Vyhledávání

Upd 2007, Cor 2007/8

5. VYHLEDÁVÁNÍ

- Máme sekvenci X = {x_1, x_2, ..., x_n} a prvek <u>x</u>
- Máme za úkol zjistit, zda x = x_k a případně zjistit <u>k</u>
- Optimální sekvenční algoritmus

  a) X není seřazená - sekvenční vyhledávání

  t(n)=O(n)   c(n)=O(n)

  (je třeba prozkoumat <u>n</u> prvků)

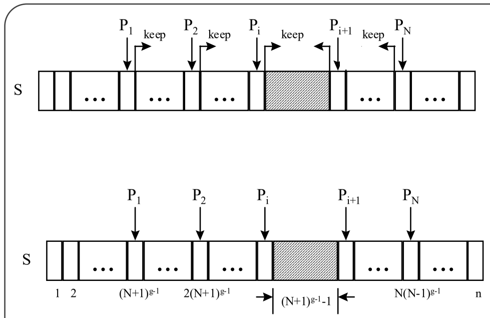
  b) X je seřazená - binární vyhledávání

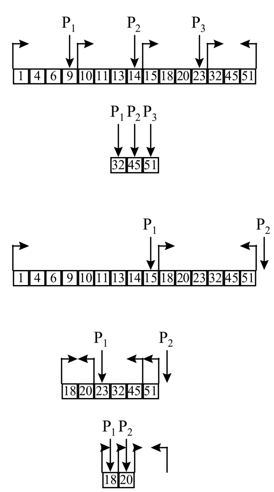
  t(n)=O(log n)   c(n)=O(log n)

  (pro rozlišení mezi <u>n</u> prvky je třeba prozkoumat log n prvků)

5.1 N-ary Search

- Vyhledává v seřazené posloupnosti
- Princip:
  - Při binárním vyhledávání zjistíme v každém kroku ve které polovině se prvek nachází za pomocí jednoho procesoru.
  - S použitím N procesorů lze provést N+1 ární vyhledávání - v jednom kroku zjistíme, ve které z N+1 části se může prvek nacházet
- Počet kroků je g = ⌈ log(n+1)/log(N+1) ⌉





- <u>Analýza</u>
- 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

5.2 Unsorted Search

- vyhledává prvek v neseřazené posloupnosti
- model je PRAM s N procesory

```
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, ..., si.(n/N)}
        SEQUENTIAL_SEARCH (Si, x, ki)
          - paralelní Search volá sekvenční SEQUENTAL Search
   endfor
3. for i = 1 to N do in parallel
        if ki > 0 then k = ki endif
   endfor
```


Analýza

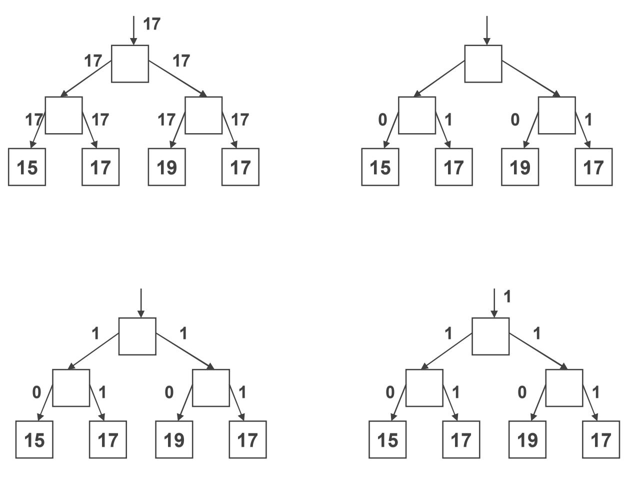
- **EREW**
  - 1. krok = O(log n) 2. krok = O(n/N) 3.krok = O(log N)
  - t(n) = O(log N + n/N) c(n) = O(N.log N + n)
- **CREW**
  - 1. krok = O(1) 2. krok = O(n/N) 3.krok = O(log N)
  - t(n) = O(log N + n/N) c(n) = O(N.log N + n)
- **CRCW**
  - 1. krok = O(1) 2. krok = O(n/N) 3.krok = O(1)
  - t(n) = O(n/N) c(n) = O(n) → což je optimální

5.3 Tree Search

- Vyhledávání v neseřazené posloupnosti
- Stromová architektura s 2n-1 procesory

- 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 ne list spočte logické or svých synů a výsledek zašle otci. 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.n-1
  - c(n) = t(n).p(n) = O(n.log n) → což není optimální

Tree Search

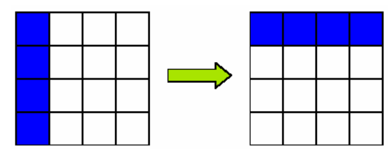


Maticové algoritmy

6. TRANSPOZICE

- Matici n x n s prvky a_{ij} převést na matici s prvky a_{ji}
- Sekvenční řešení:

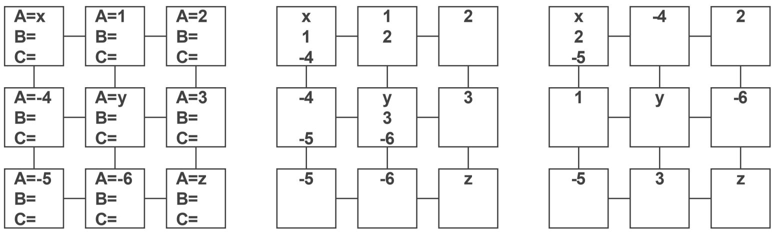
```
procedure TRANSPOSE(A)
  for i=2 to n do
    for j=1 to i-1 do
      aij ⇔ aji
    endfor
  endfor
```

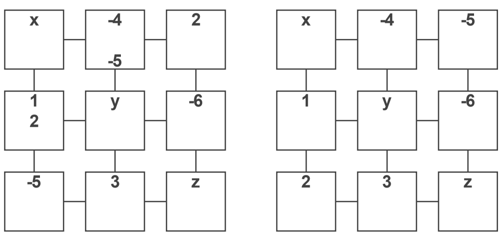


- Složitost je $O(n^2)$.
- **n je zde počet řádků/sloupců matice**

6.1 Mesh transpose

- Mřížka n x n procesorů
- Každý procesor má 3 registry
  - A - obsahuje a_{ij}, a_{ji} po ukončení
  - B - hodnoty od pravého (horního) souseda
  - C - hodnoty od levého (dolního) souseda

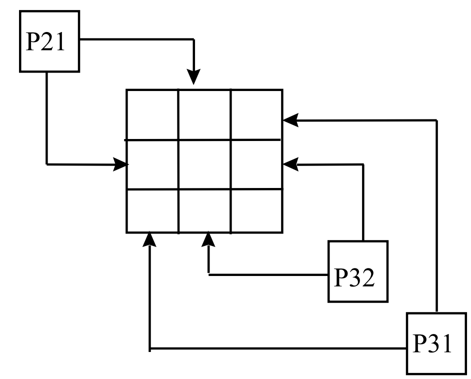




- <u>Analýza</u>
  - Nejdelší cesta prvku je 2(n-1) kroků, tj.
  - $t(n) = O(n)$
  - $p(n) = n^2$ a cena $c(n) = O(n^3)$ → není optimální

EREW transpose

•




```
procedure EREW TRANSPOSE(A)
for i=2 to n do in parallel
  for j=1 to n-1 do in parallel
       a_ij ⇔ a_ji
  endfor
endfor
```

- <u>Analýza</u>
  - $t(n) = O(1)$
  - $p(n) = (n^2-n)/2 = O(n^2)$
  - $c(n) = O(n^2)$ → což je optimální

7. NÁSOBENÍ MATIC

- Součin matic A (m x n) a B (n x k) je matice C (m x k) kde:

$$c_{ij} = \sum_{s=1}^{n} a_{is} * b_{sj} \qquad 1<=i<=m \quad 1<=j<=k$$

Složitost je $O(n^3)$

Složitost optimálního algoritmu není známa, je $O(n^x)$, kde $2<x<3$

Žádný algoritmus nemá lepší složitost než $O(n^2)$

```
procedure MATRIX MULT(A, B, C)
for i=1 to m do
  for j=1 to k do
    c_ij = 0
    for s=1 to n do
         c_ij=c_ij + (a_is * b_sj)
    endfor
  endfor
endfor
```


Násobení matic – sdílená paměť

- Nerealistická varianta

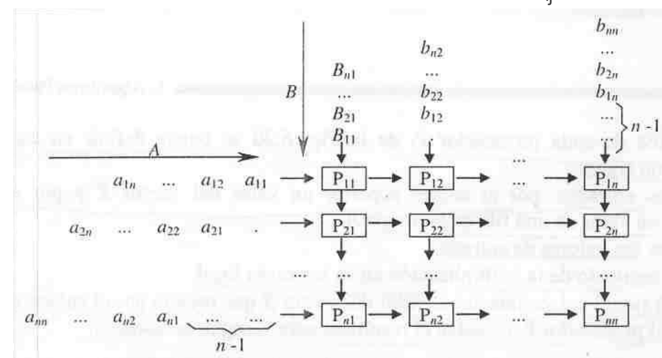
```
procedure MATRIX MULT(A, B, C)
for i=1 to m do in parallel
  for j=1 to k do  in parallel
    c_ij = 0
    for s=1 to n do  in parallel
         c_ij=c_ij + (a_is * b_sj)
    endfor
  endfor
endfor
```

- Realistická varianta

```
procedure MATRIX MULT(A, B, C)
for i=1 to m do in parallel
  for j=1 to k do  in parallel
    c_ij = 0
    for s=1 to n do
         c_ij=c_ij + (a_is * b_sj)
    endfor
  endfor
endfor
```

7.1 Mesh multiplication

- Mřížka n x k procesorů
- Prvky matic A a B se přivádějí do procesorů 1. řádku a 1 sloupce
- Každý procesor P(i,j) obsahuje prvek c_{ij}




```
Algoritmus
procedure MESH MULT(A, B, C)
for i=1 to m do in parallel
  for j=1 to k do in parallel
    cij = 0
    while P(i,j) receives inputs a,b do
         cij = cij + (a * b)
         if i<m then send b to P(i+1, j)
         if j<k then send a to P(i, j+1)
    endwhile
  endfor
endfor
```

- Analýza
- Prvky a_{m1} a b_{1k} potřebují m+k+n-2 kroků, aby se dostaly k poslednímu procesoru P(m, k)
- $t(n) = O(n)$ $p(n)=O(n^2)$
- $c(n) = O(n^3)$ $\rightarrow$ což není optimální

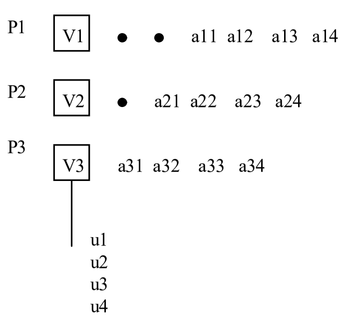
7.1.1 Násobení matice vektorem

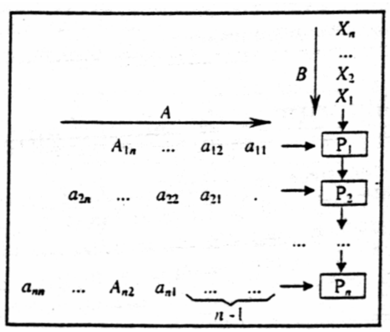
- Součin matice A (m x n) a vektoru U (n) je vektor V (m) kde:

$$v_i = \sum_{j=1}^{n} a_{ij} * u_j \qquad 1<=i<=m$$

7.1.2 Linear array multiplication

- Lineární pole m procesorů, každý obsahuje jeden prvek v_i





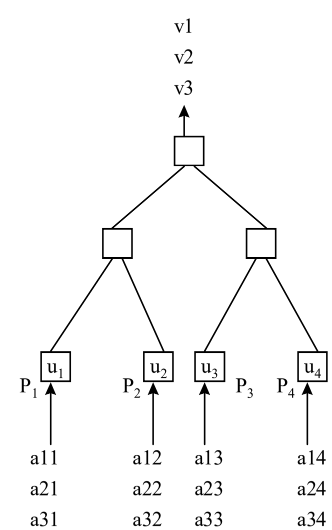
- Algoritmus

```
procedure LINEAR MV MULT
for i=1 to m do in parallel
  vi = 0
  while Pi receives inputs a and u do
    vi = vi + (a * u)
    if i>1 then send u to Pi-1
  endwhile
endfor
```

- Analýza
  - $t(n) = m+n-1 = t(n)=O(n)$ $p(n)=O(n)$
  - $c(n)=O(n^2)$ $\rightarrow$ což je optimální

7.2 Tree MV multiplication

- Čas m+n-1 předchozího algoritmu je možno zlepšit na m-1+log n při zdvojnásobení počtu procesorů
- Architektura má n listových procesorů $P_1 \ldots P_n$ a n-1 nelistových procesorů
- Listové procesory násobí, nelistové sčítají




```
Algoritmus
procedure TREE MV MULT(A, U V)
do steps 1 and 2 in parallel
(1) for i=1 to n do in parallel
        for j=1 to m do
                send ui*dij to parent
        endfor
    endfor
(2) for i=n+1 to 2n-1 do in parallel
        while Pi receives two inputs do
                compute the sum
                if i<2n-1       then send result to parent
                                else write result
                endif
        endwhile
    endfor
```

- Analýza
  - $t(n)=m-1+ \log n = O(n)$
  - $c(n)=O(n^2)$ $\rightarrow$ což je optimální

KONEC