

Přednáška #7: Paralelní prefixový součet a technika eulerovské cesty

Paralelní redukce

- Vstupní pole X n hodnot x_0, x_1, \dots, x_{n-1} z množiny D .

- Vypočti

$$S = x_0 \oplus x_1 \oplus \dots \oplus x_{n-1},$$

kde $\oplus =$ **binární** operace nad množinou D .

- Redukce je krásně paralelizovatelná $\iff \oplus =$ **asociativní**.

Vlastnosti paralelní redukce

- $T(n, p) = \alpha n/p + \beta \log p$.

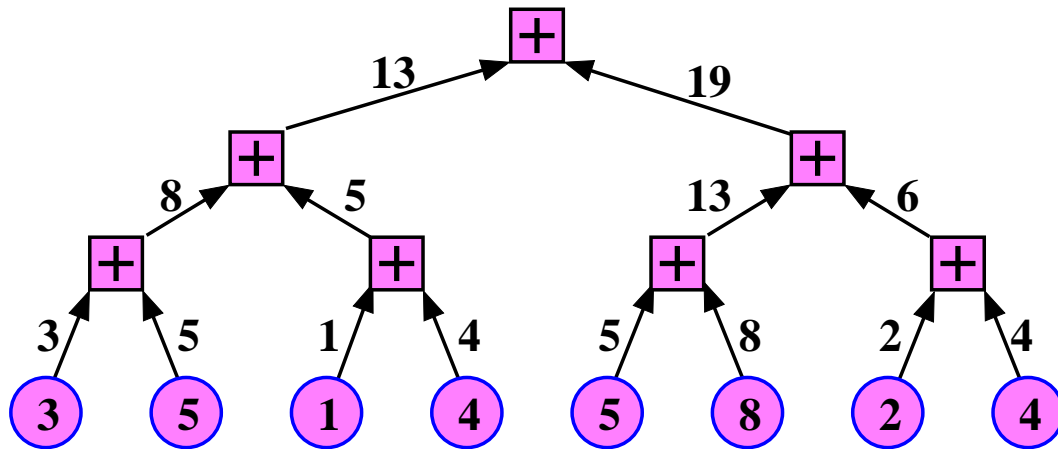
- $C(n, p) = \alpha n + \beta p \log p$.

- $E(n, p) = \alpha n / (\alpha n + \beta p \log p)$.

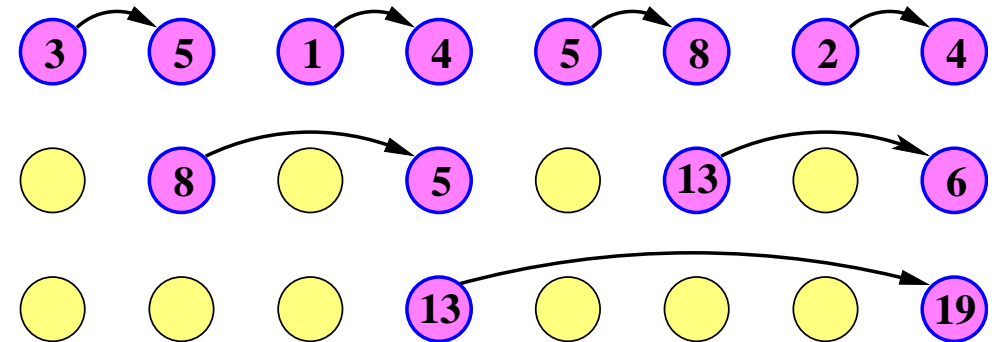
- Dobrá škálovatelnost: $\psi_1(p) = p \log p$ a $\psi_2(n) = n / \log n$.

- Časová optimalita: $T_{\min}(n, p) = O(\log n)$ a $\psi_3(n) = n / \log n$.

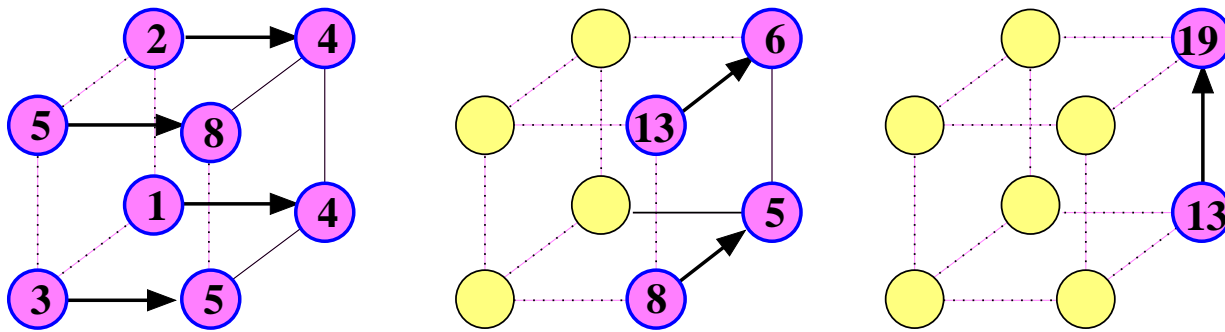
- Je to **normální hyperkubický** algoritmus \implies **optimální** na hyperkubických sítích.



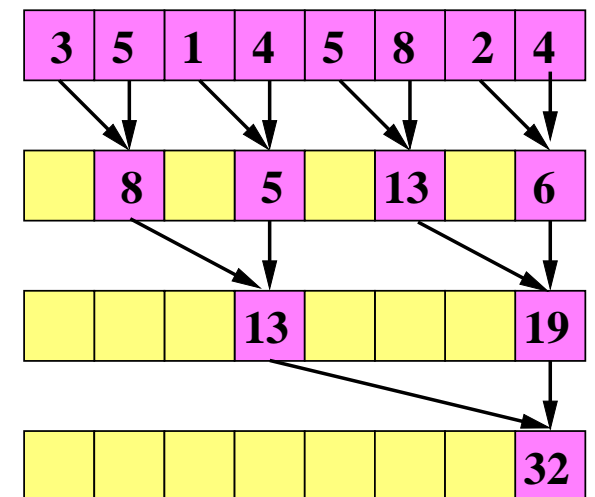
(a) (Neprimy) úplný binární strom



(c) WH 1-D mřížka

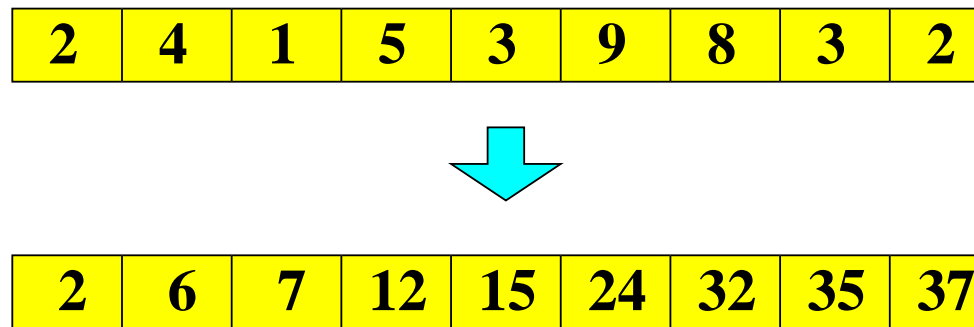


(b) Úplný graf a SF hyperkrychle



(d) EREW PRAM

- Vstupní pole $X = x_0, \dots, x_{n-1}$ z množiny D .
- **A**sociativní binární operátor \oplus v D .
- Úkolem je vypočítat pole Y všech **prefixů** pole X : $y_i = x_0 \oplus x_1 \oplus \dots \oplus x_i$.



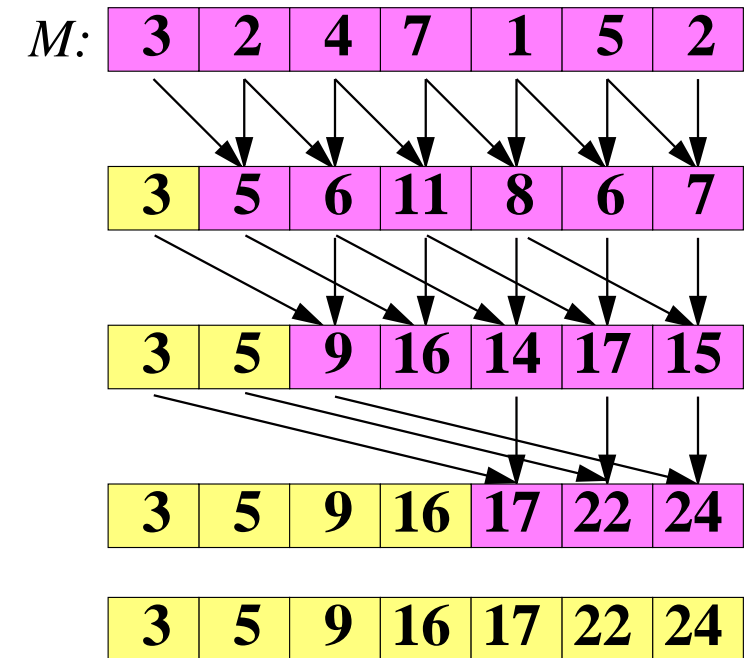
- Sekvenční algoritmus:

```
Algorithm PREFIXSUM(in:  $X[0, \dots, n-1]$ ; out:  $Y[0, \dots, n-1]$ ;)
{
  var  $i$  : int,  $sum$  :  $\mathcal{D}$ ;
   $i := 0$ ;  $sum := X[i]$ ;  $Y[i] := sum$ ;
  while ( $i < n-1$ ) do
    {  $i := i + 1$ ;  $sum := sum \oplus X[i]$ ;  $Y[i] := sum$  }
}
```

- Na počátku: P_0, \dots, P_{n-1} a $M[0], \dots, M[n-1]$ obsahující po řadě x_0, \dots, x_{n-1} .
- P_i má registr y_i .
- Krok j : $M[k] = x_k \oplus x_{k-1} \oplus \dots \oplus x_{k-2^{j+1}+1}$.
- Táž časová složitost jako EREW PRAM paralelní redukce!!!!

```

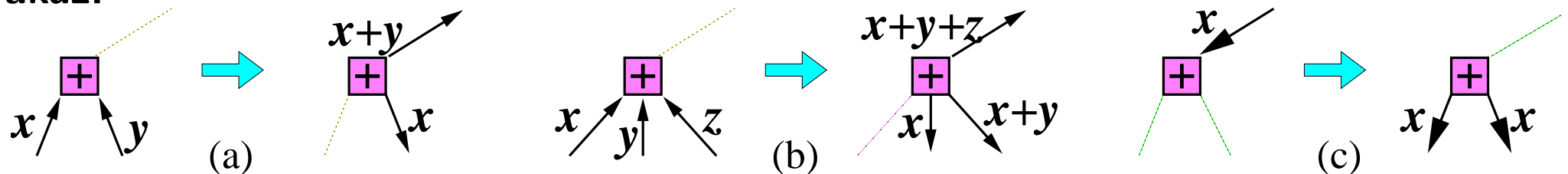
Alg. PRAM_PPS(in,out:  $M[0, \dots, n-1]$ );
for all  $i := 0, \dots, n-1$  do_in_parallel
     $y_i := M[i]$ ;
for  $j := 0, \dots, \lceil \log n \rceil - 1$  do_sequentially
    { for all  $i := 2^j, \dots, n-1$  do_in_parallel
         $y_i := y_i + M[i - 2^j]$ ;
        for all  $i := 2^j, \dots, n-1$  do_in_parallel
             $M[i] := y_i$  }
  
```



Definice 1. *Nepřímý strom:* Vstupní data pouze v listech, vnitřní uzly pouze počítají.

Lemma 2. PPS n vstupních hodnot lze řešit na *binárním nepřímém* stromu T s n listy v $2h(T)$ krocích, kde $h(T)$ je *výška* T .

Důkaz.



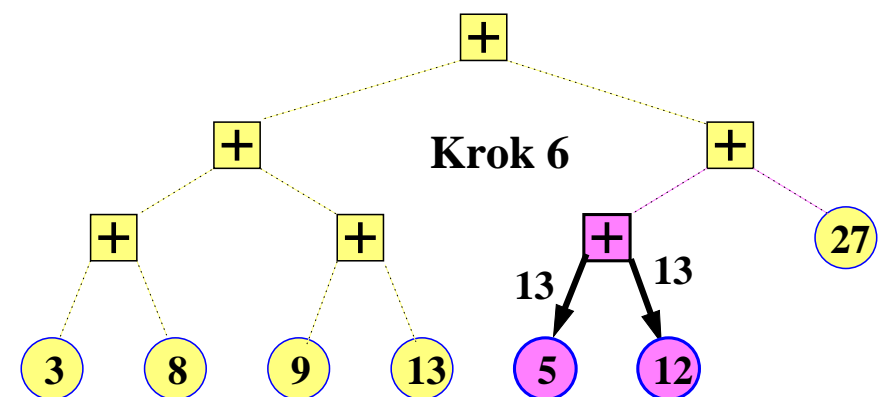
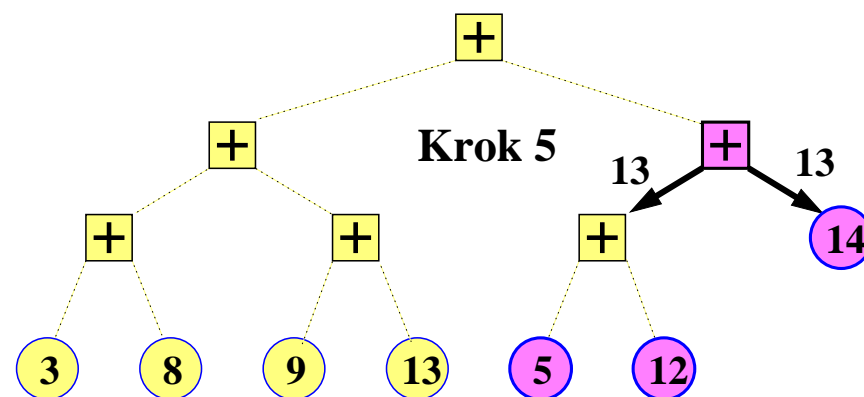
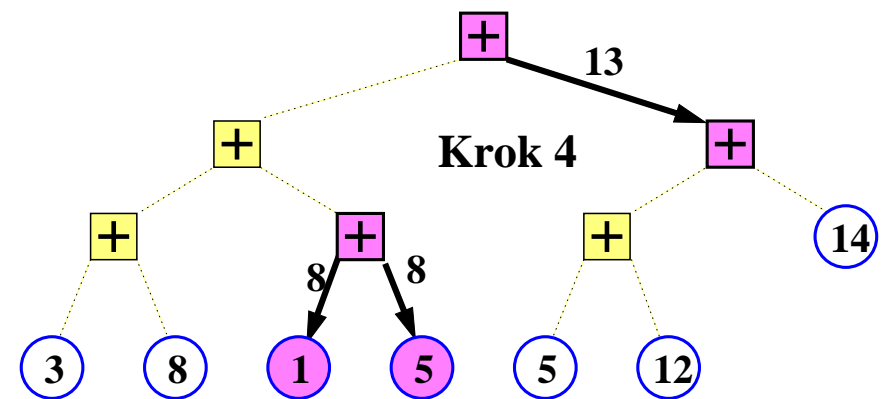
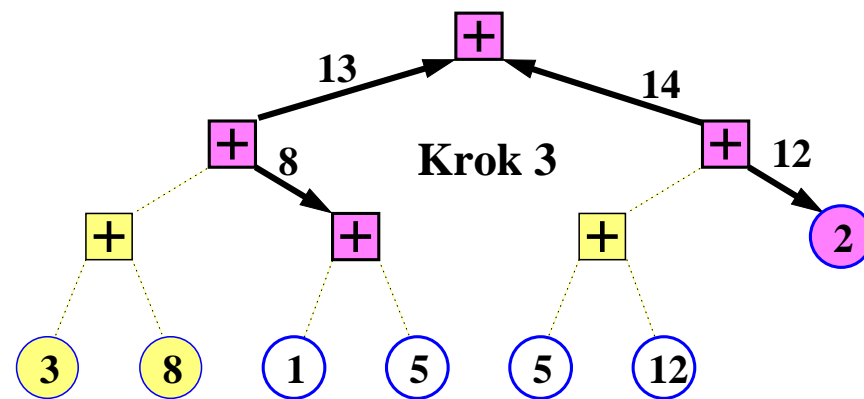
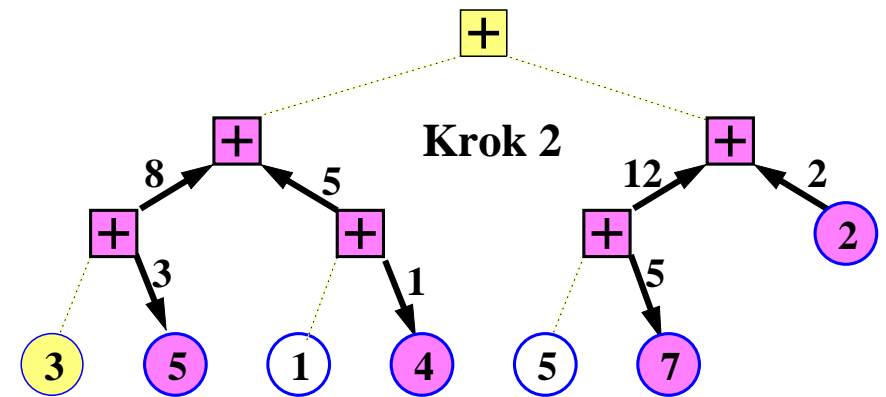
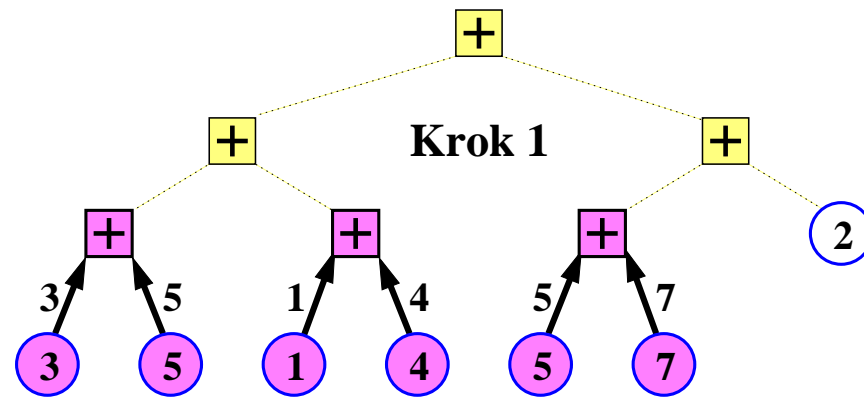
1 vzestupná vlna (a), (b) spouští $h(T)$ sestupných vln (c).



Důsledek 3. $T = \text{úplný binární strom} \implies O(\log n)$ kroků.

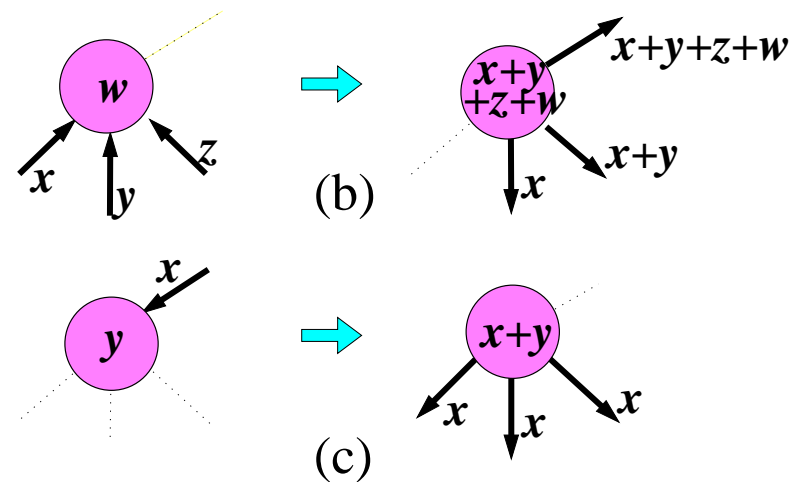
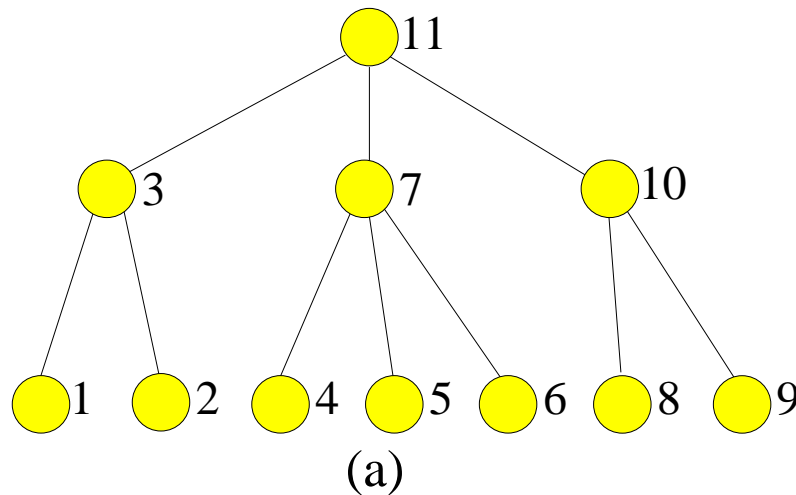
Příklad PPS na nepřímém stromu

6



Lemma 4. Uvažujme *přímý* n -uzlový strom T omezeného stupně větvení a výšky $h(T)$, na který je v pořadí *POSTORDER* namapováno pole A n vstupních hodnot. Pak PPS na poli A lze vypočítat v $2h(T)$ krocích.

Důkaz.



PPS na libovolné topologii

Lemma 5. PPS n vstupních hodnot lze řešit na *libovolné* n -uzlové síti G s omezeným stupněm a s průměrem $\text{diam}(G) = \Omega(\log n)$ v čase $O(\text{diam}(G))$.

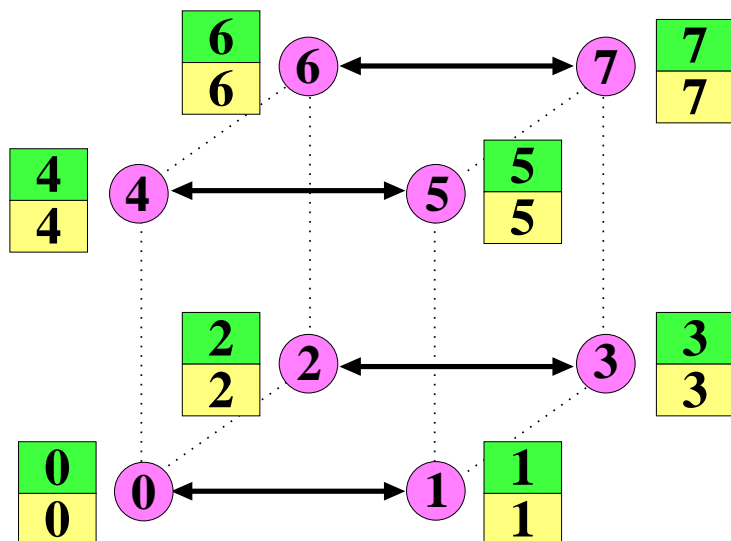
Důkaz.

1. konstrukce kostry do šířky (hloubka = excentricita kořenu $\leq \text{diam}(G)$),
2. aplikace předchozího algoritmu.

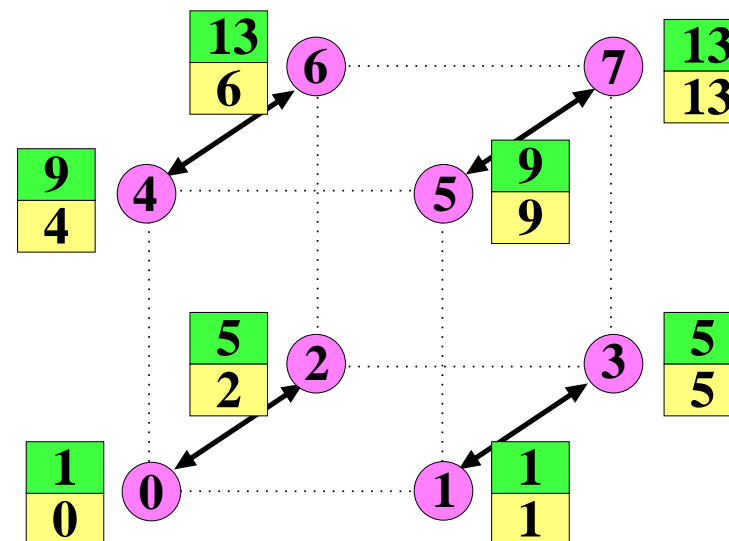


- Uvažujme Q_r , $r \geq 1$.
- Paralelní prefixový součet = normální hyperkubický algoritmus.
- PPS = jednoduché rozšíření algoritmu pro All-to-All Broadcast.
- Každý procesor P_i , $i = 0, \dots, 2^r - 1$, má 2 pomocné registry, $zeleny_i$ a $zluty_i$.

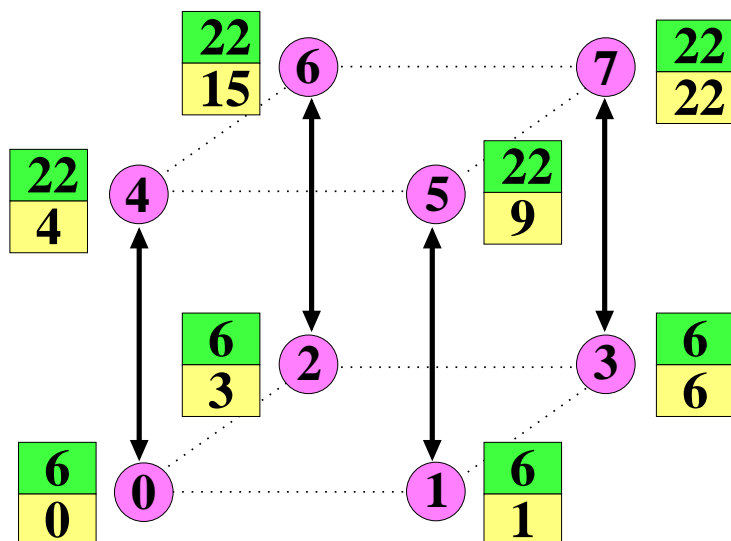
Algorithm HYPERCUBE_PPS($X[1, \dots, n - 1]$)**all** $P_i, i := 0, \dots, 2^r - 1$, **do_in_parallel**{ $zeleny_i := zluty_i := X[i]$;**for** $j := 0, \dots, r - 1$ **do_sequentially**{ **send** $zeleny_i$ **to** $P_{i \text{ XOR } 2^j}$;**receive** $novyzeleny$ **from** $P_{i \text{ XOR } 2^j}$; $zeleny_i := zeleny_i + novyzeleny$;**if** $(i \text{ XOR } 2^j < i)$ **then** $zluty_i := zluty_i + novyzeleny$ }



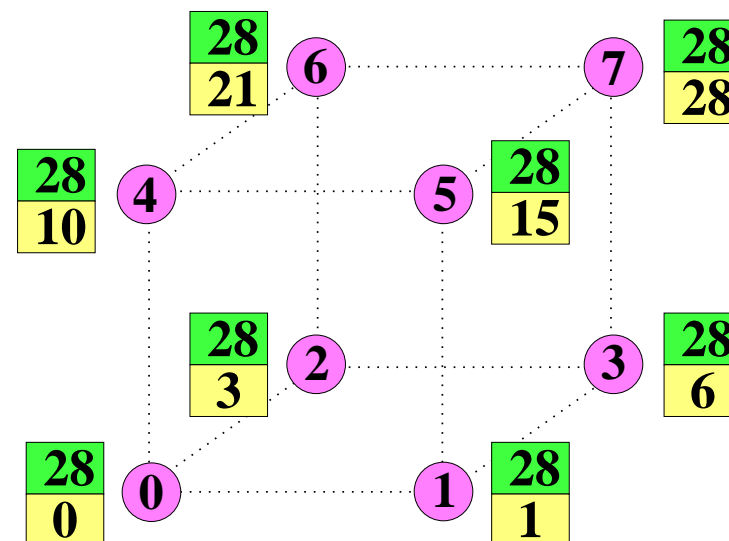
(a) Pocatecni stav



(b) Po provedeni kroku 1

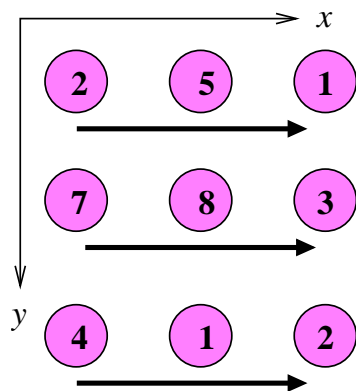


(c) Po provedeni kroku 2



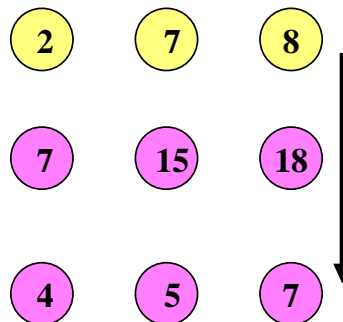
(d) Po provedeni kroku 3

vodorovná fáze



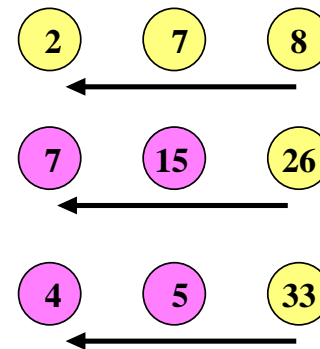
(a)

svislá fáze



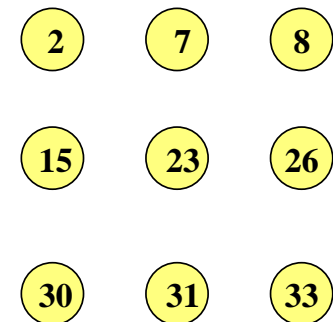
(b)

vodorovná fáze



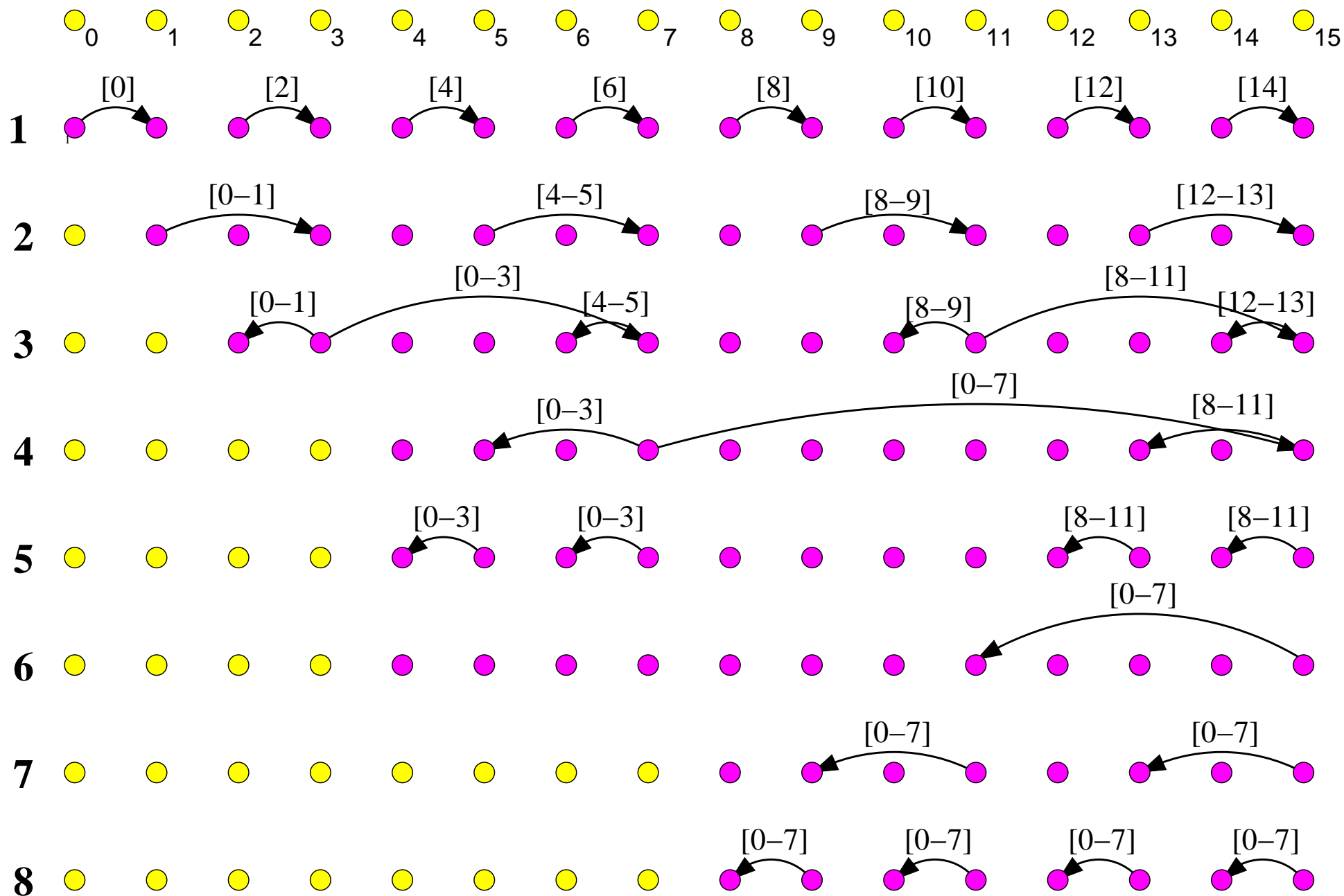
(c)

hotovo



(d)

Základní PPS algoritmus na 1-D 2-portové mřížce



1. Vstupní pole $X = \{x_0, \dots, x_{n-1}\}$ a p procesorů P_0, \dots, P_{p-1} .
2. Nechť $q = n/p$ a X je rozděleno do subpolí X_0, \dots, X_{p-1} .

Algorithm SCALED_PPS($X[1, \dots, n-1]$)

for all $i := 0, \dots, p-1$ **do_in_parallel**

{ P_i sekv. vypočte pole $S_i = [s_{i,0}, \dots, s_{i,q-1}] = \text{prefix součet } X_i$;
>(* $O(q)$ kroků.*)

Definuj $z_i := s_{i,q-1}$. }

všechny P s provedou PPS na $Z = [z_0, \dots, z_{p-1}]$ a vygenerují $[\sigma_0, \dots, \sigma_{p-1}]$;
(* $O(\log p)$ kroků. *)

(* $\sigma_i = \text{celkový součet všech čísel na procesorech } P_0, \dots, P_i$.*)

for all $i := 0, \dots, p-2$ **do_in_parallel** P_i pošle σ_i P_{i+1} ;

for all $i := 1, \dots, p-1$ **do_in_parallel** P_i přijme σ_{i-1} od P_{i-1} ;

for all $i := 1, \dots, p-1$ **do_in_parallel** P_i připočte σ_{i-1} ke všem prvkům S_i ;
(* $O(q)$ kroků. *)

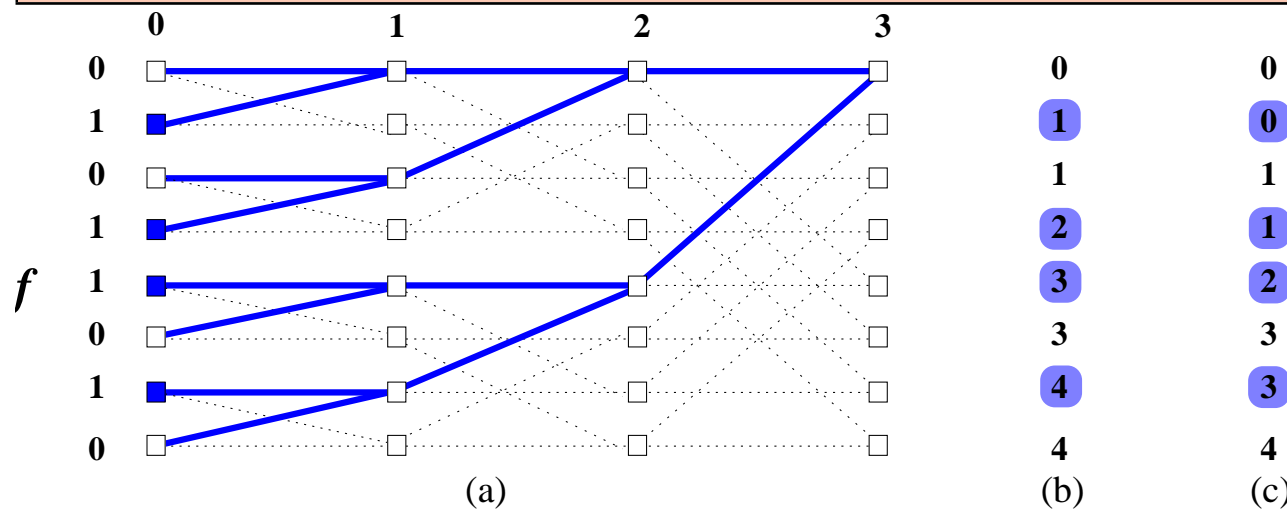
P_0				P_1				P_2				P_3				P_4			
2	1	3		1	2	0		4	2	3		5	0	3		1	4	2	pocatecni rozdeleni
2	3	6		1	3	3		4	6	9		5	5	8		1	5	7	lokalni prefixove soucty
		6				9				18				26				33	paralelni globalni prefix.soucet
																			posun doleva
2	3	6		7	9	9		13	15	18		23	23	26		27	31	33	dopocitani lokalnich souctu

- $T(n, p) = O(n/p + \log p)$ kroků na PRAMech, hyperkrychli, WH mřížkách a sítích s $O(\log p)$ průměrem.
- Paralelní prefixový součet má tutéž škálovatelnost jako paralelní redukce!!!!
- Opět je to normální hyperkubický algoritmus \Rightarrow optimální na hyperkubických sítích.

Zhušťovací problém

1. Každý vstupní uzel má příznak f , nastavený buď na 0 nebo 1.
2. Nejprve je nutno vypočítat **index** v rámci distribuované podmnožiny \mathcal{M} uzlů s $f = 1$.
3. Pak je paket z i -tého uzlu v rámci \mathcal{M} poslán na výstup číslo i .

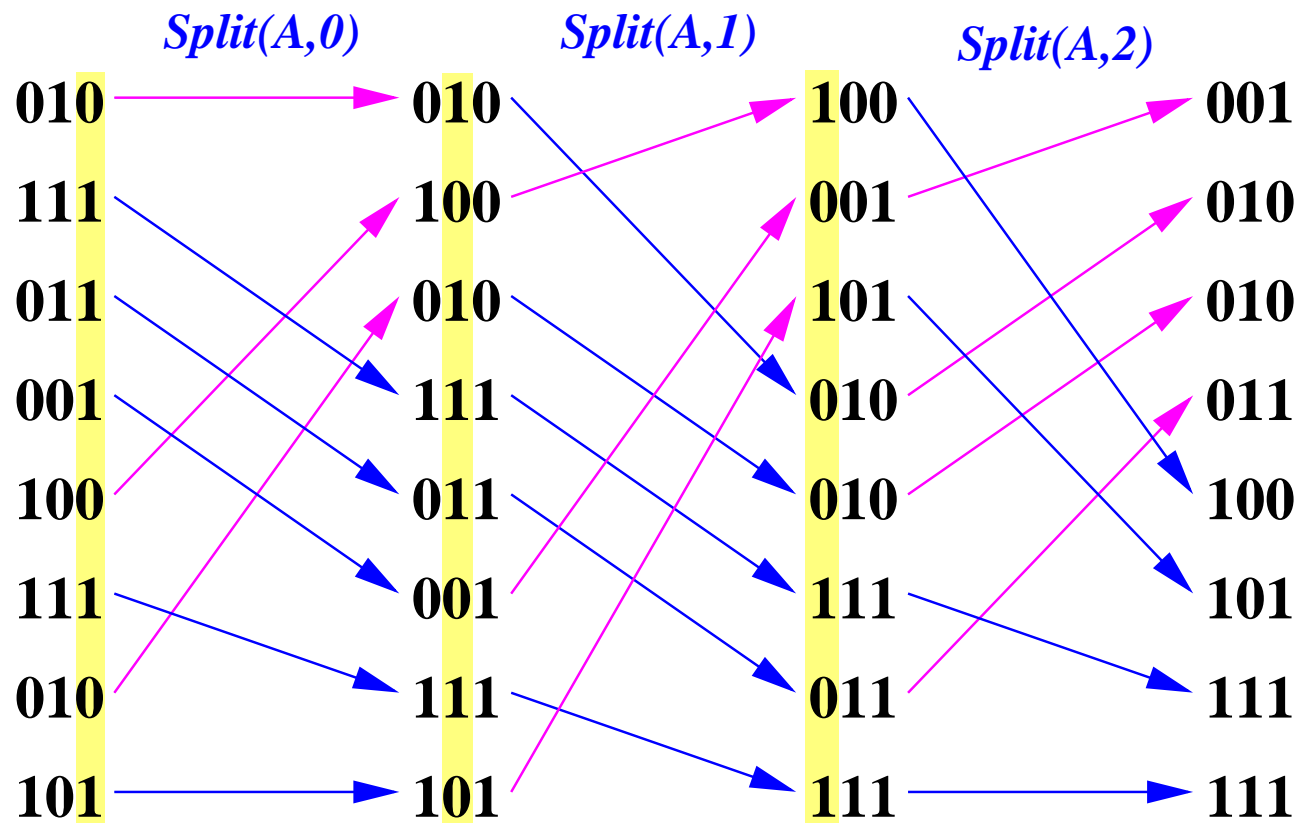
Zhušťovací problém na nepřímém motýlku $indBF_3$



- (a) Počáteční hodnoty příznaků a jeden možný nepřímý strom.
- (b) Pole příznaků po paralelním prefixovém součtu.
- (c) Konečné hodnoty indexů cílových řádků.

Algorithm RADIXSORT($A[0, \dots, 2^n - 1]$)
for $i := 0, \dots, n - 1$ **do_sequentially** $Split(A, i)$

- $Split(A, i)$ = permutace A : všechna čísla, jejichž bit i je 0, jsou vytlačena **nahoru** a všechna čísla, jejichž bit i je 1, jsou stlačena **dolů**.
 $\implies Split(A, i) = 2$ paralelní prefixové součty + 2 zhušťovací permutační směřování.
- Paralelní časová složitost RadixSortu 2^n čísel na oBF_n je $O(n^2)$.



- Vstup: 2 n -bitová binární čísla $X = x_{n-1} \dots x_0$ a $Y = y_{n-1} \dots y_0$.
- Jak vypočítat $Z = z_{n-1} \dots z_0 = X + Y$ v $\log n$ krocích? Předpočítáním **přenosových** bitů.
- Výstup: $C = c_n, \dots, c_1, c_0$, kde vždy $c_0 = 0$.

```

for  $i := 0, \dots, n - 1$  do_in_parallel
     $b_i := \begin{cases} g(enerate) & \text{if } x_i = y_i = 1, \\ s(top) & \text{if } x_i = y_i = 0, \\ p(ropagate) & \text{jinak;} \end{cases}$ 
    vypočti  $B' = [b'_{n-1}, \dots, b'_0]$  použitím
    PPS  $\odot$  nad polem  $[b_{n-1}, \dots, b_0, s]$ 

    kde
        

| $\odot$ | $s$ | $p$ | $g$ |
|---------|-----|-----|-----|
| $s$     | $s$ | $s$ | $s$ |
| $p$     | $s$ | $p$ | $g$ |
| $g$     | $g$ | $g$ | $g$ |


        ;

    for  $i := 1, \dots, n$  do_in_parallel
         $c_i := 1 \iff b'_{i-1} = g;$ 
    
```

X:	0	1	0	1	1	0	1	1
Y:	0	1	1	0	1	0	0	1
B:	s	g	p	p	g	s	p	g
B':	s	g	g	g	g	s	g	g
C:	0	1	1	1	1	0	1	1
Z:	1	1	0	0	0	1	0	0

$$\begin{array}{rcl}
 g_1 x_1 + h_1 x_2 & & = b_1 \\
 f_2 x_1 + g_2 x_2 + h_2 x_3 & & = b_2 \\
 \dots\dots\dots & & \\
 f_i x_{i-1} + g_i x_i + h_i x_{i+1} & & = b_i \\
 \dots\dots\dots & & \\
 f_n x_{n-1} + g_n x_n & = & b_n
 \end{array}$$

Hlavní trik algoritmu: přepiš i -tou rovnici do následujícího rekurentního tvaru

$$\begin{bmatrix} x_{i+1} \\ x_i \\ 1 \end{bmatrix} = \mathcal{G}_i \begin{bmatrix} x_i \\ x_{i-1} \\ 1 \end{bmatrix},$$

$$\text{kde} \quad \mathcal{G}_i = \begin{bmatrix} -\frac{g_i}{h_i} & -\frac{f_i}{h_i} & \frac{b_i}{h_i} \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad 1 \leq i \leq n-1.$$

Opakovanou substitucí dostaneme

$$\begin{bmatrix} x_{i+1} \\ x_i \\ 1 \end{bmatrix} = \mathcal{H}_i \begin{bmatrix} x_1 \\ 0 \\ 1 \end{bmatrix}, \quad \text{kde} \quad \mathcal{H}_i = \mathcal{G}_i \mathcal{G}_{i-1} \dots \mathcal{G}_1, \quad 1 \leq i \leq n-1. \quad (1)$$

Algoritmus na p procesorech P_0, \dots, P_{p-1} pracuje takto:

1. Aplikováním PPS na pole matic $\mathcal{G}_1, \dots, \mathcal{G}_{n-1}$ procesory vypočtou $\mathcal{H}_1, \dots, \mathcal{H}_{n-1}$. Asociativním operátorem je násobení (3×3) -matic ($O(n/p + \log p)$ kroků).
2. Procesor P_{p-1} vyřeší 3 rovnice

$$\begin{bmatrix} x_n \\ x_{n-1} \\ 1 \end{bmatrix} = \mathcal{H}_{n-1} \begin{bmatrix} x_1 \\ 0 \\ 1 \end{bmatrix},$$

$$f_n x_{n-1} + g_n x_n = b_n$$

s 3 neznámými a vypočte x_1 ($O(1)$ kroků).

3. P_{p-1} vyšle hodnotu x_1 všem procesorům ($O(\log p)$ kroků).
4. Všechny procesory paralelně vypočtou x_{i+1} ze všech svých rovnic (1) ($O(n/p)$ kroků).

- Vstupní pole je rozděleno do **segmentů**.
- Cílem je vypočítat všechny prefixové součty **uvnitř těchto segmentů izolovaně**.

Příklad: 4-segmentové vstupní pole:

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline & 2 & 1 & 3 & 5 & 2 & 7 & 3 & 9 & 4 & 5 & 6 & 2 & 8 & 4 & 3 & 1 \\ \hline \end{array}$$
 Provedením 4 izolovaných paralelních prefixových součtů s operací $+$ získáme

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline & 2 & 3 & 6 & 11 & 2 & 9 & 12 & 9 & 13 & 18 & 24 & 2 & 10 & 14 & 17 & 18 \\ \hline \end{array}$$



Hlavní myšlenka SPPS

SPPS = 1 globální PPS aplikovaný na celé pole s **modifikovanou** bin. operací $\overline{\oplus}$

$\overline{\oplus}$	b	b
a	$a \oplus b$	b
a	$(a \oplus b)$	b

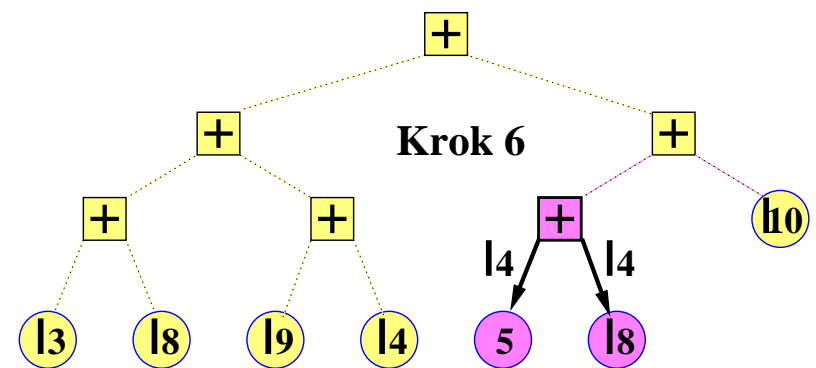
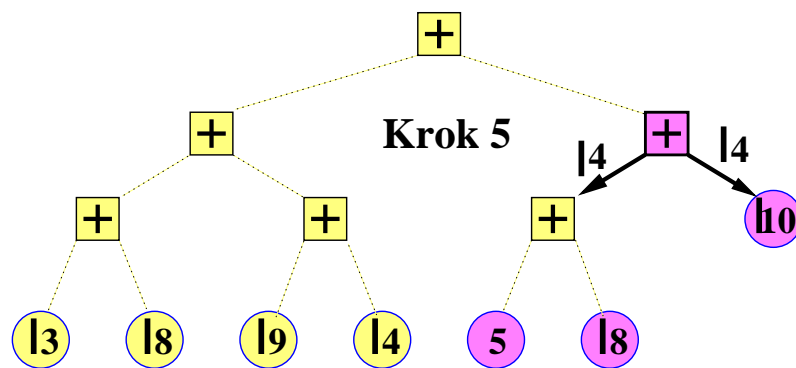
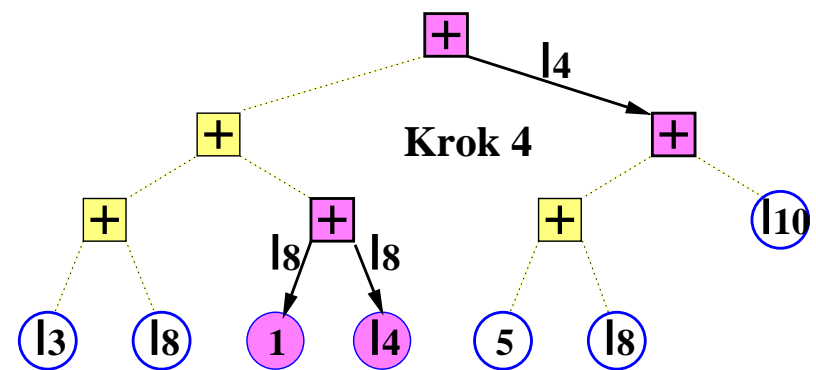
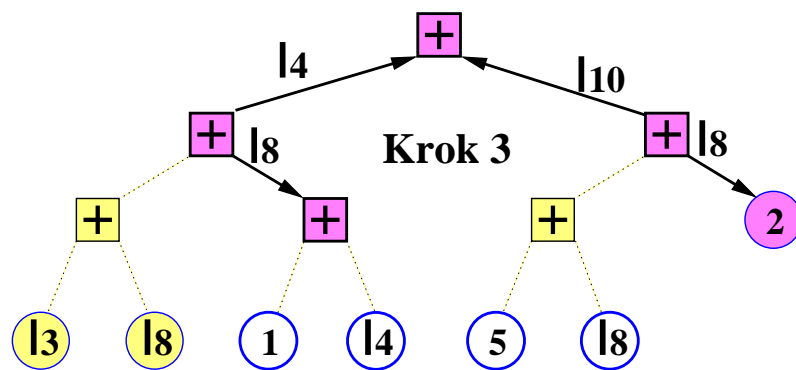
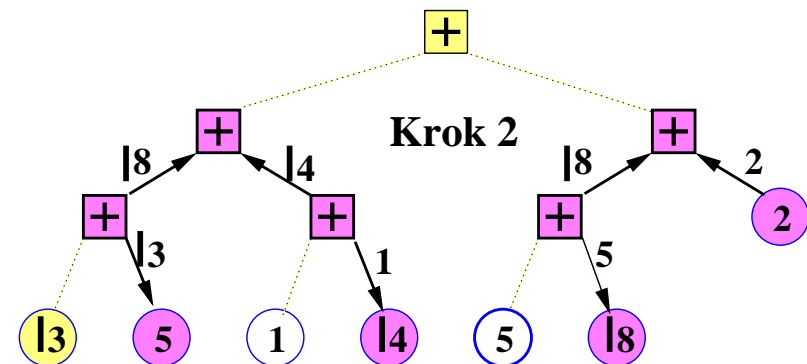
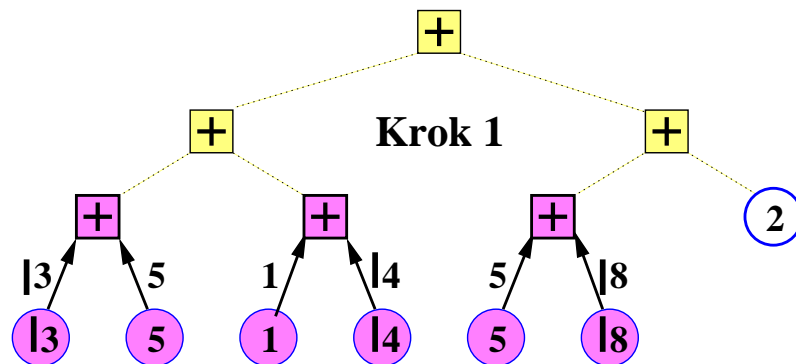
Lemma 6. *Je-li \oplus asociativní, je asociativní i $\overline{\oplus}$.*



Příklad segmentovaného paralelního prefixového součtu

20

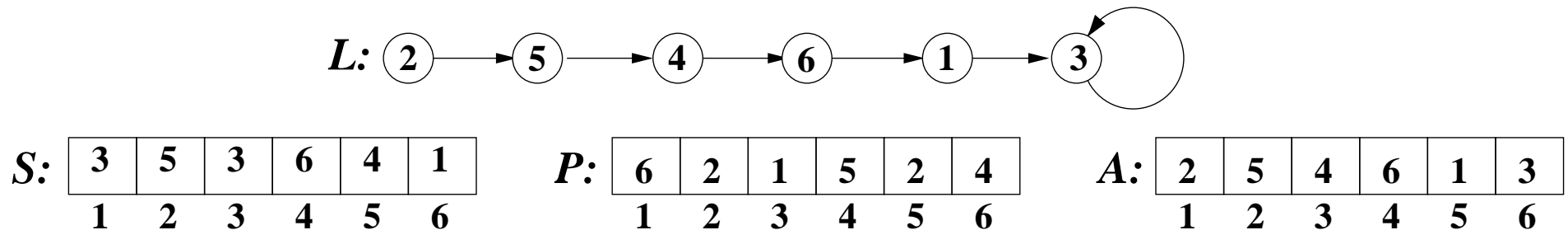
Vstupní pole: | 3 5 1 | 4 5 | 8 2 |



- Jeden z několika paralelních QuickSort algoritmů.
- Vstupní posloupnost $A = \{a_0, a_2, \dots, a_{n-1}\}$ je rozdělena rovnoměrně mezi $p < n$ procesorů.
- Z počátku, $A =$ jeden globální segment $S_0 = |a_0, a_2, \dots, a_{n-1}$.
- V jedné iteraci, každý segment S je rozdělen do 3 podsegmentů $S_{<}, S_{=}, S_{>}$.
- Každá iterace vyžaduje $O(1)$ volání SPPS a $O(1)$ paralelních PRAM operací.
- Je-li A náhodná, pak by QuickSort měl skončit v $O(\log n)$ iteracích
 $\implies T_{\text{QuickSort}}(n, p) = O(\log n)T_{\text{PPS}}(n, p).$

B:for $i := 0, \dots, n - 2$ **do_in_parallel** $f_i := (a_i \leq a_{i+1})$; (* je A setříděný? *)
 proved' paralelní redukci hodnot f_i operací AND;
if (A je setříděný) **then EXIT**;
for všechny momentální segmenty S v A **do_in_parallel**
 { vyber první prvek segmentu S jako jeho pivot b_S ;
 (* posláni hodnot všech pivotů přesně těm procesorům, které je potřebují *)
 pomocí SPPS s \oplus definovanou jako $a \oplus b = a$ pošli b_S procesorům uvnitř S ;
 $\forall a_i \in S$ **do_in_parallel**
 $g_i := (a_i <> b_S)$, kde $g_i \in \{ '<', '= ', '>' \}$;
 (* nyní musíme každý S rozdělit do 3 podsegmentů $S_<, S_ =, S_>$ *)
 pomocí SPPS výpočti nové indexy prvků $\{a_i; g_i = '<'\}$ uvnitř všech S ;
 pošli hodnoty maximálního indexu ($= |S_<|$) uvnitř všech S (SPPS doleva);
 (* $|S_<|$ je rovno indexu začátku $S_ =$ *)
 (* první prvek s indexem 1 = okrajový prvek $S_<$ *)
 pomocí SPPS výpočti nové indexy prvků $\{a_i; g_i = '= '\}$ uvnitř všech S ;
 pošli hodnoty maximálního indexu ($= |S_ =|$) uvnitř všech S (SPPS doleva);
 (* $|S_<| + |S_ =|$ je rovno indexu začátku $S_>$ *)
 (* první prvek s indexem 1 = okrajový prvek $S_ =$ *)
 pomocí SPPS výpočti nové indexy prvků $\{a_i; g_i = '>'\}$ uvnitř všech S ;
 (* první prvek s indexem 1 = okrajový prvek $S_>$ *)
 permutací segmentu S vytvoř 3 podsegmenty $S = \{S_<|S_ =|S_>\}$;
 };
 goto B;

- n -prvkový **jednosměrně zřetězený seznam** L je reprezentován pomocí pole **následníků** S (successors).
- Každý prvek má **jednoznačné ID** z $\{1, \dots, n\}$.



Konverze jednosměrně zřetězeného seznamu na obousměrně

- Platí: $P[i] = j \iff S[j] = i$.
- Plně paralelní (t.j. lineárně škálovatelný) EREW PRAM(n, p) algoritmus:
 $T(n, p) = O(n/p)$.

Algorithm EREW_PRAM_SINGLE2DOUBLE (**in:** $S[1, \dots, n]$; **out:** $P[1, \dots, n]$)

for all $i := 1, \dots, n$ **do_in_parallel**

$\{P[i] := i; \text{ if } (S[i] \neq i) \text{ then } P[S[i]] := i\}$

- **Pořadí** = vzdálenost (= počet ukazatelů) do konce seznamu.
- **Sekvenční řešení**: triviální algoritmus **projitím** ukazatelů.

Výpočet pořadí pomocí přeskočků ukazatelů (PU)

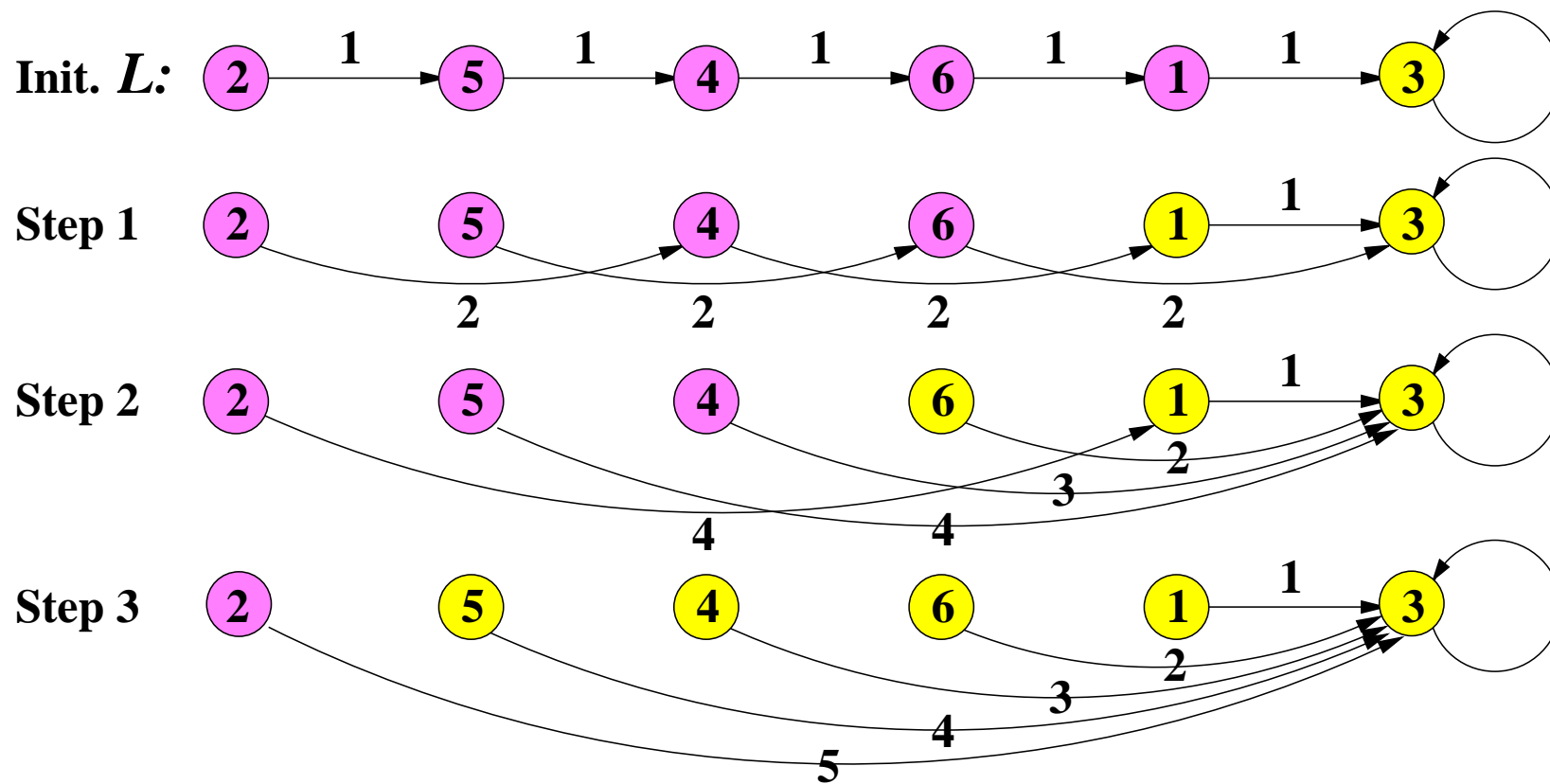
Algorithm CREW_PRAM_LIST_RANKING (**in,out:** $S[1, \dots, n]$, **out:** $R[1, \dots, n]$)

for all $i := 1, \dots, n$ **do_in_parallel**
 { **if** ($S[i] = i$) **then** $R[i] := 0$ **else** $R[i] := 1$;
 repeat $\lceil \log n \rceil$ **times**
 { $R[i] := R[i] + R[S[i]]$; $S[i] := S[S[i]]$ } (* pointer jumping *)

Paralelní postfixový součet na zřetěženém seznamu

Alg. CREW_PRAM_PARALLEL_SUFFIX_SUM (**in,out:** $S[1, \dots, n]$, **out:** $V[1, \dots, n]$)

for all $i := 1, \dots, n$ **do_in_parallel**
 { **if** ($S[i] = i$) **then** $V[i] := 0$ **else** $V[i] := v_i$;
 repeat $\lceil \log n \rceil$ **times**
 { $V[i] := V[i] \oplus V[S[i]]$; $S[i] := S[S[i]]$ }; (* pointer jumping *)
 if (original $V[last] \neq 0$) **then** $V[i] := V[i] \oplus V[last]$ }



Init. S :

3	5	3	6	4	1
---	---	---	---	---	---

Step 1

3	4	3	1	6	3
---	---	---	---	---	---

Step 2

3	1	3	3	3	3
---	---	---	---	---	---

Step 3

3	3	3	3	3	3
---	---	---	---	---	---

1	2	3	4	5	6
---	---	---	---	---	---

R :

1	1	0	1	1	1
---	---	---	---	---	---

1	2	0	2	2	2
---	---	---	---	---	---


1	4	0	3	4	2
---	---	---	---	---	---

1	5	0	3	4	2
---	---	---	---	---	---

1	2	3	4	5	6
---	---	---	---	---	---

- Znalost pořadí dovoluje **transformovat** zřetězený seznam na lineární **pole**: prvky jsou **permutovány** na **pozice** dané svým pořadím v seznamu.
- Na tomto novém poli lze po té veškeré další výpočty provádět pomocí **PPS**.

Škálovatelnost přeskakování ukazatelů (PU)

- CREW PRAM(n, n) PU není cenově optimální: $C(n, n) = O(n \log n)$.
- PU je **datově necitlivé**: provádí přeskoky i přes ukazatele na poslední prvek. Tudíž taktéž $W(n, n) = O(n \log n)$.
- Avšak, **cenovou optimalizaci** nelze založit na zmenšení p a přidělení n/p prvků každému procesoru, neboť
 - Po kroku i je # **spočítaných** prvků $2^{i-1} + 1$ pro $i \geq 1$ (viz žluté prvky  na předchozím obr.).
 - $\Theta(n/2)$ prvků vyžadují $\Theta(\log n)$ přeskoků, než se dostanou na konec.
 - CREW PRAM(n, p) algoritmus PU může být **pracovně-optimální** pouze, když každý P udržuje seznam **aktivních** (=nedopočítaných) prvků.
 - Ale ani tato optimalizovaná verze nezlepší cenu, protože 1 P může mít n/p aktivních prvků v každém z $\log n$ kroků a tudíž,
 $T(n, p) = O((n/p) \log n)$ a $C(n, p) = O(n \log n)$.

Uvažujme zřetěžený seznam s $n' = n / \log n$ prvky

a

použijme přeskokování ukazatelů pomocí $p = n / \log n$ procesorů

$$\implies T(n', p) = O(\log n) \text{ a } C(n', p) = O(n).$$

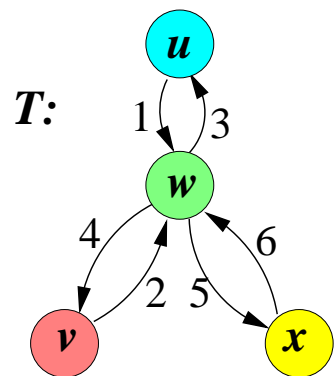
Myšlenka škálovatelného algoritmu pro výpočet pořadí v seznamu

0. Nechť $|L| = n$ prvků a $p = \Theta(n / \log n)$.
1. Transformuj L na L' velikosti $n' = O(n / \log n)$ s $C_1(n, p) = O(n)$,
t.j., v $T_1(n, p) = O(\log n)$ pomocí p procesorů.
2. Proveď přeskokování ukazatelů na L' . Pak
 $T_2(n', p) = O(\log n') = O(\log n)$ a $C_2(n', p) = O(n)$.
3. Restauruj L z L' a vypočti konečné pořadí pro prvky v $L - L'$
s toutéž složitostí jako v kroku 2, tedy $T_3(n, p) = O(T_1(n, p))$.

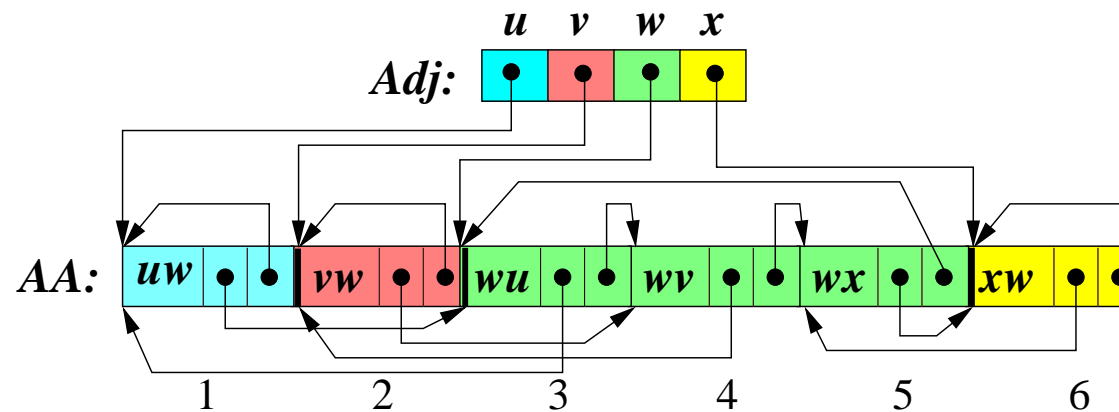
- **Eulerovské cykly** G = kružnice, které procházejí každou hranou G **přesně jednou**.
- G je eulerovský, má-li **každý** uzel G **sudý** stupeň.
- Neorientovaný **strom** může být transformován na **eulerovský**, je-li každá hrana nahrazena **dvojití anti-paralelních hran**. (a)

Reprezentace grafu a Eulerovské stromy

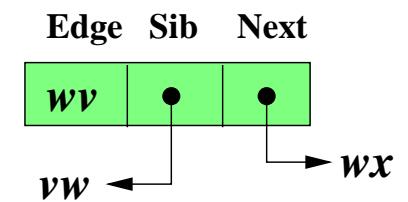
- **Pole uzlů** (Adj): $x \in V(T)$, $Adj[x]$ = ukazatel do pole hran AA . (b)
- **Pole hran** (AA) se skládá z podseznamů hran incidentních s jednotlivými uzly. (b)+(c)



(a)



(b)



(c)

Hrany $\langle x \rightarrow y \rangle$ jsou pro jednoduchost značeny xy . Sib je ukazatel na anti-paralelní **dvojče**.

Eulerovská kružnice stromu $T' =$ cyklický zřetězený seznam hran,
reprezentovaný jako pole **následníků** ET :

$$ET:$$

4	5	1	2	6	3
1	2	3	4	5	6

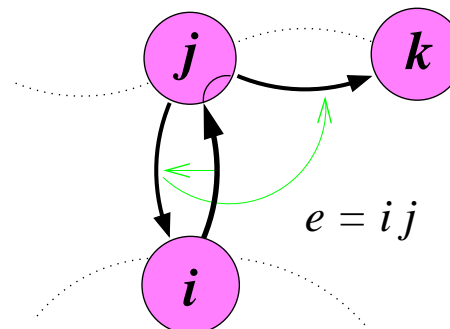
Eulerovská kružnice může být vypočítána **plně paralelním EREW PRAM** algoritmem.

Algorithm EREW_PRAM_EULERIAN_TOUR (**in:** AA ; **out:** ET ;)

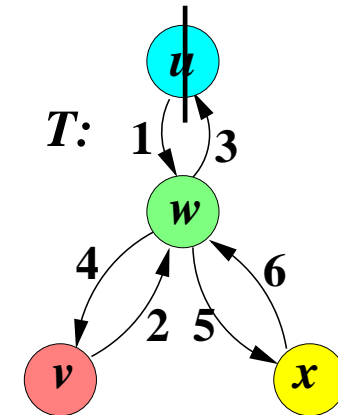
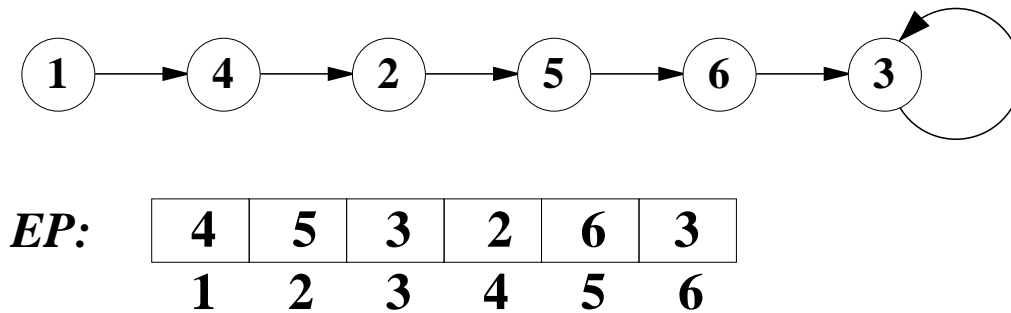
for all arcs e in AA of a tree T **do_in_parallel**
 $ET[e] := (AA[e].Sib \uparrow).Next$

Pravidlo procházení bludištěm:

kráčeť kupředu a stále přidržuj pravou ruku v kontaktu s pravou zdí.

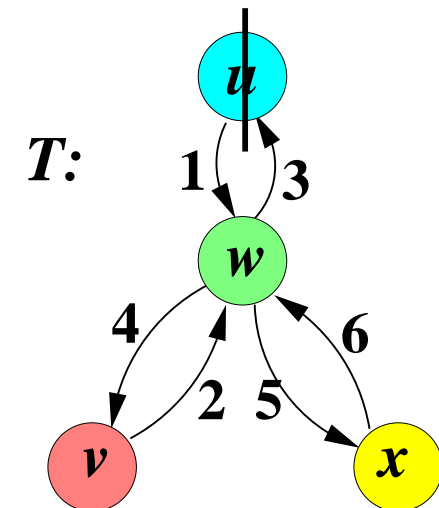
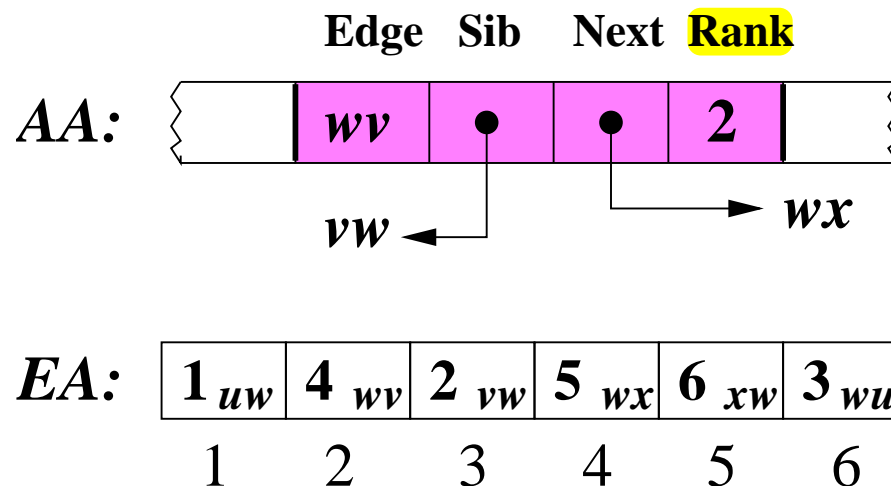


Rozetní eulerovskou kružnici ve zvoleném uzlu.



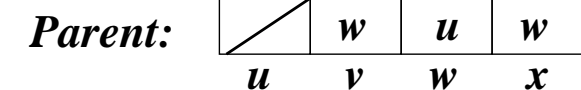
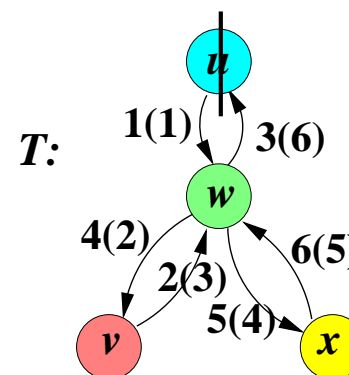
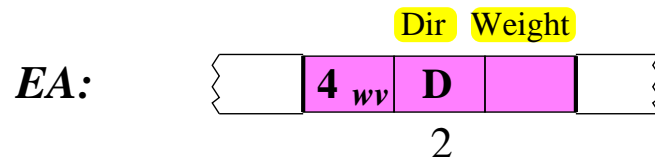
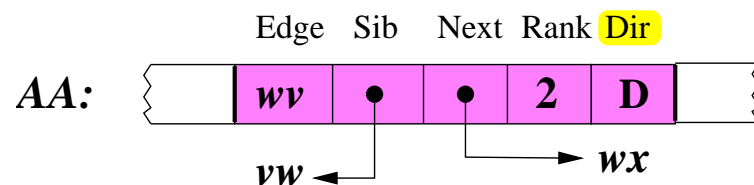
Výpočet pořadí

Pořadí hrany v $T' =$ její pozice od **začátku** eulerovské cesty (pole $Rank[...]$)
 Pak EP (zřetězený seznam) může být paralelně transformován na **pole** EA .



Myšlenka: hrany v eulerovské cestě označ jako jdoucí **dolu** (D = down) nebo **nahoru** (U = up).

Alg. EREW_PRAM_ALLPARENTS (**In:** $AA, EA[1, \dots, m]$; **Out:** $Parent[1, \dots, n]$);
for all arcs xy **do_in_parallel**
 if ($AA[xy].Rank < AA[yx].Rank$) **then**
 { $AA[xy].Dir := EA[AA[xy].Rank].Dir := \text{Down};$
 $AA[yx].Dir := EA[AA[yx].Rank].Dir := \text{Up};$ $Parent[y] := x$; }
 $Parent[root] := \text{nil};$



Myšlenka: počet uzlů v podstromu = $1/2$ počtu hran podstromu.

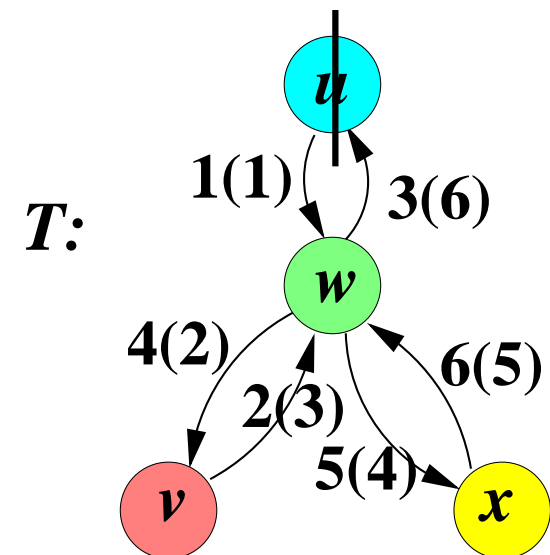
Alg. EREW_PRAM_ALLSUBTREESIZESI (**In:** AA , $Parent$; **Out:** $Subtree[1, \dots, n]$);
for all vertices $i \neq root$ in T **do_in_parallel**
 $Subtree[i] = (AA[ij].Rank - AA[ji].Rank + 1)/2$, where $j = Parent[i]$;
 $Subtree[root] := n$;

Parent:

	w	u	w
u	v	w	x

Subtree:

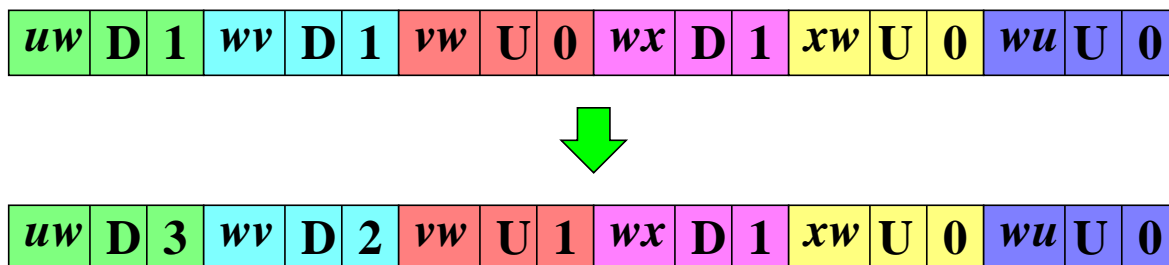
4	1	3	1
u	v	w	x



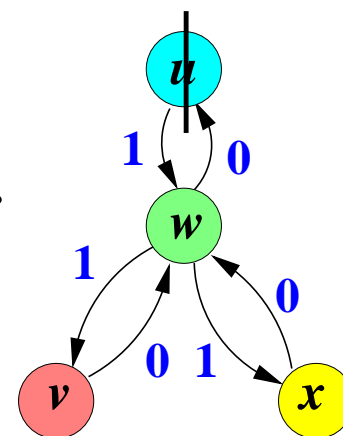
Myšlenka: velikost podstromu = počet jeho Down hran.

Alg. EREW_PRAM_ALLSUBTREESIZESII (**In:** $EA[1, \dots, m]$, **Out:** $Subtree[1, \dots, n]$);
for all arcs $e \in EA$ **do_in_parallel**
 if ($EA[e].Dir = \text{Down}$) **then** $EA[e].Weight := 1$ **else** $EA[e].Weight := 0$;
apply PSS on the array $EA[1, \dots, m].Weight$;
for all arcs $xy \in EA$ **do_in_parallel**
 if ($EA[xy].Dir = \text{Down}$) **then** $Subtree[y] := EA[xy].Weight - EA[yx].Weight$;
 $Subtree[root] := n$;

EA:



T:

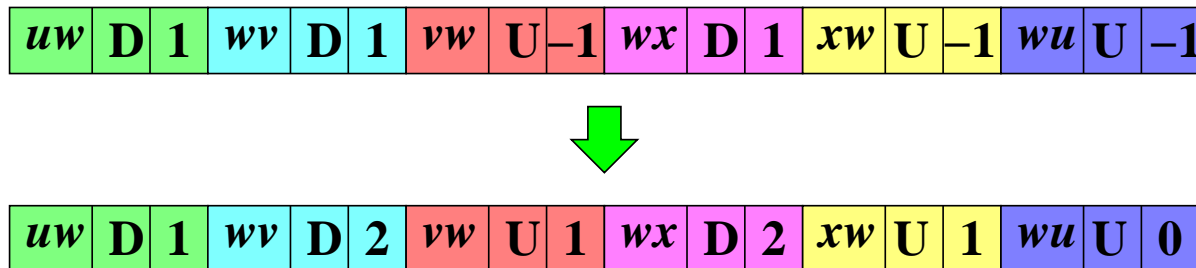


Myšlenka: paralelní párování závorek.

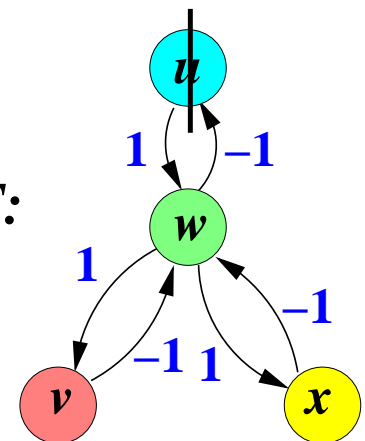
Alg. EREW_PRAM_ALLLEVELS (**In:** $EA[1, \dots, m]$, **Out:** $Level[1, \dots, n]$);

for all arcs $e \in EA$ **do_in_parallel**
 if ($EA[e].Dir = \text{Down}$) **then** $EA[e].Weight := 1$ **else** $EA[e].Weight := -1$;
apply PPS on array $EA[1, \dots, m].Weight$;
for all arcs $xy \in EA$ **do_in_parallel**
 if ($EA[xy].Dir = \text{Down}$) **then** $Level[y] := EA[xy].Weight$;
 $Level[root] := 0$;

EA:



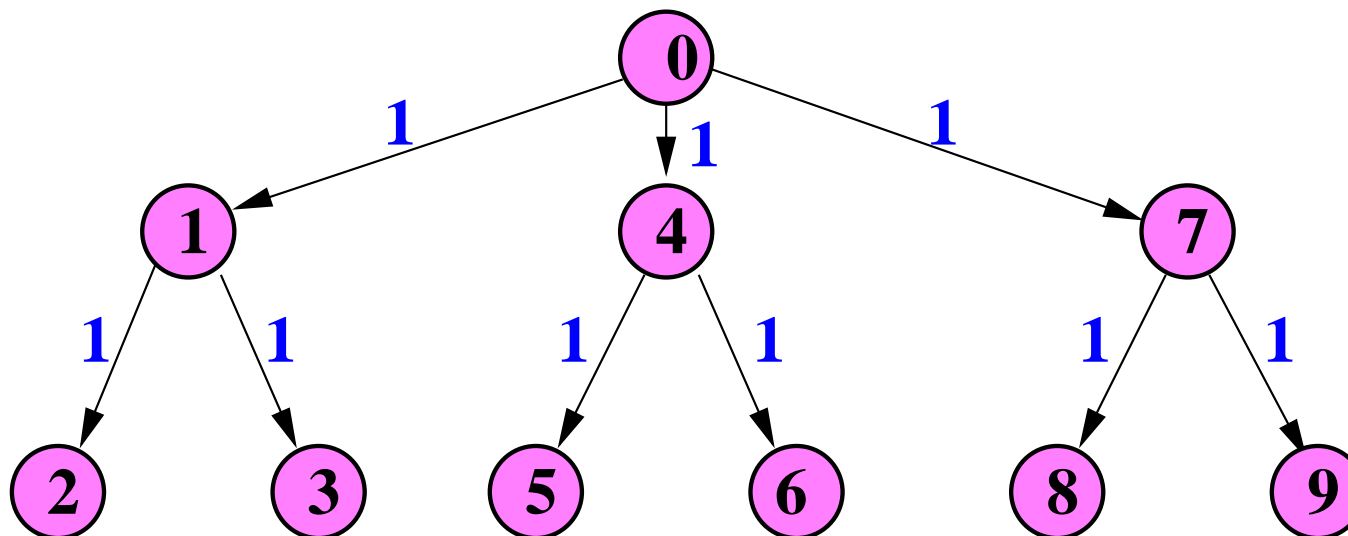
T:



- průchod Preorder (počínaje od 0): kořen, levý podstrom, pravý podstrom.

Myšlenka: Preorder pořadí uzlu = počet Down hran v ET z kořene do uzlu.

Alg. EREW_PRAM_PREORDER (**In:** $EA[1, \dots, m]$; **Out:** $Pre[1, \dots, n]$);
for all arcs $e \in EA$ **do_in_parallel**
 if ($EA[e].Dir = \text{Down}$) **then** $EA[e].Weight := 1$ **else** $EA[e].Weight := 0$;
apply PPS on $EA[1, \dots, m].Weight$;
for all arcs $xy \in EA$ **do_in_parallel**
 if ($EA[e].Dir = \text{Down}$) **then** $Pre[y] := EA[xy].Weight$;



Technika eulerovské cesty je základem pro široké spektrum dalších paralelních výpočtů na grafech, jako např.

- výpočet minimální kostry,
- vrcholová/hranová souvislost,
- ear decomposition search,
- testování planarity.