_i|/2)
    end for
(3) m = PARALLEL_SELECT(M,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

Časová složitost pro N procesorů, je rovna O(n/N), cena je tedy O(n), jedná se tedy o optimální paralelní algoritmus. Této časové složitosti dosáhneme díky implementaci kroku 4 pomocí algoritmu parallel splitting, který rozděluje posloupnost prvků na hodnoty větší/rovno/menší než zadaná hodnota (sekvenční algoritmus má tedy lineární časovou složitost). Algoritmus funguje následovně:

Algoritmus
(i) m se zašle všem procesorům procedurou BROADCAST
(ii) Každý procesor P_i rozdělí svoji sekvenci S_i na sekvence L_i, E_i, G_i
(iii) Všechny sekvence L_i jsou spojeny do sekvence L (a stejně tak E_i a G_i) následovně:
Nechť $a_i = |L_i|$
Pro všechny i, kde $1 \leq i \leq N$ se spočte suma:

$$z_i = \sum_{j=1}^{i} a_j \qquad \text{a } z_0 = 0$$

(iv) Nyní všechny procesory paralelně ukládají své posloupnosti L_i do L tak, že procesor P_i kopiruje L_i do L od pozice $z_{i-1}+1$.



V principu každý procesor spočítá dílčí množiny L, E, G a následně pomocí sumy prefixů (viz další otázka) určíme, na který index celkové posloupnosti má procesor začít nahrávat svoji dílčí posloupnost. Tímto se předchází kolizím při paralelním zápisu do paměti. Rozdělení se provádí sekvenčním algoritmem, pro N procesorů je tedy složitost O(n/N). Druhý krok *(na obrázku jako iii)* je suma prefixů, který má složitost O(log N), spojení subsekvencí má opět složitost O(n/N). Celkem tedy O(log N + n/N) = O(n/N) pro dostatečně malé N. Cena je tedy optimální.

Algoritmy vyhledávání

Při vyhledávání máme sekvenci prvků X = {x1, x2, …, xn} a hledaný prvek x. Máme za úkol zjistit, zda existuje k takové, že x = xk, a případně zjistit k. Rozlišujeme dva případy

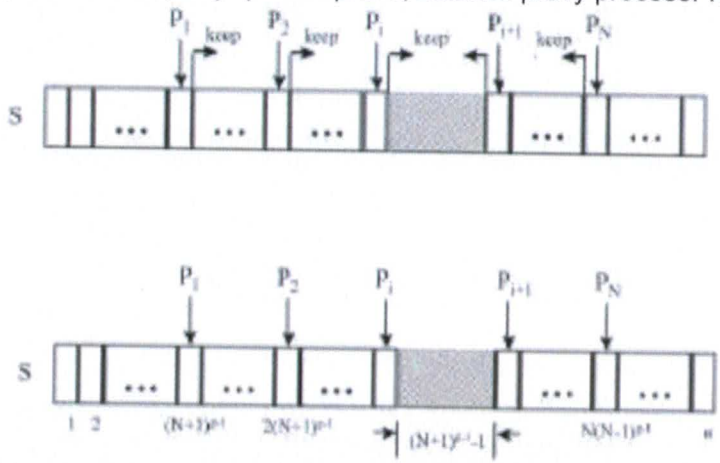
- posloupnost není seřazená. Optimální sekvenční algoritmus má lineární časovou složitost
- poslouponost je seřazená. Optimální sekvenční algoritmus má logaritmickou časovou složitost (binary search)

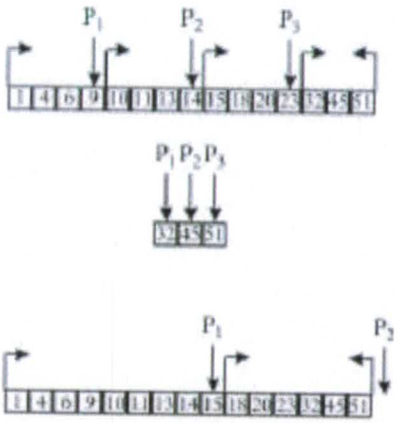
N-ary search

Paralelní rozšíření binárního vyhledávání (pracuje nad seřazenou sekvencí dat). U binárního vyhledávání zjistíme v každém kroku, ve které polovině se prvek nachází s pomocí jednoho procesoru. Pokud máme N procesorů, můžeme zjistit, ve které z N+1 částí se prvek nachází. Každý procesor dostane přidělenou podsekvenci a podívá se podle svého posledního prvku,

- jediný, z těch co probíráme, co umí využít, že máme seřazenou posloupnost

jestli je hledaný prvek vpravo nebo vlevo. Rekurzivně pak hledáme v podsekvenci, kde levý procesor říká, že je prvek vpravo, zatímco pravý procesor říká, že je prvek vlevo:





- Analýza
- Je třeba CREW PRAM
  - $t(n) = O(\log(n+1)/\log(N+1)) = O(\log_{N+1}(n+1))$
  - $c(n) = O(N.\log_{N+1}(n+1))$ →což není optimální

Unsorted search – předpokládáme PRAM

Vyhledávání v neseřazené posloupnosti s využitím N procesorů. Každý procesor hledá sekvenčním algoritmem v neseřazené podsekvenci:

```
Algoritmus
procedura SEARCH(S, x, k) { posloupnost, hledaný prvek, výsledek}
1. for i = 1 to N do in parallel
       read x
   endfor
2. for i = 1 to N do in parallel
       Si = (s_((i-1).(n/N))+1, s_((i-1).(n/N))+2, ..., s_(i.(n/N)))
       SEQUENTIAL_SEARCH (S_i, x, k_i)
         - paralelní Search volá sekvenční SEQUENTAL Search
   endfor
3. for i = 1 to N do in parallel
       if k_i > 0 then k = k_i endif
   endfor
```

Problémem je hlavně první krok (načtení dat) a třetí krok (spojení dílčích výsledků jednotlivých procesorů) – jejich složitost záleží na architektuře, na které algoritmus implementujeme. Druhý krok má nezávisle na architektuře složitost O(n/N). První a třetí krok:

- EREW – 1. krok načtení s využitím stromu O(log N), třetí krok taktéž, cena je tedy O(NlogN + n), což není optimální
- CREW – 1. krok konstantní díky sdílenému čtení, 3. krok zůstává logaritmický
- CRCW – umožňuje i sdílený zápis, tedy sjednocení výsledků v kroce 3 proběhne v konstantním čase, celková složitost je tedy O(n/N), cena je tedy O(n) -> optimální.

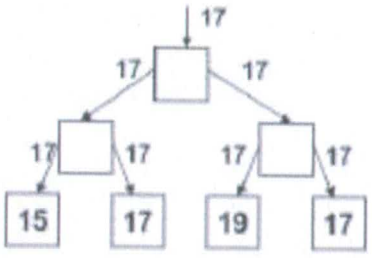
Tree search

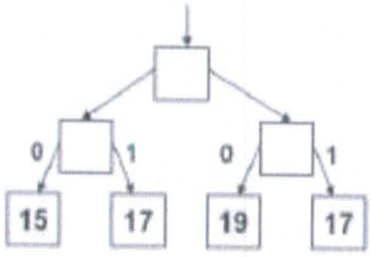
Každý listový uzel má jeden prvek seznamu. Kořen načte hledanou hodnotu x a předá ji synům, až se takto dostane hledaná hodnota k listům. Každý list pak nastaví výsledek na 1, pokud jeho prvek je roven x, jinak na 0. Následně se po úrovních ve stromě provádí v nelistových uzlech operace OR nad syny, až se do kořene dostane výsledná hodnota – 1 v případě nalezení a 0 v případě nenalezení. Složitost je vzhledem ke stromovým operacím O(log n), což vzhledem k lineárnímu počtu procesorů dává neoptimální cenu O(n log n). Příklad hledání čísla 17 v posloupnosti {15, 17, 19, 17}:

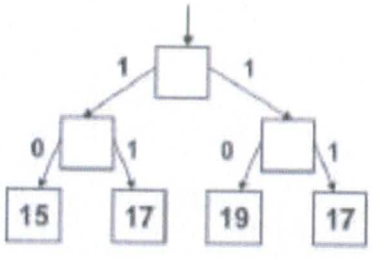
$t(n) = O(\log n)$

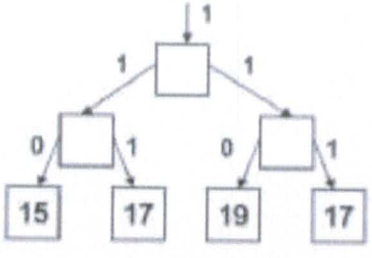
$p(n) = O(n)$

$c(n) = O(n \cdot \log n)$, což není optimální









Pokud v textu najdete chybu, nebudete něčemu rozumět nebo budete mít dojem, že by bylo vhodné něco doplnit, kontaktujte na discordu uživatele Fifinas.