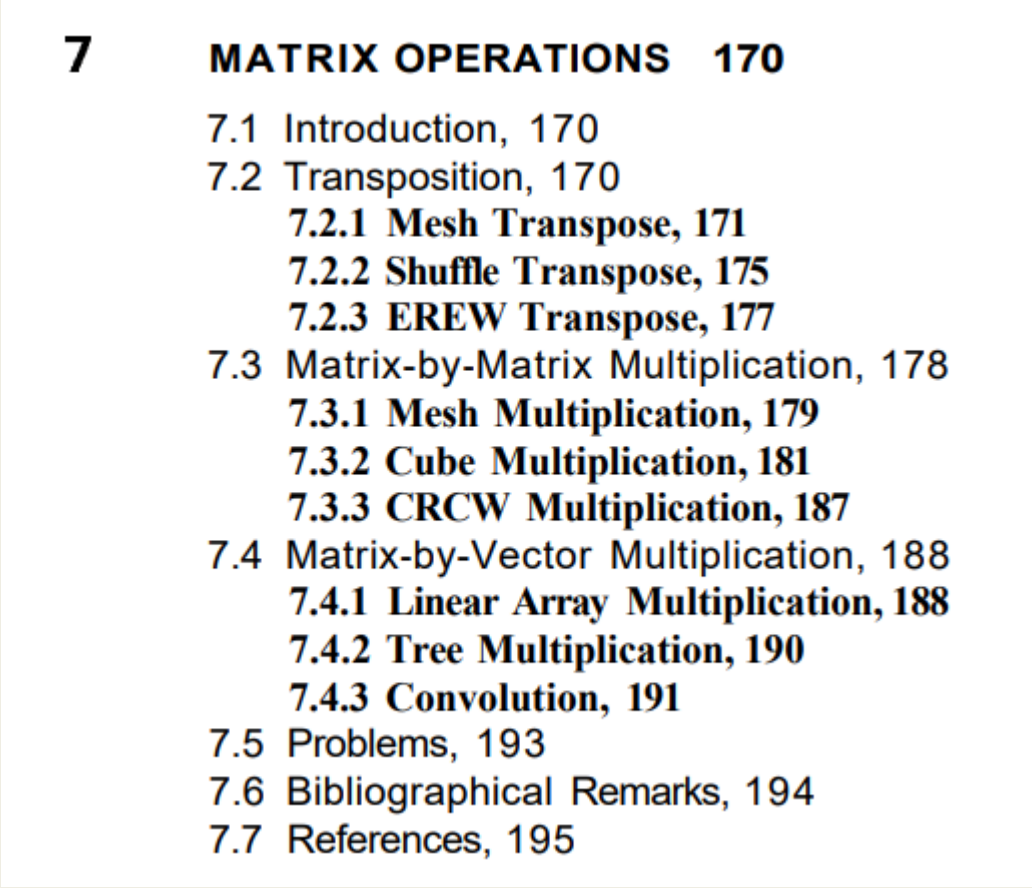
ALGORITMY PRO MATICE A VEKTORY

Literatura k této přednášce

- Akl, kapitola 7
- https://research.iaun.ac.ir/pd/saeed-nasri/pdfs/UploadFile_9325.pdf



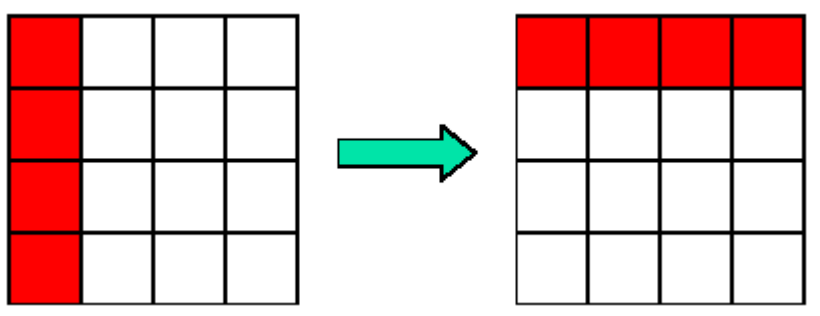
7 **MATRIX OPERATIONS 170**

Transpozice matice

- Čtvercová matice n x n s prvky $a_{i\,j}$
- Čtvercová matice n x n s prvky $a_{j\,i}$
- Sekvenční řešení:

```
procedure TRANSPOSE(A)
  for i=2 to n do
    for j=1 to i-1 do
      a_ij ? a_ji
    endfor
  endfor
```

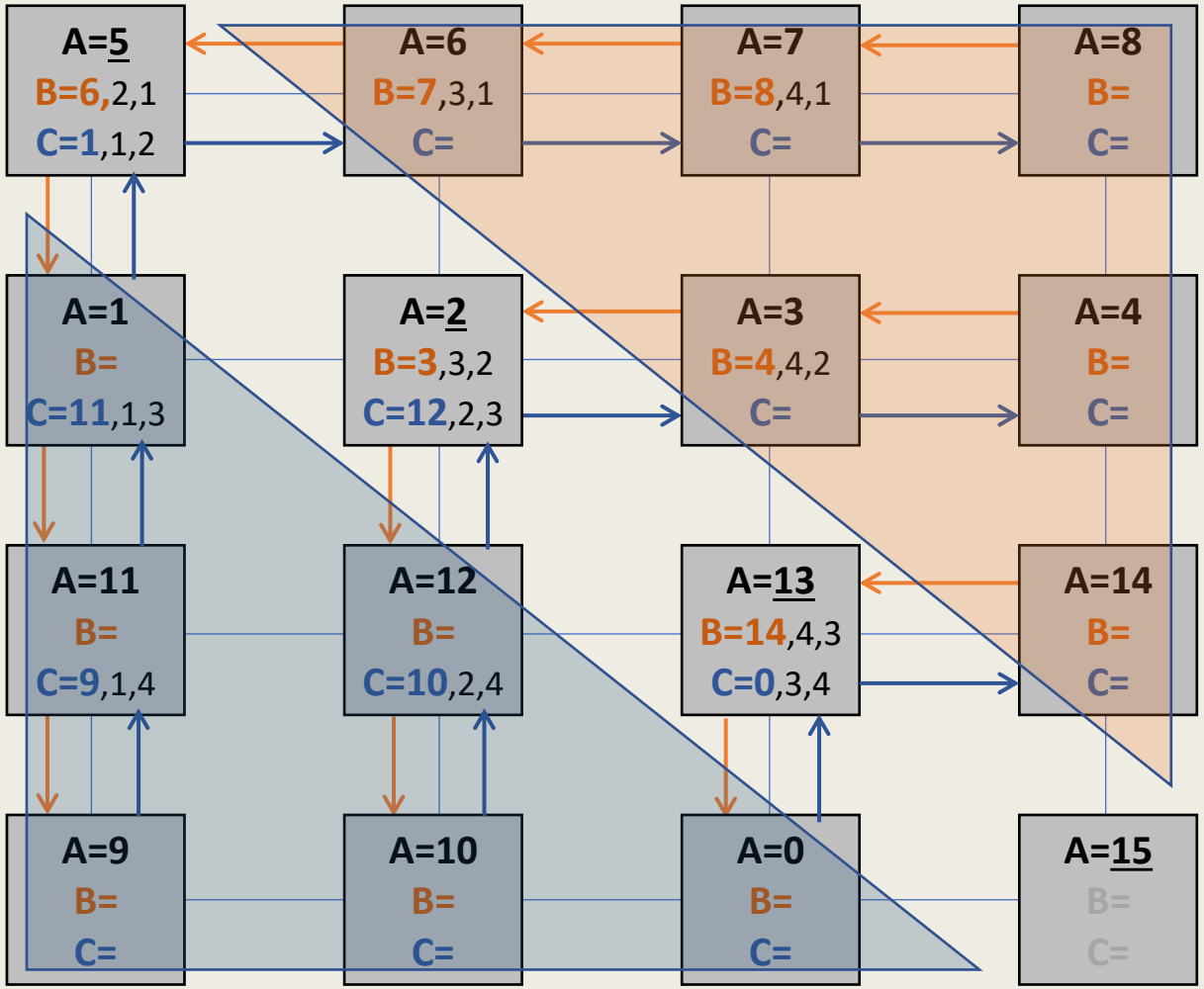
- Složitost je $t(n) = O(n^2)$



Transpozice matice, mřížková topologie

- Topologie, mřížková n x n pro čtvercovou matici n x n
- 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*

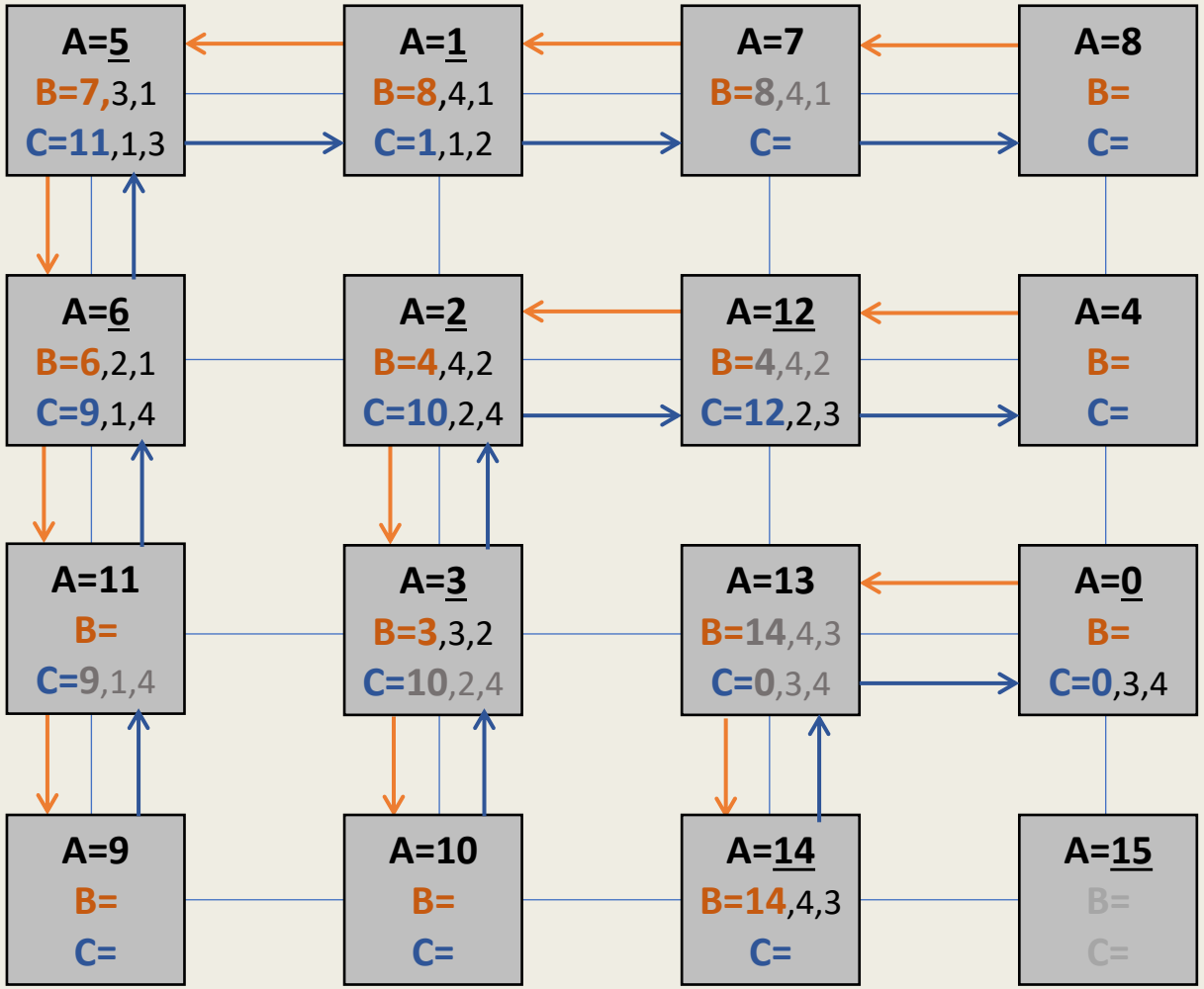
Transpozice matice, mřížková topologie




```
Step 1:
do steps 1.1 and 1.2 in parallel
(1.1)
 for i = 2 to n do in parallel
  for j = 1 to i - 1 do in parallel
   C(i - 1, j) ← (a_ij, j, i)
end for
end for
(1.2)
 for i = 1 to n - 1 do in parallel
  for j = i + 1 to n do in parallel
   B(i, j - 1) ← (a_ij, j, i)
  end for
 end for.
```

(hodnota, cilovy_x, cilovy_y)

Transpozice matice, mřížková topologie



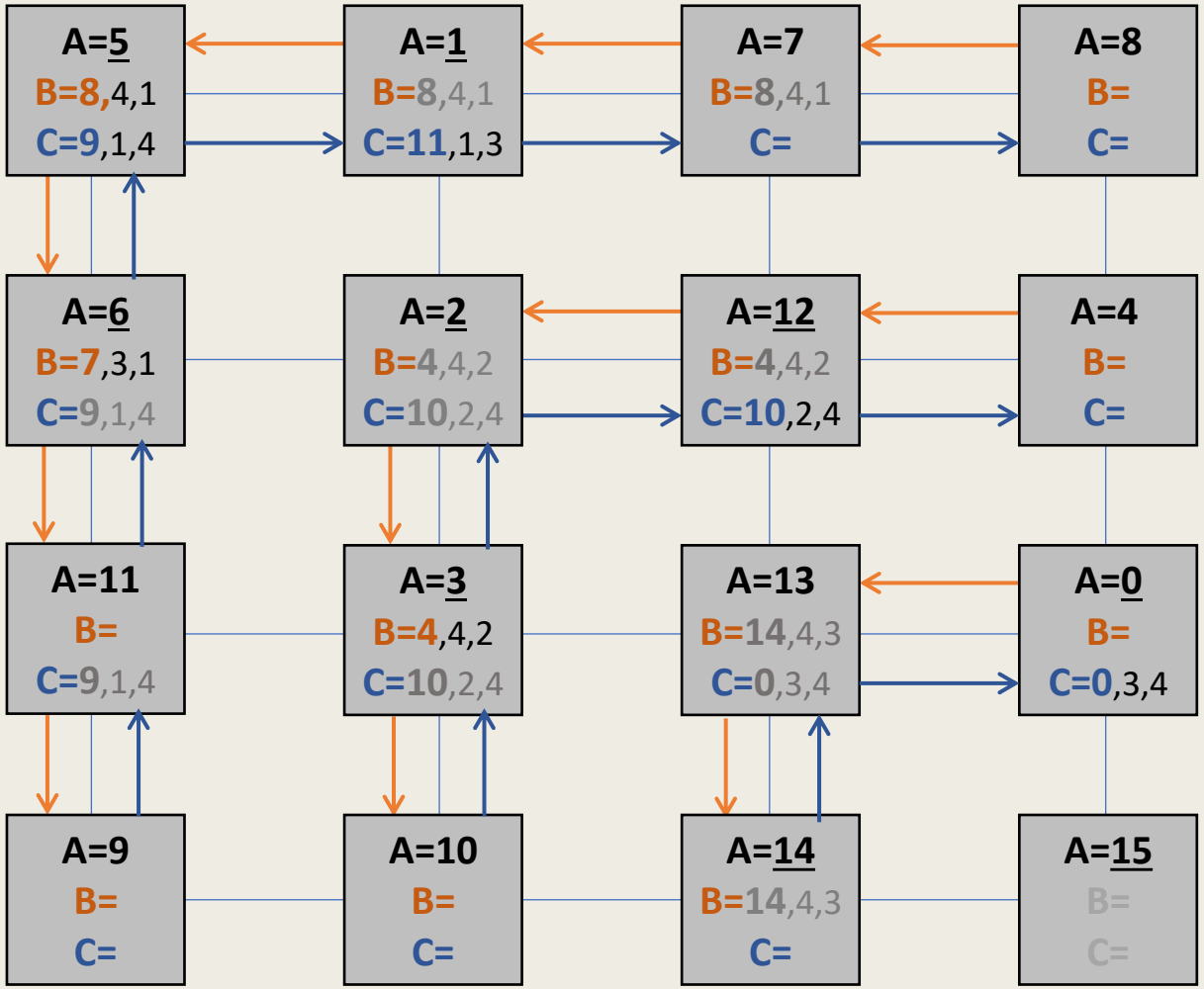
Paralelně 2.1, 2.2, 2.3

2.1
Pokud obdrží hodnoty zprava, posílají dál
Pokud obdrží hodnotu zleva, a jsou cílové, uloží, jinak posílají dál

2.2
Pokud diagonální obdrží zdola, posílají vpravo
Pokud diagonální obdrží zprava, posílají dolů

2.3
Pokud obdrží hodnoty zdola, posílají dál
Pokud obdrží hodnotu shora, a jsou cílové, uloží, jinak posílají dál

Transpozice matice, mřížková topologie



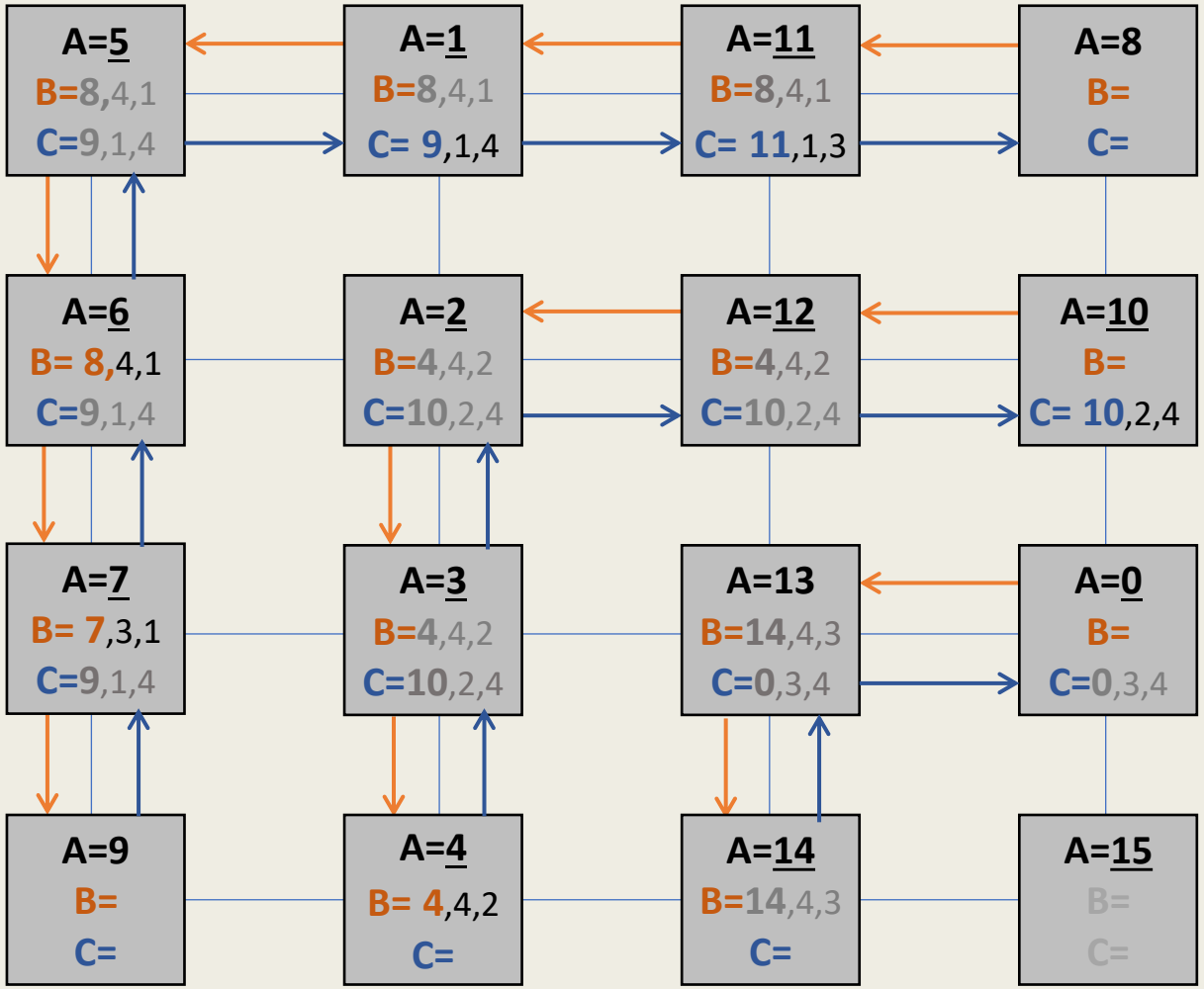
Paralelně 2.1, 2.2, 2.3

2.1
Pokud obdrží hodnoty zprava, posílají dál
Pokud obdrží hodnotu zleva, a jsou cílové, uloží, jinak posílají dál

2.2
Pokud diagonální obdrží zdola, posílají vpravo
Pokud diagonální obdrží zprava, posílají dolů

2.3
Pokud obdrží hodnoty zdola, posílají dál
Pokud obdrží hodnotu shora, a jsou cílové, uloží, jinak posílají dál

Transpozice matice, mřížková topologii



+ další dva kroky TODO

Paralelně 2.1, 2.2, 2.3

2.1
Pokud obdrží hodnoty zprava, posílají dál
Pokud obdrží hodnotu zleva, a jsou cílové, uloží, jinak posílají dál

2.2
Pokud diagonální obdrží zdola, posílají vpravo
Pokud diagonální obdrží zprava, posílají dolů

2.3
Pokud obdrží hodnoty zdola, posílají dál
Pokud obdrží hodnotu shora, a jsou cílové, uloží, jinak posílají dál

Transpozice matice, mřížková topologie, analýza

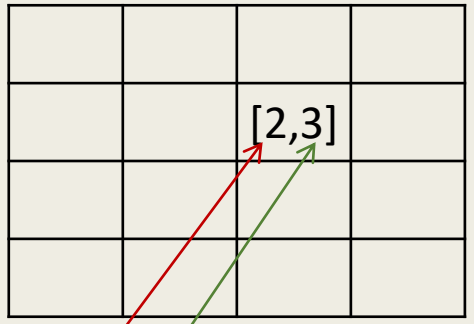
- Počet procesorů $p(n) = O(n^2)$
- Časová složitost, k přesunutí je třeba $2 * (n - 1)$ kroků $\mathrm{t(n)} = O(n)$
- Cena $\mathrm{c(n)} = O(n^3)$ což **není optimální**

Transpozice matice, Perfect Shuffle

- **Topologie**
  - *Uzly propojené jako **Perfect Sfuffle** (dokonalý přesun)*
  - *Funguje pro matice $n \times n$ kde $n = 2^q$ pro nějaké přirozené číslo q*
  - *Celkem tedy $2^{q^2} = 2^{2q}$ uzlů*
  - *Pro indexování uzlů je třeba $2q$ bitů*
- **Uzel**
  - *Může přeposlat hodnotu na uzel s indexem levě bitově posunutým*
    - (např. 0101 -> 1010, 1100 -> 1001 apod.)
- Princip
  - *Po $\log n$ tedy q přesunech každé z hodnot matice je matice transponovaná*

Transpozice matice, Perfect Shuffle

Uzel matice $[i, j]$ je indexován jako $2^q(i-1)+(j-1)$



$$2^q(i-1)+(j-1) = 2^2 * 1 + 2$$
$$= 6 = \mathbf{0110}$$

```
procedure SHUFFLE TRANSPOSE (A)
for i = 1 to q do
      for k = 1 to 2^2q − 2 do in parallel
            P_k, sends the element of A it currently holds to P_2kmod(2^2q−1)
      end for
end for
```

Levý shift

Transpozice matice, Perfect shuffle, příklad

Matice $4x4$ neboli 2^2x2^2, q=2

$$2^q(i-1)+(j-1) \rightarrow 2^q(j-1)+(i-1)$$

Pro $[1,2] \rightarrow [2,1]$, tedy

$$2^2(1-1)+(2-1) \rightarrow 2^2(2-1)+(1-1)$$

$$1 \rightarrow 4, (0001 \rightarrow 0010)$$

Pro $[2,4] \rightarrow [4,2]$, tedy

$$2^2(2-1)+(4-1) \rightarrow 2^2(4-1)+(2-1)$$

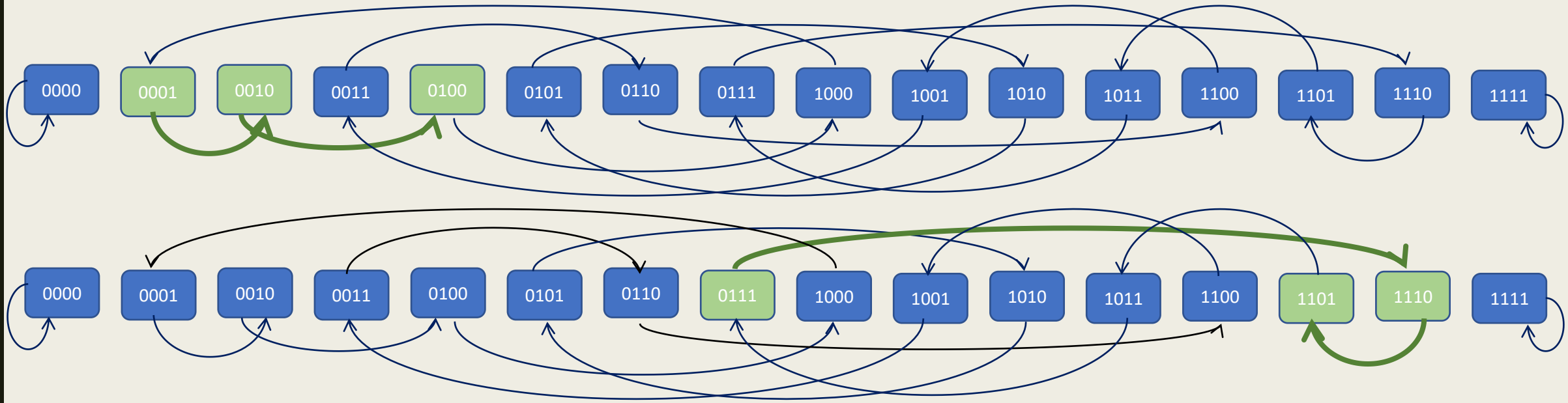
$$7 \rightarrow 13, (0111 \rightarrow 1101)$$

	(1,2)		
(2,1)			

0001 → 0100

		(2,4)	
	(4,2)		

0111 → 1101



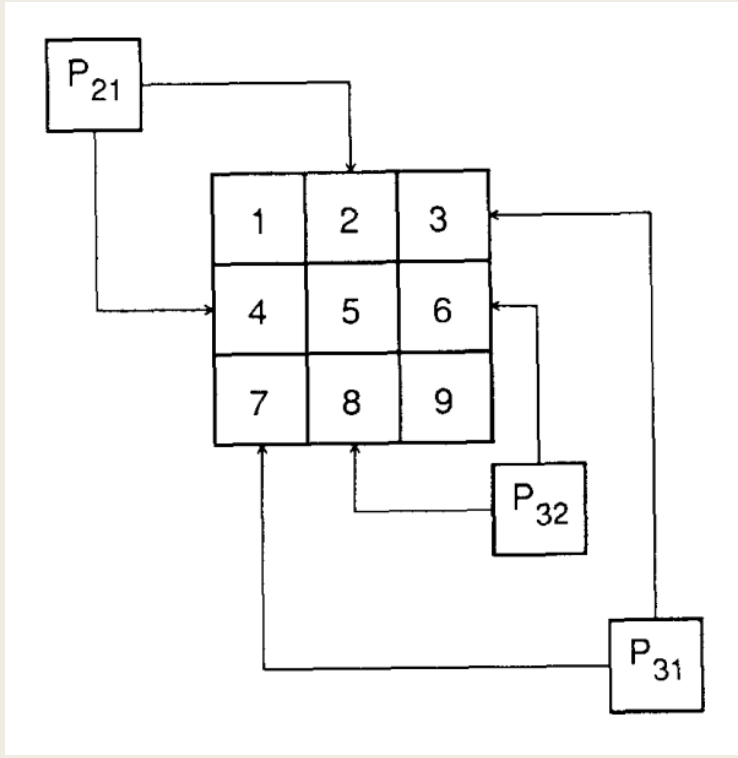
Transpozice matice, Perfect shuffle

- Při přenosu nedochází ke konfliktům -> každý odesilatel je unikátní a tedy i příjemce po levém shiftu je unikátní
- Počet procesorů $p(n) = O(n^2)$
- Časová složitost, k přesunutí je třeba $\log \mathrm{n}$ kroků $\mathrm{t}(\mathrm{n}) = O(\log n)$
- Cena $\mathrm{c}(\mathrm{n}) = O(\log n \ * n^3)$ což **není optimální**

Transpozice matice, sdílená paměť

- I pro EREW Ize provést transpozici v konstantním čase
- Potom cena je $c = O(n^2)$

```
procedure EREW TRANSPOSE
for i = 2 to n do in parallel
        for j = 1 to i - 1 do in parallel
                aij ++ aji
        end for
end for
```



Násobení matic

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

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

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

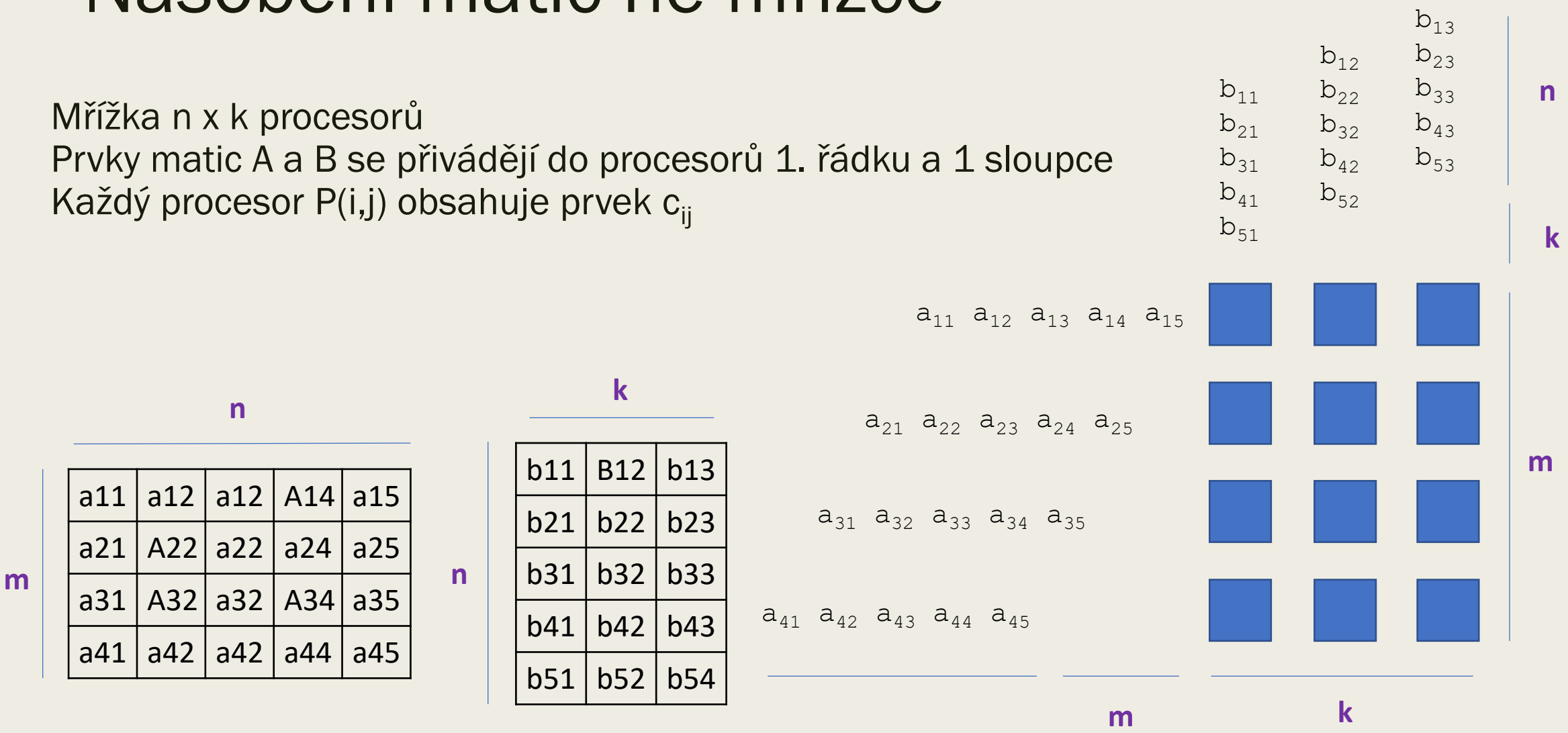
```
procedure MATRIX MULT(A, B, C)
for i=1 to m do in parallel
  for j=1 to k do  in parallel
    c_ij = 0
    for s=1 to n do  in parallel
      c_ij=c_ij + (a_is * b_sj)
    endfor
  endfor
endfor
```

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
```

Násobení matic ne mřížce

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}



Násobení matic ne mřížce

```
Algoritmus
procedure MESH MULT(A, B, C)
for i=1 to m do in parallel
  for j=1 to k do in parallel
    c_ij = 0
    while P(i,j) receives inputs a,b do
        c_ij = c_ij + (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)

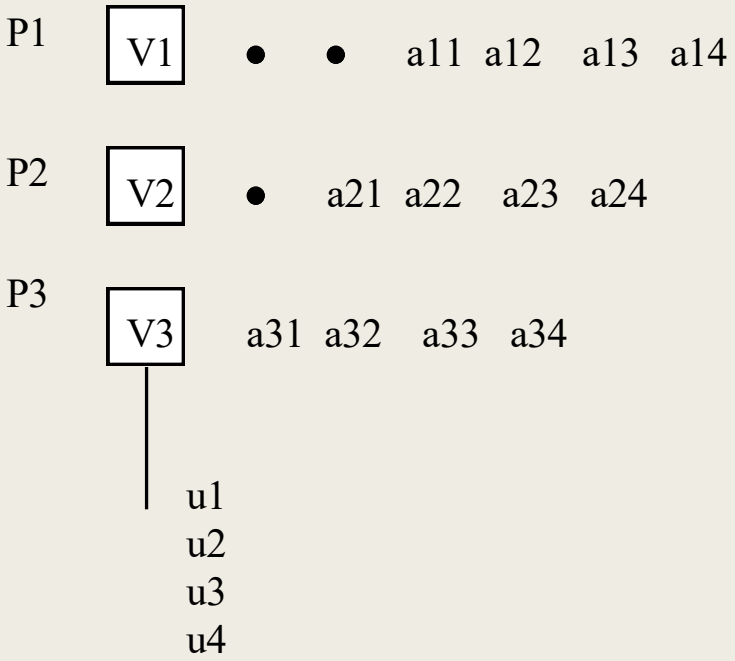
t(n) = **O**(n) **p**(n) = **O**(n^2) **c**(n) = **O**(n^3) … což **není optimální**

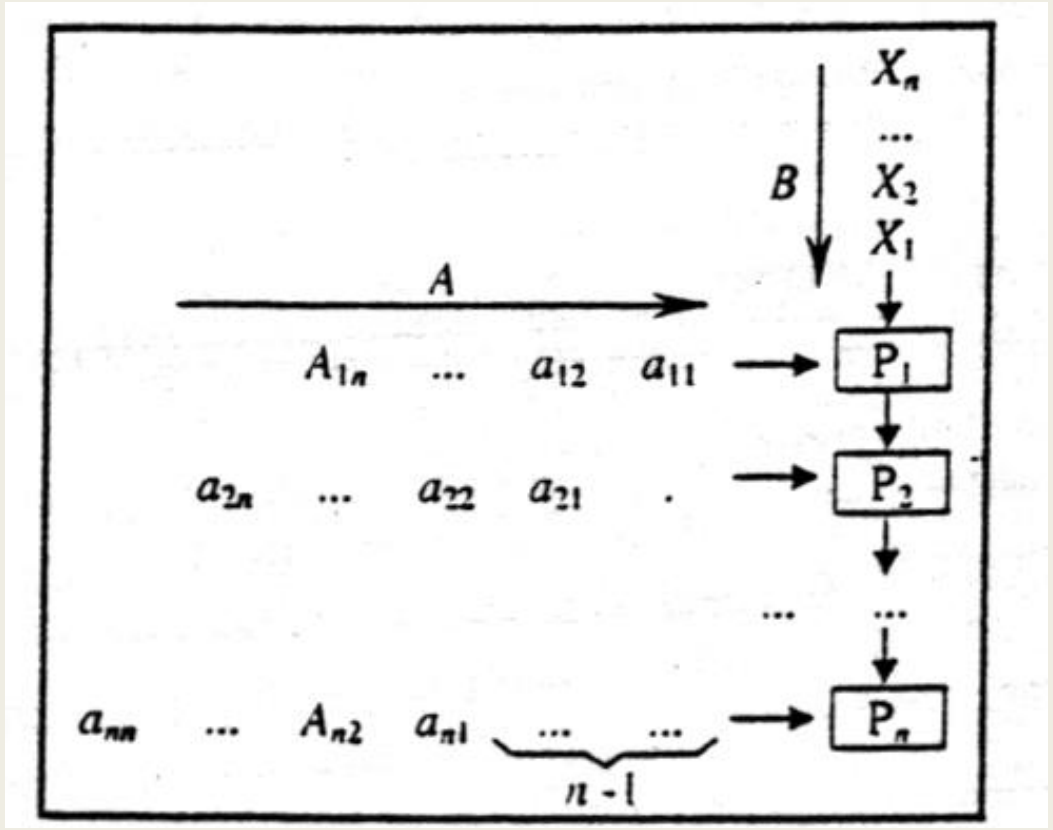
Násobení matice vektorem

- Součin matice A (m x n) a vektoru U (n) je vektor V (m) kde:
- $v_i = \sum_{l=1}^{n} a_{ji} * u_i \; 1 \leq i \leq n$
- Časová složitost optimálního sekvenčního algoritmu je $\boldsymbol{t}(n) = \boldsymbol{O}(m * n)$
  - *Je třeba pracovat s každým prvkem matice*

Paralelní násobení matice vektorem

- Lineární pole m procesorů, každý obsahuje jeden prvek v_i





Paralelní násobení matice vektorem, algoritmus a analýza

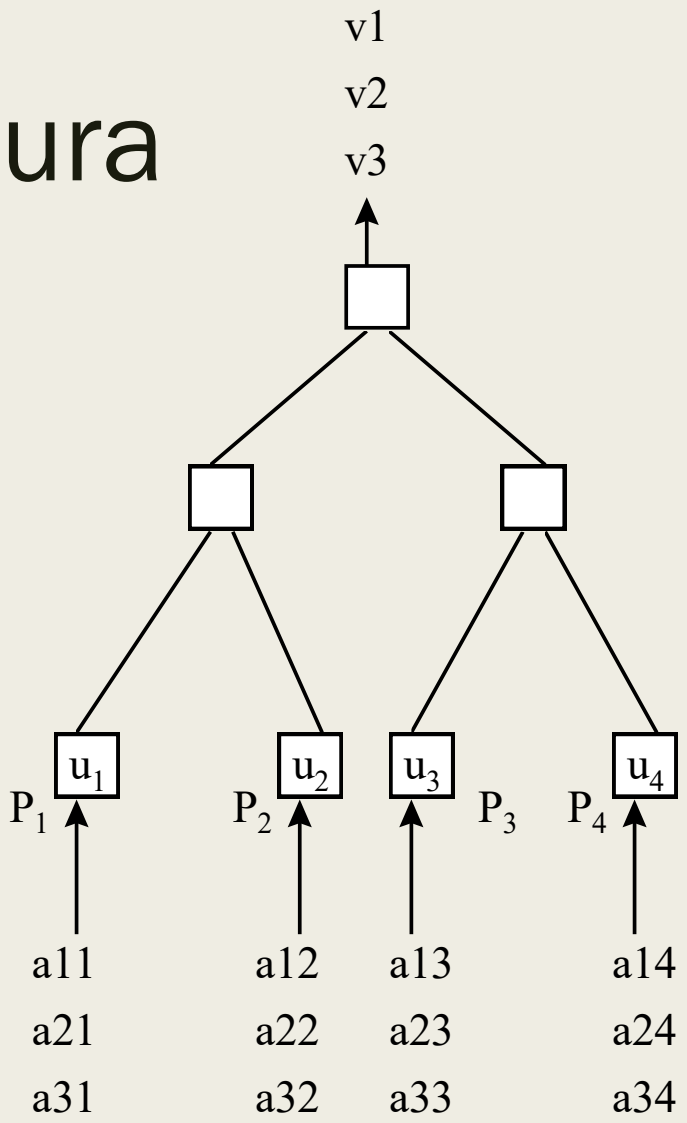
```
procedure LINEAR MV MULT
for i=1 to m do in parallel
  vi = 0
  while Pi receives inputs a and u do
    v_i = v_i + (a * u)
    if i>1 then send u to P_i-1
  endwhile
endfor
```

Analýza

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

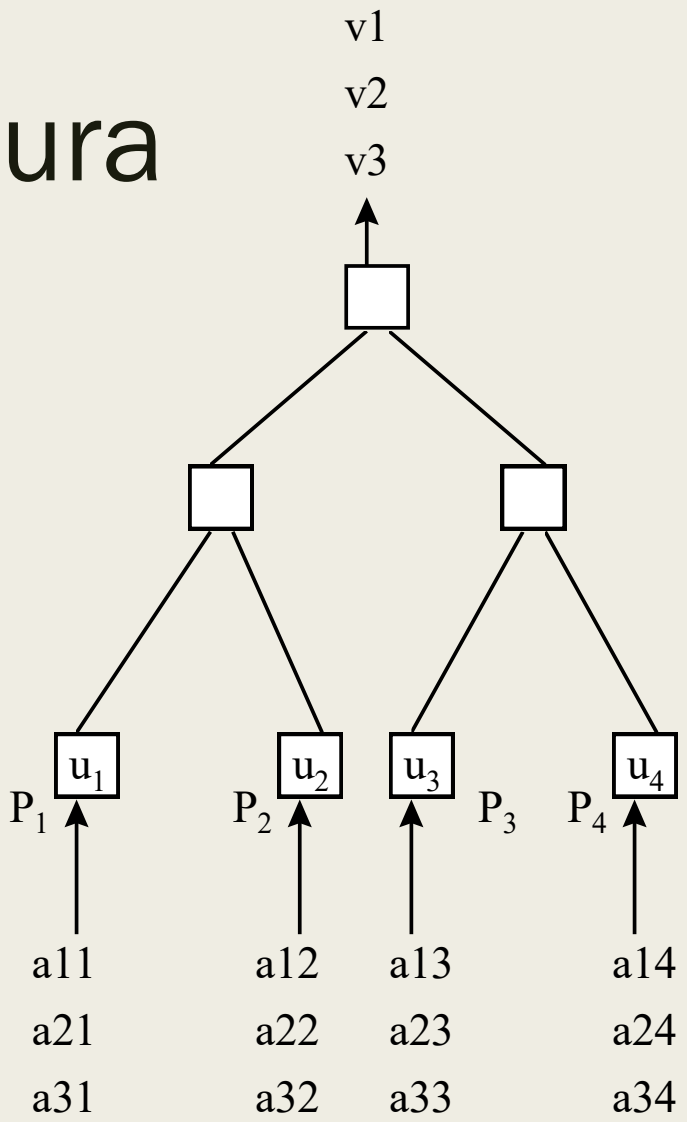
Paralelní násobení matice vektorem, stromová struktura

- 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...P_n a n-1 nelistových procesorů
- Listové procesory násobí, nelistové sčítají



Paralelní násobení matice vektorem, stromová struktura

```
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 u_i*d_ij to parent
      endfor
   endfor
(2) for i=n+1 to 2n-1 do in parallel
      while P_i 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(m)*
  - *c(n)=O(m*n)* → *což je optimální*