MI-SPOL-15

Paralelní redukce, prefixový součet, segmentový prefixový součet na PRAM, ortogonálních a hyperkubických sítích, v OpenMP a MPI.

Paralelní redukce

Vstupy:

- Pole $X[0, ..., n-1]$ z množiny D a množina p procesů
- Asociatovní a komutativní binární operace $\oplus$ v D

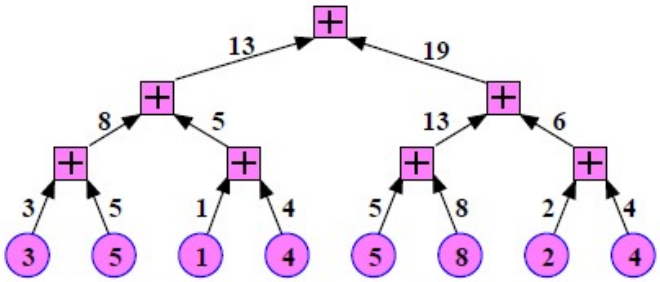
Výstup:

- Globální redukovaná hodnota prvků $S = X[0] \oplus X[1] \oplus ... \oplus X[n-1]$

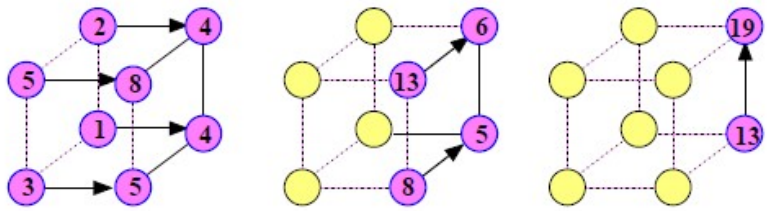
Vlastnosti:

- Paralelní čas: $T(n,p) = \alpha n/p + \beta \log p$
- Dolní mez: $L(n,n) = \Omega(\log n)$
- Časová optimalita: $T_{min}(n,p) = T(n, n/c) = \Theta(\log n)$
- Dobrá škálovatelnost: $\psi_1(p) = p \log p$ a $\psi_2(n) = n/\log n$
- Normální hyperkubická algoritmus $\Rightarrow$ optimální na hyperkubických sítích

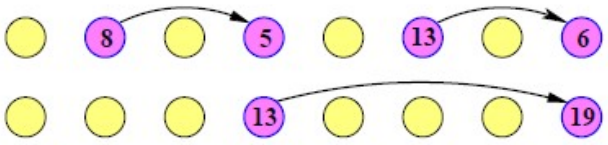
Implementace:



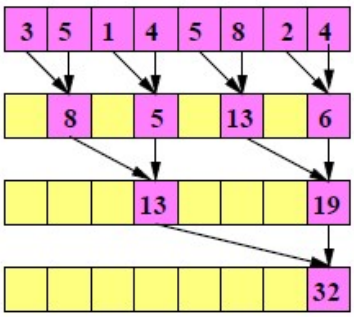
(a) (Neprimy) uplny binarni strom



(b) Uplny graf a SF hyperkrychle



(c) WH 1−D mrizka



(d) EREW PRAM

Redukce v OpenMP

Vlastnost `reduction(operator:variable)`

Sdílená proměnná **lokálně nakopírována** do každého vlákna.
Po skončení paralelního regionu redukce všech lokálních instancí pomocí zadaného operátoru, výsledek do **původní sdílené proměnné**

Proměnná musí být **skalárního typu**

Operátory: +, *, -, &, ^, |, &&, ||
Operátor **nesmí být přetížen.**
Operátor **nemusí být asociativní** pro reálná čísla

1 region -- i více redukovaných proměnných, pouze **jedna proměnná v jedné redukci**

Implementace:

- **Lineární:** Všechna vlákna zapíší svá data do sdíleného pole, po bariéře jedno sekvenčně spočítá výsledek
  $T_{PR}(n,p) = (\frac{n}{p} + p)(T_{comp} + T_{mem}) + T_{barr}(p)$
- **Logaritmická:** vyžaduje bariéru v každém kroku
  $T_{PR} = (\frac{n}{p} + \log p)(T_{comp} + T_{mem}) + \log p \cdot T_{barr}(p)$

Protože v OpenMP obvykle jednoduché operátory, $T_{comp}, T_{mem} << T_{barr}$ a počet vláken je malý ($p << n$), vyplatí je v OpenMP **lineární verze**

Redukce v MPI

```
MPI_Reduce(const void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype, MPI_OP op, int root, MPI_Comm comm)
```

Výstup uložen do `recvbuf` kořene, ostatní ho ignorují

Operace (`MPI_OP`) buď předdefinovaná (`MIP_SUM, MPI_MAX,...`) nebo uživatelsky definovaná

- `MPI_MAXLOC` : Vypočte maximum a jeho index (číslo prvního procesu obsahujícího maximum)

Varianty `MPI_Alreduce` , `MPI_Reduce_scatter_block` , `MPI_Reduce_scatter` : všechny procesy v `comm` obdrží výsledek do `recvbuf`

Paralelní prefixový součet (scan)

Vstupy:

- Pole $X[0,...,n-1]$ z množiny D
- Asociativní a komutativní binární operace $\oplus$ v D

Výstup:

- Pole Y všech prefixových součtů pole X: $Y[i] = X[0] \oplus X[1] \oplus ... \oplus X[i]$

Sekvenční algoritmus:

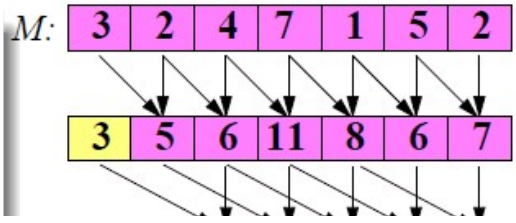
```
i = 0
suma = X[i]
Y[i] = suma
while i < n-1:
    i = i+1
    suma = operace(suma, X[i])
    Y[i] = suma
```

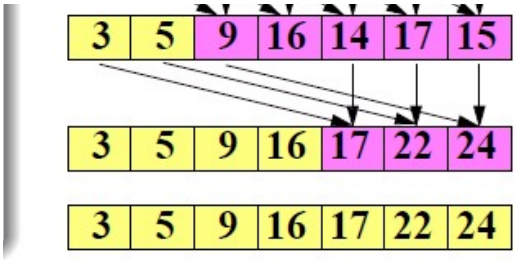
Od sekvenční redukce se liší pouze zapsáním mezivýsledku do Y

PPS na EREW PRAM

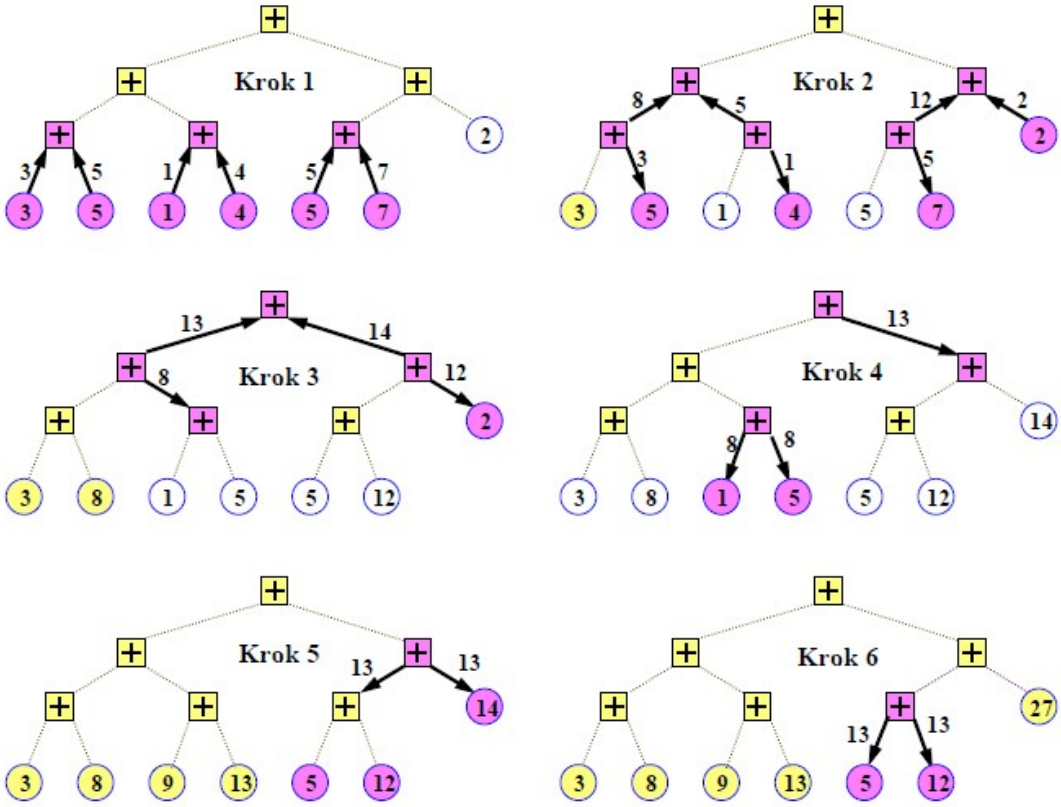
n vláken počítá in-place nad vstupním sdíleným polem $M[0,...,n-1]$
Každé vlákno π_i má privátní proměnnou y_i

```
for i in 0 ... n-1 do_parallel:
    y_i = X[i]
    M[i] = y[i]
for j in 0 ... log(n)-1 do_seq:
    for i in 2^j ... n-1 do_parallel:
        y_i = y_i + M[i-2^j]
    for i in 2^j ... n-1 do_parallel:
        M[i] = y_i
```





Nepřímý strom: Vstupní data pouze v listech, vnitřní uzly pouze počítají



PPS n vstupních hodnot lze řešit na **libovolném souvislém** n**-uzlovém řídkém grafu** G v $O(\mathrm{diam}(G))$ krocích

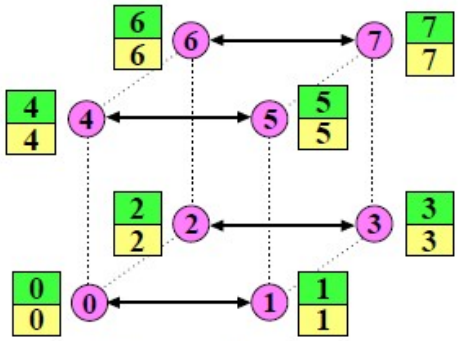
PPS na hyperkrychli:

V **lexikografickém** pořadí

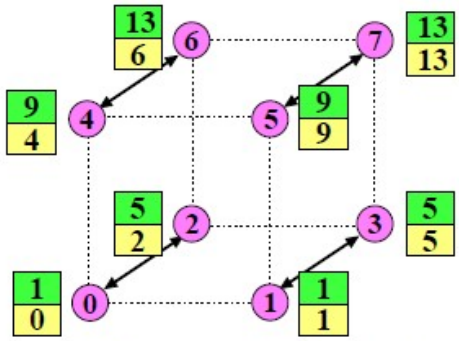
Jednoduché rozšíření AAB na Q_n
Každý procesor má $P_i, i = 0, ..., 2^r - 1$ má:

- 2 lokální proměnné: `zluty_i` a `zeleny_i`
- Na počátku `zluty_i = zeleny_i = X[i]`
- Na konci `zluty_i = Y[i]` , `zeleny_i = Y[n-1]`

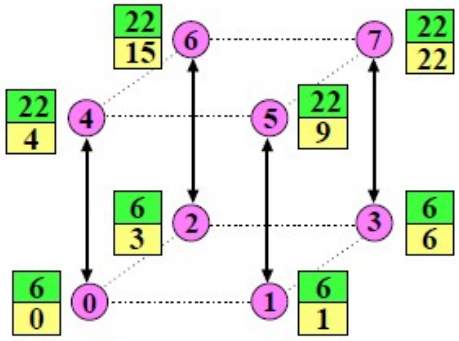
```
for i in 0 ... 2^r-1 do_parallel:
    # Inicializace proměnné každého procesoru
    zluty_i = zeleny_i = X[i]
    for j in 0 ... r-1 do_seq:
        send(zeleny_i, P_i XOR 2^j)
        receive(novyzeleny, P_i XOR 2^j)
        zeleny_i +=  novyzeleny
        if i XOR 2^j < i:
            zluty_i += novyzeleny
```



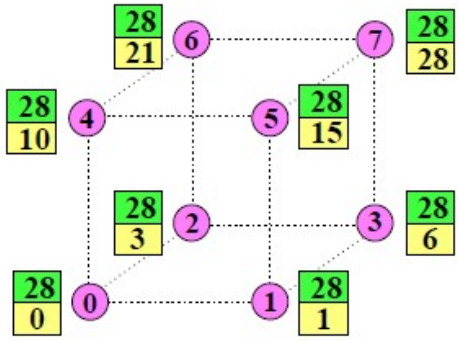
(a) Pocatecni stav



(b) Po provedeni kroku 1



(c) Po provedeni kroku 2

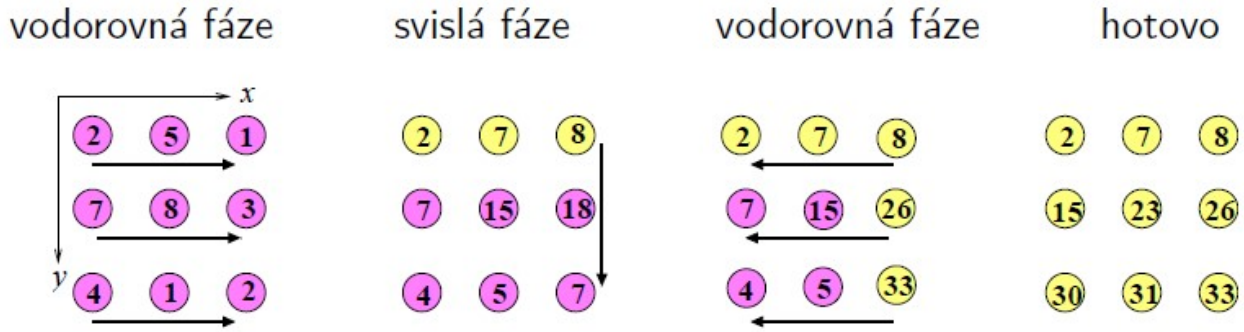


(d) Po provedeni kroku 3

PPS na SF mřížkách

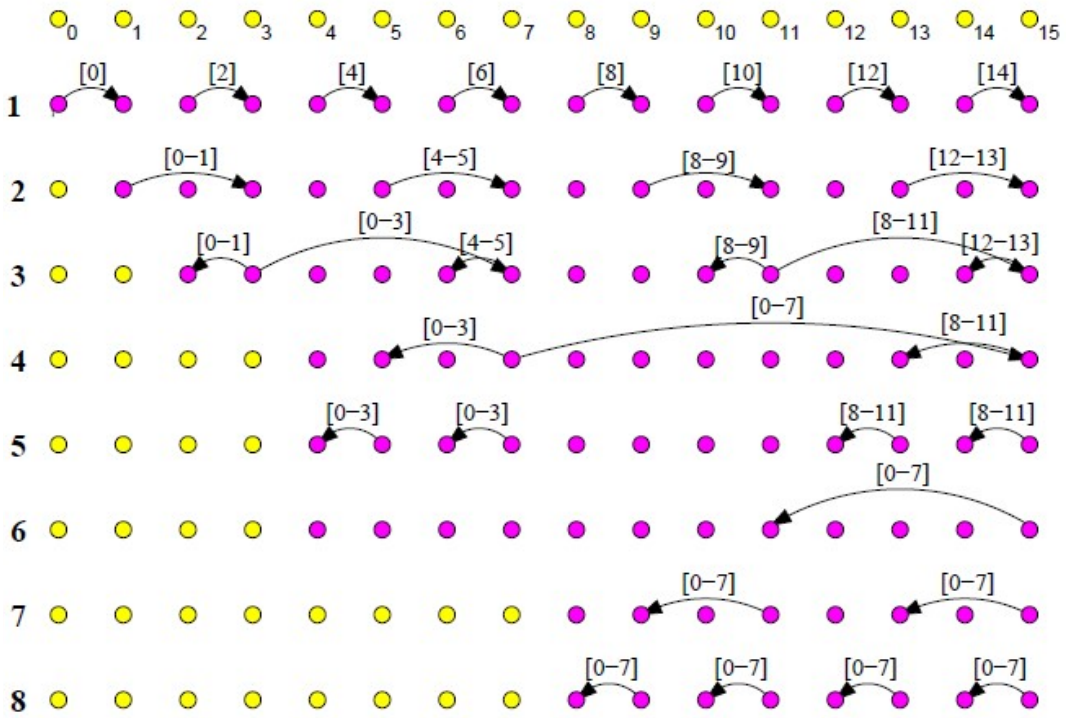
U vícerozměrných mřížek nejprve nutná **linearizace**

Vstupní pole namapované na 2D mřížku po řádcích shora dolů



PPS na WH mřížkách

Simulace PPS na nepřímém binárním stromu



PPS škálovatelnost

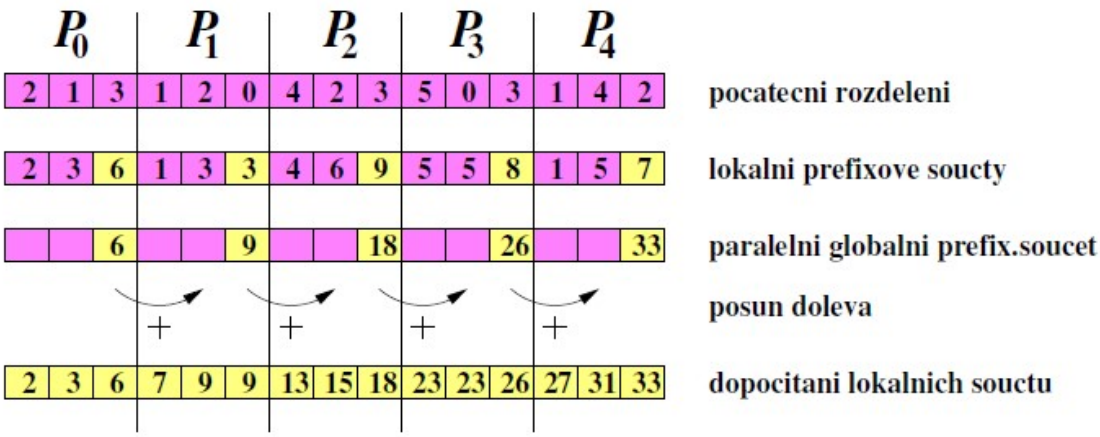
Vstupní pole velikosti n pro p procesorů rozděleno na subpole $X_0, ..., X_{p-1}$

P_i sekvenčně vypočte PPS $S_i = [s_{i,0}, ..., s_{i,q-1}]$ nad X_i. Nastaví proměnnou $z_i = s_{i,q-1}$ (jeho nejvyšší výsledek).

Všechny procesory provedou PPS nad $Z = [z_0, ..., z_{p-1}]$ a vygenerují $[\sigma_0, ..., \sigma_{p-1}]$

Pro každý $P_i, i < p - 1$: P_i paralelně pošle svou σ_i procesoru P_{i+1}.

Pro každý $P_i, i < p$: Procesor P_i paralelně připočte σ_{i-1} ke všem prvkům S_i



Stejná škálovatelnost jako paralelní redukce

Normální hyperkubický algoritmus

Segmentovaný PPS

Vstup: Pole rozdělené libovolně do segmentů

Výstup: Prefixové součty **uvnitř těchto segmentů**

$$\left|\ 2\ \ 1\ \ 3\ \ 5\ \right|\ 2\ \ 7\ \ 3\ \left|\ 9\ \ 4\ \ 5\ \ 6\ \right|\ 2\ \ 8\ \ 4\ \ 3\ \ 1\ \left|\right.$$

Provedením 4 izolovaných paralelních prefixových součtů s operací $+$ získáme

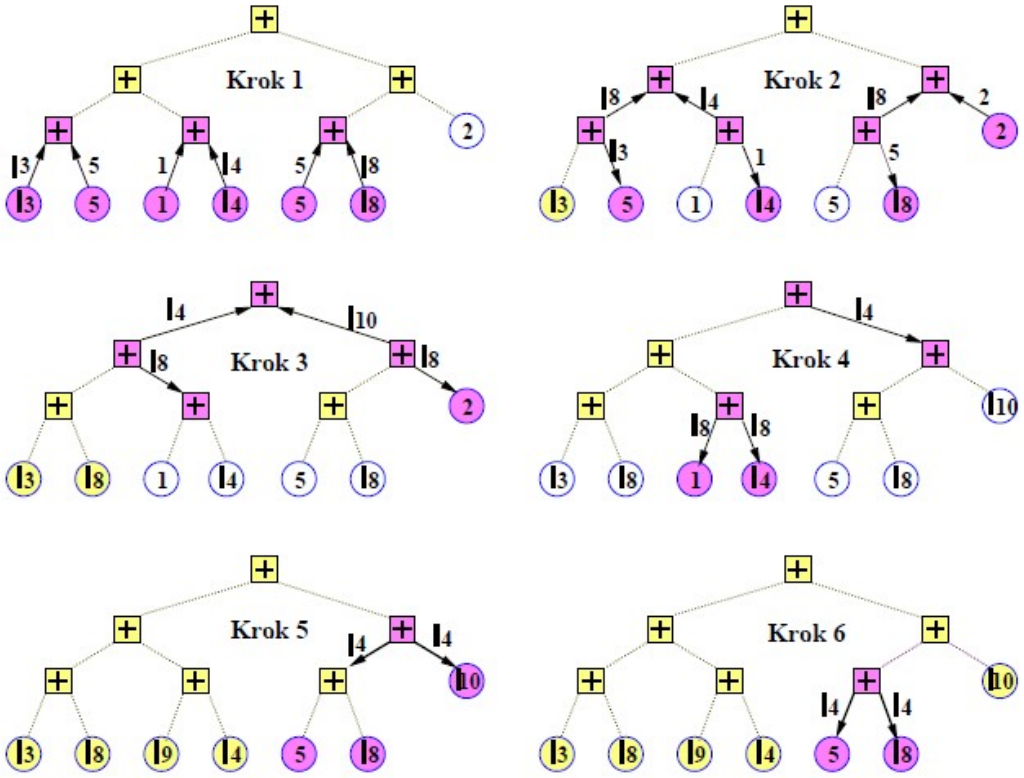
$$\left|\ 2\ \ 3\ \ 6\ \ 11\ \right|\ 2\ \ 9\ \ 12\ \left|\ 9\ \ 13\ \ 18\ \ 24\ \right|\ 2\ \ 10\ \ 14\ \ 17\ \ 18\ \left|\right.$$

Hlavní myšlenka: 1 globální PPS s modifikovanou operací $\overline{\oplus}$:

- $a \overline{\oplus} b = a \oplus b$ *(sčítání uvnitř segmentu)*
- $|a \overline{\oplus} b = |(a \oplus b)$ *(sčítání na levém kraji segmentu)*
- $a \overline{\oplus} |b = |b$ *(konec segmentu + začátek jiného)*
- $|a \overline{\oplus} |b = |b$ *(začátek + začátek jiného)*

Příklad na nepřímém stromu:

Vstupní pole: | 3 5 1 | 4 5 | 8 2 |



Sken v MPI

`MPI_Scan(void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype, MPI_OP op, MPI_COMM comm)` (standardní)
`MPI_Exscan` (exkluzivní -- proces P_i obsahuje součet nad $P_0, ..., P_{i-1}$)