Přednáška #12: **Paralelní algoritmy pro lineární algebru**

Základní definice

- $(m \times n)$-matice:

$$\mathcal{A} = (a_{ij}) = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}$$

- $\mathcal{A}$ je čtvercová, je-li $n = m$, a jinak je obdélníková.
- Je-li $\mathcal{A}$ $(m \times n)$-matice, pak transpozice $\mathcal{A}^T$ je $(n \times m)$-matice

$$\mathcal{A}^T = (a_{ji}) = \begin{bmatrix} a_{11} & \dots & a_{m1} \\ \vdots & & \vdots \\ a_{1n} & \dots & a_{mn} \end{bmatrix}$$

- Sloupcový vektor $= (n \times 1)$-matice (implicitně)

$$\vec{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

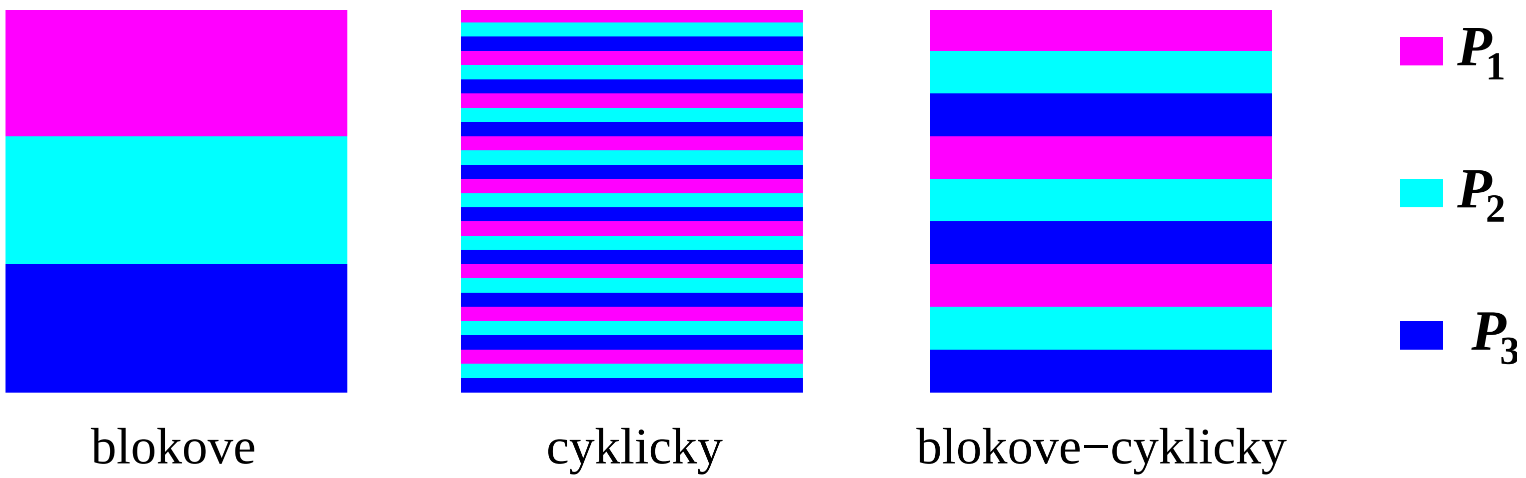
- Řádkový vektor $\vec{x}^T = [x_1, \dots, x_n] = (1 \times n)$-matice. skalární součin vektorů

Základní schémata mapování matic na procesory

Proužkové mapování

- Po řádcích nebo po sloupcích.
- Blokově, cyklicky, nebo blokově-cyklicky.

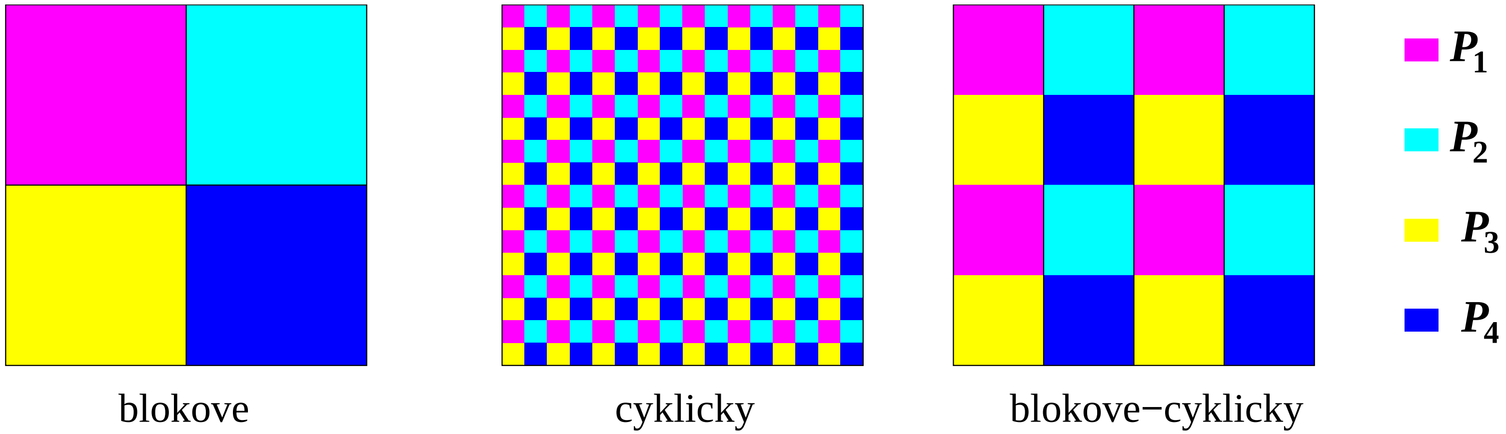
Příklad: Proužkové mapování (27×27)-matice po řádcích na $p = 3$ procesory P_1, P_2, P_3.



- Procesory tvoří virtuální 1-D mřížku $M(p)$.
- p nedělí n $\Longrightarrow$ spodní proužky jsou užší.
- Všechna 3 mapování jsou stejnoměrná.
- $\frac{n}{p} \geq p$ $\Longrightarrow$ blokově $\leftrightarrow$ cyklicky $\equiv$ AAS.

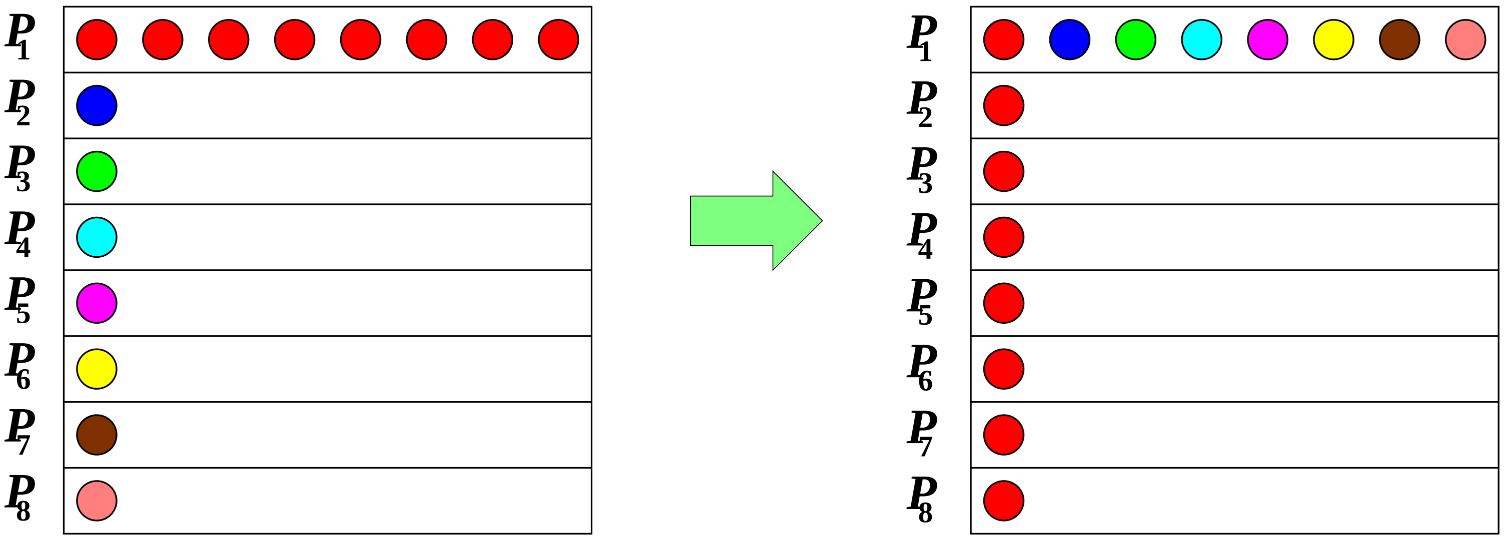
Šachovnicové mapování

- Procesory tvoří virtuální 2-D mřížku $M(\sqrt{p}, \sqrt{p})$.



Šachovnicové mapování (16×16)-matice na $p = 2 \times 2$ procesory $P_1 - P_4$.

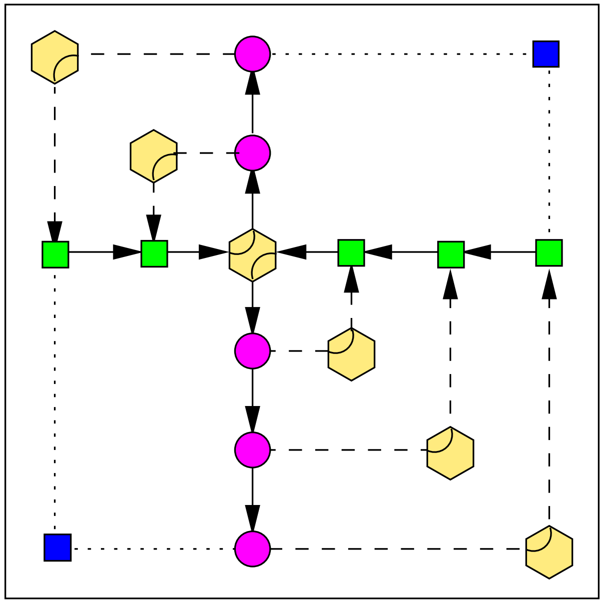
Transpozice matice mapované proužkově = AAS



Transpozice matice mapované šachovnicově = permutace

SF 2-D mřížka

SF všeportová $M(\sqrt{p}, \sqrt{p})$ s XY směrováním, $\mathcal{A} = (n \times n)$-matice, $N = n^2$



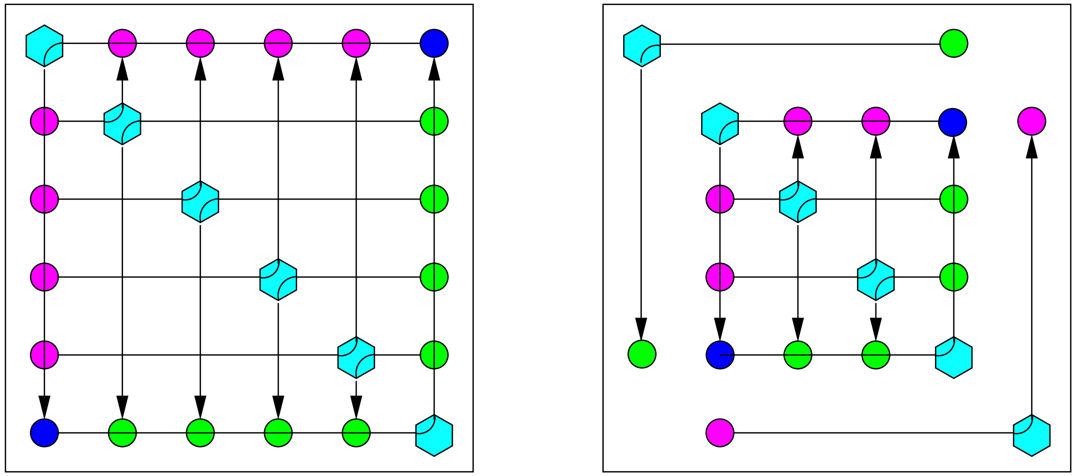
Časová složitost a škálovatelnost

$$T(N,p) = t_s + 2(\sqrt{p} - 1)\frac{N}{p}t_m + O\left(\frac{N}{p}\right).$$

$$E(N,p) = \Theta\left(\frac{1}{\sqrt{p}}\right) \implies \psi_2(N) = 1!!!$$

WH 2-D mřížka

- WH všeportová $M(\sqrt{p},\sqrt{p})$ s XY směrováním, $\mathcal{A}=(n\times n)$-matice, $N=n^2$.
- Blokově šachovnicově mapování $\Longrightarrow$ $(\frac{n}{\sqrt{p}}\times\frac{n}{\sqrt{p}})$-submatice na 1 procesor.



První 2 kroky transpozice matice na WH 2-D mřížce.

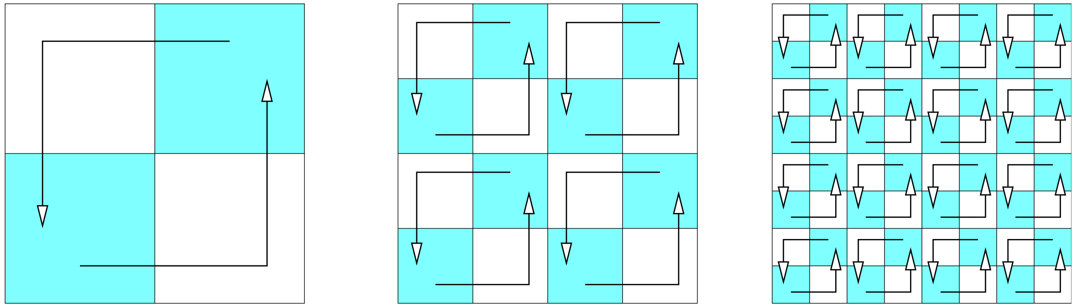
Časová složitost a škálovatelnost

$$T(N,p)\doteq(\sqrt{p}-1)(t_s+\sqrt{p}t_d+\frac{N}{p}t_m)+O\left(\frac{N}{p}\right).$$

$$E(N,p)=\Theta\left(\frac{1}{\sqrt{p}}\right)\quad\Longrightarrow\quad\psi_2(N)=1!!!$$

Hyperkrychle

- $\mathcal{A} = (n \times n)$-matice, $N = n^2$, poloduplexní SF jednoportová Q_q s e-cube směrováním.
- Rekurzivní algoritmus.
  - Fyzická hyperkrychle je vnořena do virtuální 2-D mřížky $M(\sqrt{p}, \sqrt{p})$, kde $p = 2^q$.
  - 1 krok = výměna posunem mezi diagonálně symetr. čtvrtinami matice (submatic).
  - Rekurze skončí na Q_2 = čtverec 4 procesorů



krok 1 krok 2 krok 3

Časová složitost a škálovatelnost

$$T(N,p) = \left(t_s + 2\frac{N}{p}t_m\right)\frac{q}{2} + O\left(\frac{N}{p}\right) = \left(\frac{t_s}{2} + \frac{N}{p}t_m\right)\log p + O\left(\frac{N}{p}\right).$$

$$E(N,p) = \Theta(1/\log p) \implies \psi_2(N) = 1!!!$$

- WH Q_q: přibližně totéž.

Násobení matice vektorem $\vec{y} = \mathcal{A}\vec{x}$ (MVM)

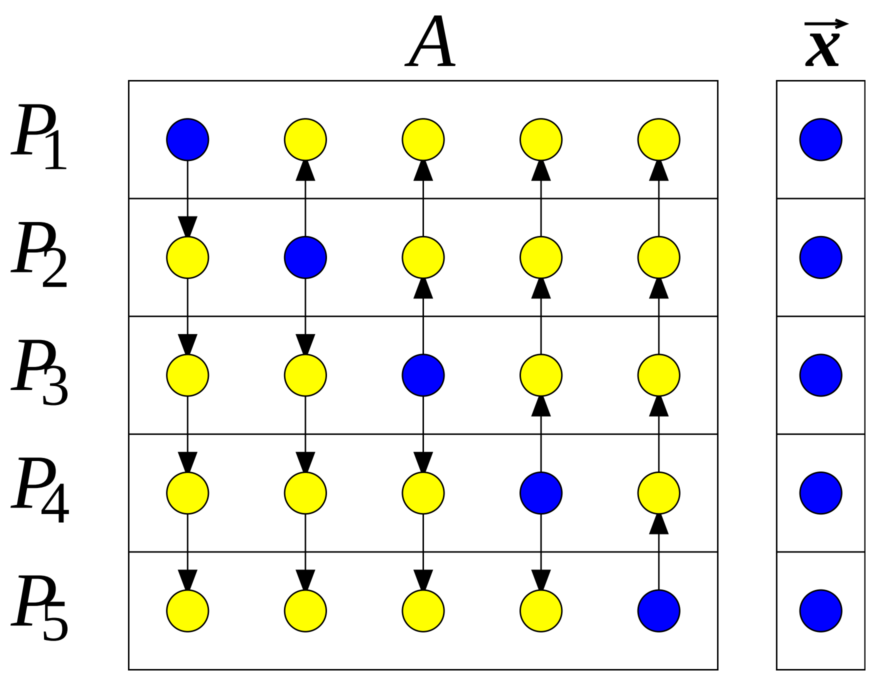
MVM: mapování po řádcích

- $\mathcal{A} = (n \times n)$-matice a $\vec{x}$ a $\vec{y} = (n \times 1)$-vektory mapované po řádcích na virtuální 1-D mřížku procesorů $M(p)$. Nechť $N = n^2$ a $r = \frac{n}{p}$.

Algoritmus $\text{RowWiseMVM}(\mathcal{A}, \overrightarrow{x}, \overrightarrow{y})$

Fáze 1: Každý P_i pošle svůj subvektor $\vec{x}_{(i-1)r+1}, \ldots, x_{ir}$ všem ostatním procesorům (AAB).

Fáze 2: Každý P_i vypočte svůj subvektor $\vec{y}_{(i-1)r+1}, \ldots, x_{ir}$ sekvenčním provedením r skalárních součinů n-vektorů.



Analýza složitosti a škálovatelnosti algoritmu RowWiseMVM

- Složitost fáze 2: $T_2(N,p) = k_2\frac{N}{p}$ (r skalárních součinů vektorů o délce n).
- Složitost fáze 1: (AAB r čísel) závisí na topologii a HW propojovací sítě.

1. Plně-duplexní 2-portová SF $M(p)$ a nekombinující AAB:
   $T_1(N,p) = p(t_s + k_1 r) \doteq k_1\sqrt{N}$.
2. Plně-duplexní 1-portová SF $M(\sqrt{p},\sqrt{p})$ a kombinující AAB: AAB po dimenzích
   $T_1(N,p) = \sqrt{p}t_s + k_1\sqrt{p}r + \sqrt{p}t_s + k_1\sqrt{p}\sqrt{p}r \doteq k_1\sqrt{N}$.
3. Plně-duplexní 4-portová SF $M(\sqrt{p},\sqrt{p})$ a nekombinující AAB: TADT metoda
   $T_1(N,p) = \frac{p}{4}(t_s + k_1 r) \doteq \frac{k_1\sqrt{N}}{4}$.
4. Hyperkrychle: podobně.

- Ve všech uvedených případech je $T(N,p) = T_1(N,p) + T_2(N,p) \doteq k_1'\sqrt{N} + k_2\frac{N}{p}$.
- Čili $E(N,p) = \frac{k_2 N}{k_2 N + k_1' p\sqrt{N}} \geq E_0$ pro $\sqrt{N} \geq \frac{E_0 k_1'}{(1-E_0)k_2}p$ a $p \leq \frac{(1-E_0)k_2}{E_0 k_1'}\sqrt{N}$

  $\Longrightarrow$ vynikající škálovatelnost:
  konstantní efektivnost lze dosáhnout i pro konstantní počet řádků $\mathcal{A}$ na 1 procesor!!!

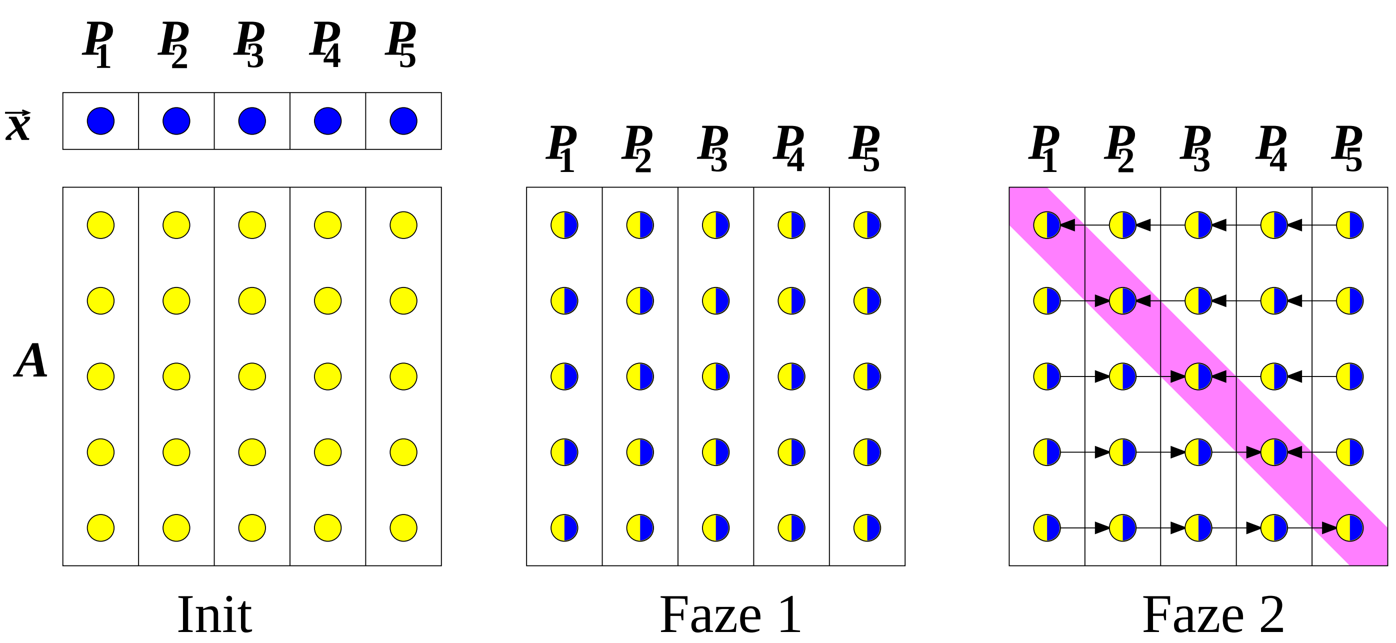
MVM: mapování po sloupcích

- $\mathcal{A} = (n \times n)$-matice, $\vec{x} = (n \times 1)$-vektor, virtuální 1-D mřížka $M(p)$ procesorů.
- Na počátku: P_i má **subvektor** $x_{(i-1)r+1}, \ldots, x_{ir}$ a **sloupce** $(i-1)r+1, \ldots, ir$ matice $\mathcal{A}$.
- Na konci: P_i má **subvektor** $y_{(i-1)r+1}, \ldots, x_{ir}$.

Algoritmus ColumnWiseMVM$(\mathcal{A}, \overrightarrow{x}, \overrightarrow{y})$

Fáze 1: Každý P_i může okamžitě spočítat svůj příspěvek k $\vec{y}$.

Fáze 2: Všechny procesory zredukují všech p polí částečných skalárních součinů provedením p redukcí s operací $+$ paralelně (redukce všech-se-všemi).



- Řádově stejná časová složitost i škálovatelnost jako v předchozím případě.

MVM: šachovnicové mapování

- $\mathcal{A} = (n \times n)$-matice šachovnicově map. na virtuální $M(\sqrt{p}, \sqrt{p})$, $N = n^2$.
- $\vec{x}, \vec{y} = (n \times 1)$-vektory mapované na poslední sloupec mřížky $M(\sqrt{p}, \sqrt{p})$

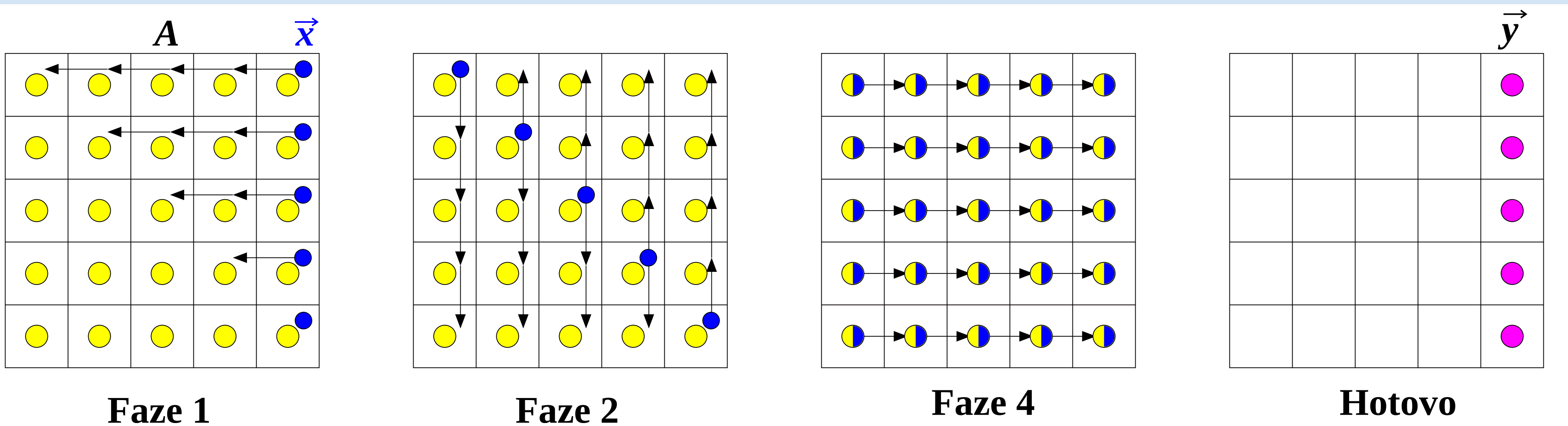
Algoritmus CheckerBoardMVM$(\mathcal{A}, \overrightarrow{x}, \overrightarrow{y})$

Fáze 1: **for all** $i = 1, \ldots, \sqrt{p}$ **do_in_parallel**
pravý krajní procesor $P_{i,\sqrt{p}}$ pošle svůj subvektor $\vec{x}$ diagonálnímu $P_{i,i}$.

Fáze 2: **for all** $i = 1, \ldots, \sqrt{p}$ **do_in_parallel**
$P_{i,i}$ informuje o obdrženém subvektoru $\vec{x}$ svůj sloupec (OAB).

Fáze 3: **for all** $i, j = 1, \ldots, \sqrt{p}$ **do_in_parallel**
$P_{i,j}$ vynásobí lokálně submatici $\mathcal{A}$ a subvektor $\vec{x}$.

Fáze 4: **for all** $i = 1, \ldots, \sqrt{p}$ **do_in_parallel**
procesory $P_{i,*}$ v řádku i provedou paralelní redukci,
kde kořenem redukčního stromu je pravý krajní procesor $P_{i,\sqrt{p}}$.

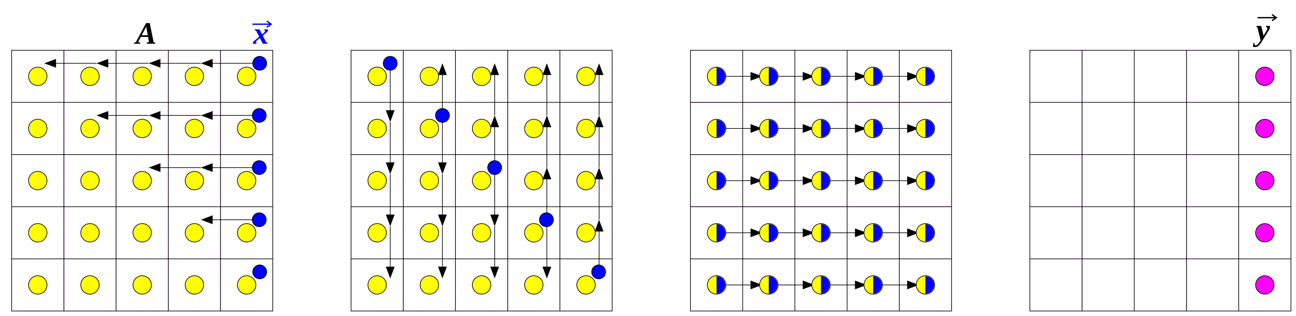


Analýza složitosti a škálovatelnosti algoritmu CheckerBoardMVM

- Každý procesor má přiřazenou $(\sqrt{\frac{N}{p}} \times \sqrt{\frac{N}{p}})$-submatici.
- Fáze 3: $T_3 = k_3\frac{N}{p}$ paralelních aritmetických operací.
- Složitost fáze 1 je řádově stejná (SF) nebo nižší (WH) než fáze 2 (tudíž lze zanedbat).
- Složitost fáze 2 $\doteq$ řádová složitost fáze 4.
- Fáze 2 = OAB $\sqrt{\frac{N}{p}}$ čísel: opět závisí na topologii a HW propojovací sítě
  - SF mřížka $M(\sqrt{p}, \sqrt{p})$: $T_2 = k_2 2\sqrt{p}\sqrt{\frac{N}{p}} = 2k_2\sqrt{N} \quad \Longrightarrow$ stejného řádu jako u algoritmu RowWiseMVM, čili $E(N,p) \geq E_0$, jestliže $\frac{\sqrt{N}}{p} \geq$ konst.
  - $Q_{\log p}$: $T_2 = k_2 \log p\sqrt{\frac{N}{p}}$

$$\Longrightarrow \quad T(N,p) \doteq 2T_2(N,p) + T_3(N,p) = k_2' \log p\sqrt{\frac{N}{p}} + k_3\frac{N}{p}$$

$\Longrightarrow \quad E(N,p) \geq E_0 \quad \Longleftrightarrow \quad N \geq \alpha^2 p \log^2 p$ a $p \leq \frac{\beta^2 N}{4\log^2(\beta\sqrt{N})}$, kde $\alpha = \frac{1}{\beta} = \frac{E_0 k_2'}{(1-E_0)k_3}$.



Faze 1 Faze 2 Faze 4 Hotovo

Násobení matice-matice $\mathcal{C} = \mathcal{A}\mathcal{B}$ (MMM)

MMM: naivní algoritmus

- $\mathcal{A} = (m \times l)$-matice a $\mathcal{B} = (l \times n)$-matice $\implies$ $p = mln$ procesorů, $N = ml + ln$.

Algoritmus $\textsc{NaiveMMM}(\mathcal{A}, \mathcal{B}, \mathcal{C})$

Fáze 1: **for all** i, j, k **do_in_parallel**,
procesor $P_{i,j,k}$ vypočítá součin $a_{i,j}b_{j,k}$.

Fáze 2: **for all** i, k **do_in_parallel**,
procesory $P_{i,*,k}$ pomocí paralelní redukce s kořenem redukčního stromu v $P_{i,1,k}$ vypočtou $c_{i,k} = \sum_{j=1}^{l} a_{i,j}b_{j,k}$.

- Příklady časů: $O(l)$ na SF 3-D mřížce $M(m, l, n)$ a $O(\log l)$ na hyperkrychli $Q_{\log p}$ nebo na WH 3-D mřížce, a proto
$E(N, p) = \Theta(1/\sqrt[3]{p})$ na 3-D mřížce a $E(N, p) = \Theta(1/\log p)$ na hyperkrychli.
- Škálování: $p' = m'l'n' < p$ & blokové mapování matic na $M(m', l', n')$.
- Počáteční rozeslání matic: nejvýše stejná složitost jako má fáze 2.

MMM: méně naivní algoritmus

- $(n \times n)$-matice $\mathcal{C} = \mathcal{AB}$ na $M(\sqrt{p}, \sqrt{p})$, $\sqrt{p} \leq n$: blokově šachovnicové mapování

Algoritmus $\textsc{Standard}\mathrm{MMM}(\mathcal{A}, \mathcal{B}, \mathcal{C})$

Fáze 1: **for all** $i = 1, \ldots, \sqrt{p}$ **do_in_parallel**,
procesory $P_{i,*}$ v i-tém řádku virtuální mřížky provedou operaci AAB,
ve které $P_{i,j}$ vysílá svou submatici $\mathcal{A}_{i,j}$.

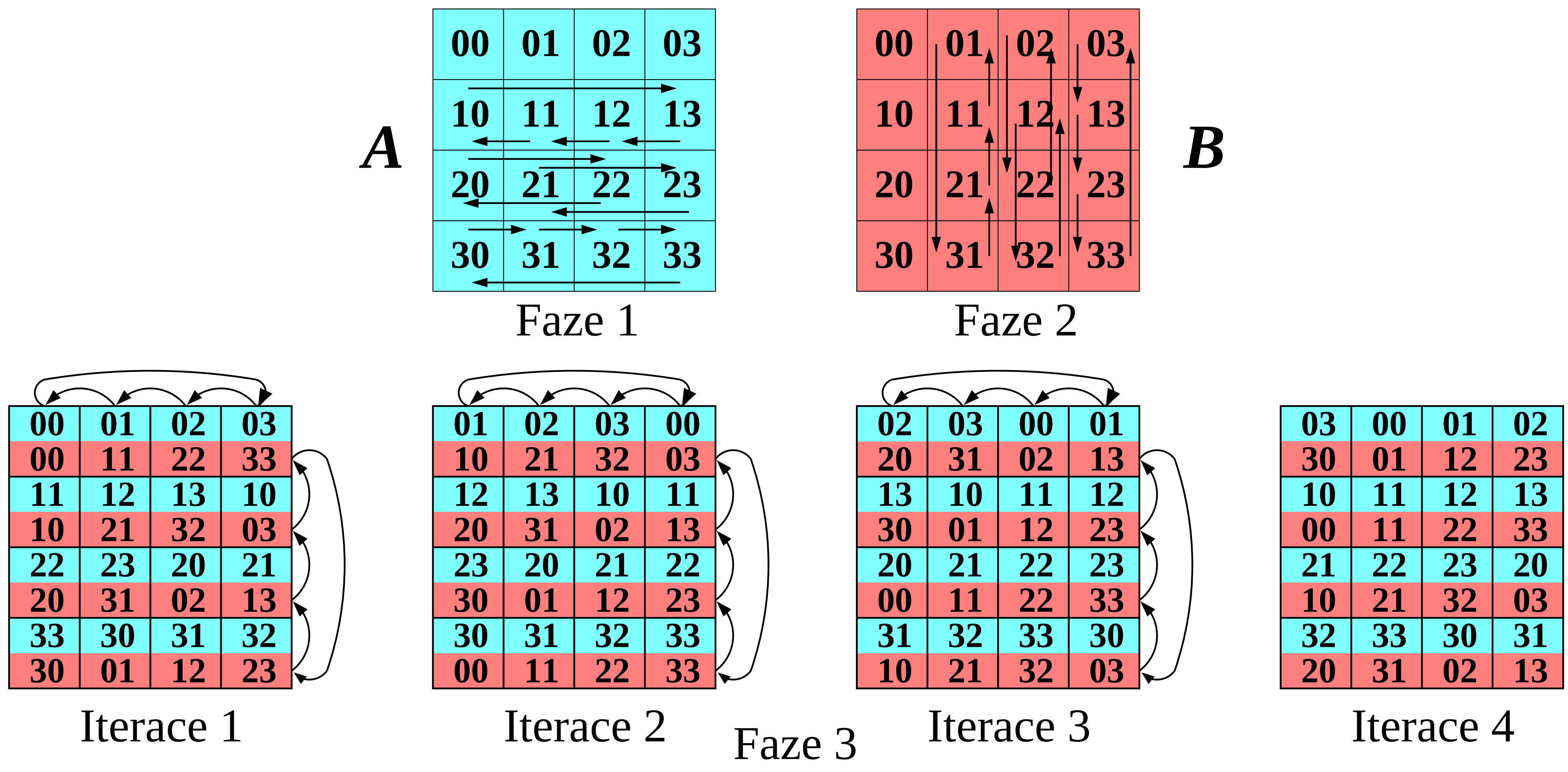
Fáze 2: **for all** $k = 1, \ldots, \sqrt{p}$ **do_in_parallel**,
procesory $P_{*,k}$ v k-tém sloupci virtuální mřížky provedou operaci AAB,
ve které $P_{j,k}$ vysílá svou submatici $\mathcal{B}_{j,k}$.

Fáze 3: **for all** $i, k = 1, \ldots, \sqrt{p}$ **do_in_parallel**,
$P_{i,k}$ vypočítá $\mathcal{C}_{i,k} = \sum_{j=1}^{\sqrt{p}} \mathcal{A}_{i,j}\mathcal{B}_{j,k}$.

- Předpokládáme-li SF 2-D mřížku a $N = n^2$, pak
$T(N,p) = O\left(\frac{N}{p}\sqrt{p} + \frac{N}{p}\sqrt{N}\right)$, a proto $E(N,p) = \Theta\left(\frac{\sqrt{N}}{\sqrt{N}+\sqrt{p}}\right)$.
$\Longrightarrow \quad \psi_2(N) = N$, což značí ideální škálovatelnost.

- Paměťově neefektivní: potřebuje celkem $\sqrt{p}\times$ více paměti než sekvenční alg.

MMM: Cannonův algoritmus



Cannonův algoritmus násobení matic na $T(4,4)$.

MMM: Cannonův algoritmus (pokr).

Algoritmus $\textsc{Cannon}\mathrm{MMM}(\mathcal{A},\mathcal{B},\mathcal{C})$

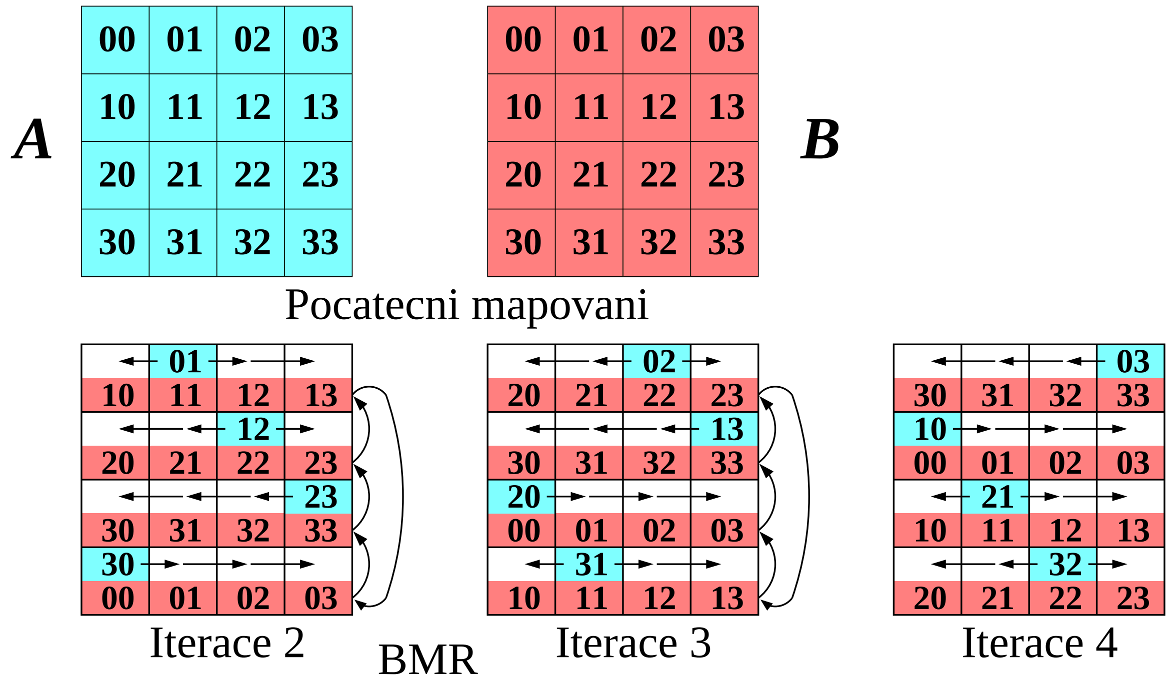
Fáze 1: **for all** $i = 1,\ldots,\sqrt{p}$ **do_in_parallel**
  všechny submatice $\mathcal{A}_{i,*}$ v řádku i se orotují o $i-1$ pozic doleva;
Fáze 2: **for all** $k = 1,\ldots,\sqrt{p}$ **do_in_parallel**
  všechny submatice $\mathcal{B}_{*,k}$ v sloupci k se orotují o $k-1$ pozic nahoru;
  $(*$ viz permutace cyklický posun v přednášce 9 $*)$
Fáze 3: **repeat** $\sqrt{p}$ **times**
  $\{$ **for all** $i,k = 1,\ldots,\sqrt{p}$ **do_in_parallel**
    $P_{i,k}$ vynásobí momentální submatice $\mathcal{A}$ a $\mathcal{B}$ a přičte výsledek do $\mathcal{C}_{i,k}$;
   for all $i = 1,\ldots,\sqrt{p}$ **do_in_parallel**
    všechny submatice $\mathcal{A}_{i,*}$ v řádku i se orotují o 1 pozici doleva;
   for all $k = 1,\ldots,\sqrt{p}$ **do_in_parallel**
    všechny submatice $\mathcal{B}_{*,k}$ v sloupci k se orotují o 1 pozici nahoru; $\}$

Časová složitost a škálovatelnost

Příklad: Všeportová WH hyperkrychle $Q_{\log p}$ $\Longrightarrow$

$$T_{\text{Cannon}}(N,p) \doteq O(t_s\sqrt{p}) + O\left(\frac{N}{\sqrt{p}}t_m\right) + O\left(\frac{n^3}{p}\right).$$

MMM: Foxův algoritmus $\equiv$ Broadcast-Multiply-Roll (BMR)



Algoritmus $\textsc{FoxMMM}(\mathcal{A}, \mathcal{B}, \mathcal{C})$

for all $j = 1, \ldots, \sqrt{p}$ **do_sequentially**

$\{$ **for all** $i = 1, \ldots, \sqrt{p}$ **do_in_parallel**

$P_{i,(i+j) \bmod \sqrt{p}}$ rozešle submatici $\mathcal{A}_{i,(i+j) \bmod \sqrt{p}}$ v rámci řádku i (OAB);

for all $i, k = 1, \ldots, \sqrt{p}$ **do_in_parallel**

$P_{i,k}$ přičte k $\mathcal{C}_{i,k}$ součin přijatých submatic $\mathcal{A}$ a $\mathcal{B}$;

for all $k = 1, \ldots, \sqrt{p}$ **do_in_parallel**

všechny submatice $\mathcal{B}_{*,k}$ v sloupci k se orotují o 1 pozici nahoru; $\}$

Časová složitost a škálovatelnost podobná.

Úlohy lineární algebry

- Řešení soustavy lineárních rovnic (SLR) s regulární čtvercovou $(n \times n)$-maticí $\mathcal{A}$ pro 1 nebo více pravých stran $\vec{b}$.
  - Metody řešení se liší podle typů matic: husté vs. řídké, dobře vs. špatně podmíněné, symetrické vs. nesymetrické, pozitivně definitní vs. indefinitní ap.
  - Základní skupiny metod jsou:
    * přímé metody (finitní): pro husté matice,
    * iterační: pro řídké matice,
    * gradientní: pro řídké matice.
    * speciální: pro speciální matice
- Výpočet inverzní matice.
- Výpočet determinantu.
- Hledání vlastních čísel a vektorů.

Speciální tridiagonální SLR a sudo-lichá redukce

$$
\begin{aligned}
(f_1 x_0 +)\, g_1 x_1 + h_1 x_2 \qquad\qquad &= b_1 \\
f_2 x_1 + g_2 x_2 + h_2 x_3 \qquad\qquad &= b_2 \\
\dots\dots\dots\dots \qquad\qquad & \\
f_i x_{i-1} + g_i x_i + h_i x_{i+1} \qquad\qquad &= b_i \\
\dots\dots\dots\dots \qquad\qquad & \\
f_n x_{n-1} + g_n x_n (+ h_n x_{n+1}) &= b_n
\end{aligned}
$$

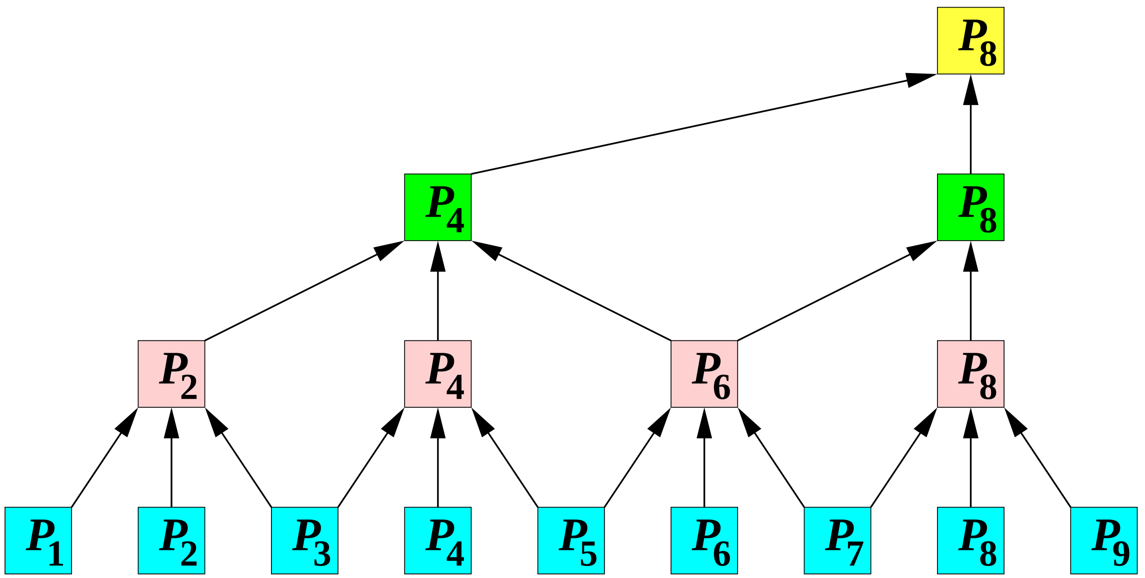
Myšlenka sudo-liché redukce:

- Jestliže $\forall i;\ g_i \neq 0$, pak
$$x_i = \frac{1}{g_i}(b_i - f_i x_{i-1} - h_i x_{i+1}).$$
- Substituuj tyto výrazy za x_i s lichým i
  $\Longrightarrow$ nový tridiagonální SLR s proměnnými $x_2, x_4, \ldots$ a s rovnicemi ve tvaru

$$-\left(\frac{f_i f_{i-1}}{g_{i-1}}\right) x_{i-2} + \left(g_i - \frac{h_{i-1} f_i}{g_{i-1}} - \frac{h_i f_{i+1}}{g_{i+1}}\right) x_i - \left(\frac{h_i h_{i+1}}{g_{i+1}}\right) x_{i+2} = b_i - \frac{f_i}{g_{i-1}} b_{i-1} - \frac{h_i}{g_{i+1}} b_{i+1}.$$

- Počet rovnic a neznámých proměnných klesnul na jednu polovinu.
- $SU(n) = O(n)$

Mapování po řádcích: paralelní implementace na virtuální 1-D mřížce



- Po k paralelních redukcích: P_{j2^k} komunikuje s $P_{(j+1)2^k}$.
- SF $M(n)$: celková komunikační složitost v $\lceil \log n \rceil$ krocích je $\Theta(n) = \Theta(SU(n))$.
- SF síť G: celková komunikační složitost $\approx \mathrm{diam}(G)$.
- SF Q_n: BRGC $\implies$ konstantní komunikační složitost/krok: $\varrho(G_n(j2^k), G_n((j+1)2^k)) \leq 2$!!!
- WH $M(n)$: konstantní komunikační složitost/krok.
- Škálovatelnost: řádkově blokové mapování na hyperkrychli n. WH síti $\implies$ $T(n,p) = O(n/p + n/(2p) + n/(4p) + \cdots + n/(2^{\log k} p) + 1 + \ldots + 1)$, kde $k = \log(n/p)$ $\implies$ $T(n,p) = O(2n/p + \log p)$ $\implies$ dobrá škálovatelnost, jako u PPS algoritmu.

Definice 1. *Elementární řádkové operace (EŘO):*

1. *prohození 2 řádků matice,*
2. *vynásobení/vydělení řádku konstantou,*
3. *přičtení násobku 1 řádku k jinému řádku.*

Gaussova eliminace při řešení $\mathcal{A}\vec{x} = \vec{b}$

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

- Vytvoříme rozšířenou matici $\mathcal{A}'$

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} & b_1 \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} & b_n \end{bmatrix}$$

- Postupnou aplikací EŘO transformujeme $\mathcal{A}'$ na tvar

$$\begin{bmatrix} a'_{1,1} & a'_{1,2} & \dots & a'_{1,n} & b'_1 \\ 0 & a'_{2,2} & \dots & a'_{2,n} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & a'_{n,n} & b'_n \end{bmatrix}$$

- Provedením **zpětné substituce**

$$x_i = \frac{1}{a'_{i,i}} \left(b'_i - \sum_{j=i+1}^{n} a'_{i,j} x_j \right)$$

vypočteme postupně $x_n, x_{n-1}, \dots, x_1$.

Algoritmus stabilní Gaussovy eliminace

Algoritmus $\textsc{GaussElim}(\mathcal{A}, \overrightarrow{x}, \overrightarrow{b})$

for $i := 0, \ldots, n-1$ **do_sequentially**

(1) **if** $(a_{i,i} = 0 \quad \& \quad \forall j > i; a_{j,i} = 0)$ **then** *Exit(Neřešitelné)*

(2) **if** $(a_{i,i} = 0 \quad \& \quad \exists j > i; a_{j,i} \neq 0)$ **then**

vyber **hlavní prvek** (*pivot*) ($*$ nechť je na řádku j $*$)

prohoď řádek i a řádek j ($*$ nyní $a_{i,i} \neq 0$ $*$)

(3) vyděl řádek i prvkem $a_{i,i}$ ($*$ $n-i$ operací dělení $*$)

(4) vynuluj sloupec i pod diagonálou odčítáním násobků řádku i od řádků $i+1, \ldots, n$.

($*$ přibližně $(n-i)^2$ operací násobení a odčítání $*$)

Sekvenční složitost Gaussovy eliminace

- Celkový počet operací dělení na ř. (3) je přibližně $\sum_{i=1}^{n-1}(n-i) \doteq \frac{n^2}{2}$.
- Celkový počet operací násobení a odčítání na ř. (4) je přibližně $\sum_{i=1}^{n-1}(n-i)^2 \doteq \frac{n^3}{3}$.
- Celková sekvenční složitost $SU(n) \doteq \frac{2n^3}{3}t_{\mathrm{mul}} + \frac{n^2}{2}t_{\mathrm{div}}$ za předpokladu, že $t_{\mathrm{mul}} \doteq t_{\mathrm{sub}}$.

Mapování po řádcích: paralelní implementace na virtuální 1-D mřížce $M(n)$

- $p = n$: Každý procesor vlastní 1 řádek
- Paralelní operace prováděné v 1 iteraci:
  - Krok (1): paralelní redukce + OAB 1 čísla
  - Krok (2): 1-1 komunikace n čísel
  - Krok (3): lokální provedení přibližně $n - i$ aritm. operací (žádná komunikace)
  - Krok (4): OAB řádku i + lokální provedení přibližně $2(n - i)$ aritm. operací
- Celková // složitost:
  - Dominují kroky (3)+(4).
  - Paralelní aritm. operace: $3\sum_{i=1}^{n-1}(n-i) \doteq \frac{3n^2}{2}$.
  - OAB v (4) závisí na kom. modelu. Např. WH mřížka: $\sum_{i=1}^{n-1}\left((t_s + (n-i)t_m)\log n\right)$.

    Pak: $T(n^2, n) \doteq \frac{3n^2}{2} + t_s n \log n + \frac{1}{2} t_m n^2 \log n$
- Závěr: není efektivní: komunikační režie převyšuje výpočetní složitost
- Vysvětlení: sekvenční závislost iterací: iterace $i + 1$ začne, až když skončí iterace i
- Řešení: **systolický** (data-flow přístup), pokud řešení **nevyžaduje pivotizaci** (čili provádí se pouze kroky (3) a (4)) !!!!

Systolická (asynchronní, vlnová) GE na virtuální 1-D mřížce $M(n)$

- 1 iterace $=$ 1 komunikační a výpočetní vlna. (V předchozím algoritmu se vlny nepřekrývaly.)
- Každý procesor P_i přijímá postupně řádky od P_{i-1}, přeposílá je P_{i+1} a současně je zapracovává (krok (4)).
- Jakmile dostane poslední $(i-1)$-tou řádku, nečeká, až celá iterace $i-1$ doběhne, ale jakmile provede krok (3), okamžitě spouští další vlnu.
- $T(n^2, n) = O(n^2)$ i na fyzické 1-D mřížce.
- $p < n$: blokově-řádkové mapování.

Gauss-Jordanova eliminace=metoda výpočtu inverzní matice

- Vytvoříme rozšířenou matici $\mathcal{A}' = [\mathcal{A}|\mathcal{I}_n]$

$$\left[\begin{array}{cccc|cccc} a_{1,1} & a_{1,2} & \dots & a_{1,n} & 1 & 0 & \dots & 0 \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} & 0 & 0 & \dots & 1 \end{array}\right]$$

- Postupnou aplikací EŘO transformujeme $\mathcal{A}'$ na tvar

$$\left[\begin{array}{cccc|cccc} 1 & 0 & \dots & 0 & b_{1,1} & b_{1,2} & \dots & b_{1,n} \\ 0 & 1 & \dots & 0 & b_{2,1} & b_{2,2} & \dots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & b_{n,1} & b_{n,2} & \dots & b_{n,n} \end{array}\right] = [\mathcal{I}_n|\mathcal{B}] = \left[\mathcal{I}_n|\mathcal{A}^{-1}\right]$$

- Paralelizace, její možnosti, efektivnosti a škálovatelnosti - naprosto analogické.

Trojúhelníkové matice a zpětná substituce

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ 0 & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

- $n =$ počet rovnic $\implies SU(n) = O(n^2)$.
- $p = n$ & $t_{\mathrm{OAB}}(p) = \Theta(p)$ (SF 1-D mřížka) $\implies T(n,n) = O(n^2)$.
- $p = n$ & $t_{\mathrm{OAB}}(p) = O(\log p)$ (hyperkrychle, WH mřížky) $\implies T(n,n) = O(n \log n)$.
- $p \ll n$ & cyklické mapování po řádcích $\implies$ n paralelních kroků $\implies$

$$T(n,p) = nO\left(\log p\right) + nO\left(\frac{n}{p}\right) = O(n \log p) + O\left(\frac{n^2}{p}\right).$$

- $p \ll n$ & blokové mapování po řádcích $\implies$ p paralelních kroků $\implies$

$$T(n,p) = pO\left(\frac{n^2}{p^2}\right) + pO\left(\frac{n}{p}\log p\right).$$

Přímá metoda pro řešení SLR s více pravými stranami

- $\mathcal{A}\overrightarrow{x} = \overrightarrow{b}$
  kde $\mathcal{A} = (n \times n)$-matice, $\overrightarrow{b} = (n \times 1)$-vektor, a $\vec{x} = (n \times 1)$-vektor n neznámých.
- (1) LU dekompozice: $\mathcal{A} = \mathcal{L}\mathcal{U}$,
  kde $\mathcal{L} =$ spodní trojúhelníková matice, $\mathcal{U} =$ horní trojúhelníková matice.
- (2) Dopředná redukce: $\mathcal{U}\overrightarrow{x} = \mathcal{L}^{-1}\overrightarrow{b} = \overrightarrow{y}$ (čili $\forall \overrightarrow{b}$ řešíme $\mathcal{L}\overrightarrow{y} = \overrightarrow{b}$).
- (3) Zpětná substituce: $\overrightarrow{x} = \mathcal{U}^{-1}\overrightarrow{y}$ (čili $\forall \overrightarrow{y}$ řešíme $\mathcal{U}\overrightarrow{x} = \overrightarrow{y}$).

LU dekompozice

- Gaussova LU dekompozice: $\mathrm{diag}(\mathcal{L}) = \overrightarrow{1}$.
- Choleskyho LU dekompozice: $\mathrm{diag}(\mathcal{L}) = \mathrm{diag}(\mathcal{U})$.

Idea paralelní LU dekompozice

$$\overset{A}{\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}} = \overset{L}{\begin{bmatrix} 1 & & & 0 \\ l_{21} & 1 & & \\ l_{31} & l_{32} & 1 & \\ l_{41} & l_{42} & l_{43} & 1 \end{bmatrix}} \overset{U}{\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ & u_{22} & u_{23} & u_{24} \\ 0 & & u_{33} & u_{34} \\ & & & u_{44} \end{bmatrix}} =$$

$$\left[\begin{array}{c:c:c:c} u_{11} & u_{12} & u_{13} & u_{14} \\ \hdashline l_{21}u_{11} & l_{21}u_{12}+u_{22} & l_{21}u_{13}+u_{23} & l_{21}u_{14}+u_{24}+ \\ \hdashline l_{31}u_{11} & l_{31}u_{12}+l_{32}u_{22} & l_{31}u_{13}+l_{32}u_{23}+u_{33} & l_{31}u_{14}+l_{32}u_{24}+u_{34} \\ \hdashline l_{41}u_{11} & l_{41}u_{12}+l_{42}u_{22} & l_{41}u_{13}+l_{42}u_{23}+l_{43}u_{33} & l_{41}u_{14}+l_{42}u_{24}+l_{43}u_{34}+u_{44} \end{array}\right]$$

Optimální mapování pro paralelní LU dekompozici

Blokově-cyklicky šachovnicové mapování $(n \times n)$-$\mathcal{A}$ na mřížce procesorů $M(r,r)$,

$r = \sqrt{p}$, $n = rsq$

$$
\left[\begin{array}{ccc|c|ccc}
\mathcal{A}_{1,1} & \cdots & \mathcal{A}_{1,r} & & \mathcal{A}_{1,(q-1)r+1} & \cdots & \mathcal{A}_{1,qr} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\mathcal{A}_{r,1} & \cdots & \mathcal{A}_{r,r} & & \mathcal{A}_{r,(q-1)r+1} & \cdots & \mathcal{A}_{r,qr} \\
\hline
 & \vdots & & \ddots & & \vdots & \\
\hline
\mathcal{A}_{(q-1)r+1,1} & \cdots & \mathcal{A}_{(q-1)r+1,r} & & \mathcal{A}_{(q-1)r+1,(q-1)r+1} & \cdots & \mathcal{A}_{(q-1)r+1,qr} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\mathcal{A}_{qr,1} & \cdots & \mathcal{A}_{qr,r} & & \mathcal{A}_{qr,(q-1)r+1} & \cdots & \mathcal{A}_{qr,qr}
\end{array}\right]
$$

Mapování 2-D mřížky procesorů $M(r,r)$ na $\mathcal{A}$

$$
\left[\begin{array}{ccc|c|ccc}
P_{1,1} & \cdots & P_{1,\sqrt{p}} & & P_{1,1} & \cdots & P_{1,\sqrt{p}} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
P_{\sqrt{p},1} & \cdots & P_{\sqrt{p},r} & & P_{\sqrt{p},1} & \cdots & P_{\sqrt{p},r} \\
\hline
 & \vdots & & \ddots & & \vdots & \\
\hline
P_{1,1} & \cdots & P_{1,\sqrt{p}} & & P_{1,1} & \cdots & P_{1,\sqrt{p}} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
P_{\sqrt{p},1} & \cdots & P_{\sqrt{p},r} & & P_{\sqrt{p},1} & \cdots & P_{\sqrt{p},r}
\end{array}\right]
$$

Implementace paralelní LU dekompozice

Algoritmus $\textsc{LUDecomposition}(\mathcal{A})$

for $k = 1, \ldots, qr$ **do_sequentially** $\{$

Fáze 1: $P_{\widetilde{k},\widetilde{k}}$ počítá $\mathcal{A}^{-1}_{k,k}$;

Fáze 2: **for** $j = k, \ldots, qr$ **do_in_parallel**

$P_{\widetilde{k},\widetilde{j}}$ vyšle $\mathcal{A}^{-1}_{k,k}$ dolů v sloupci $\widetilde{k}$, je-li $j = k$

$P_{\widetilde{k},\widetilde{j}}$ vyšle $\mathcal{A}_{k,j}$ dolů v sloupci $\widetilde{j}$ jinak;

Fáze 3: **for** $i = k+1, \ldots, qr$ **do_in_parallel**

$\{$ $P_{\widetilde{i},\widetilde{k}}$ počítá $\mathcal{A}_{i,k} = \mathcal{A}^{-1}_{k,k}\mathcal{A}_{i,k}$;

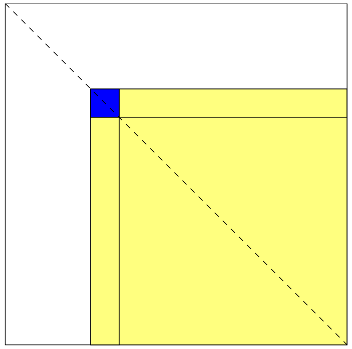
$P_{\widetilde{i},\widetilde{k}}$ vyšle $\mathcal{A}_{i,k}$ doprava v řádku $\widetilde{i}$ $\}$;

Fáze 4: **for** $i = k+1, \ldots, qr$

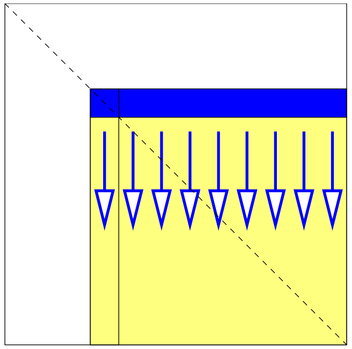
for $j = k+1, \ldots, qr$ **do_in_parallel**

$P_{\widetilde{i},\widetilde{j}}$ počítá $\mathcal{A}_{i,j} = \mathcal{A}_{i,j} - \mathcal{A}_{i,k}\mathcal{A}_{k,j}$

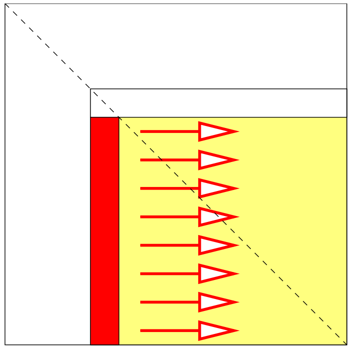
$\}$



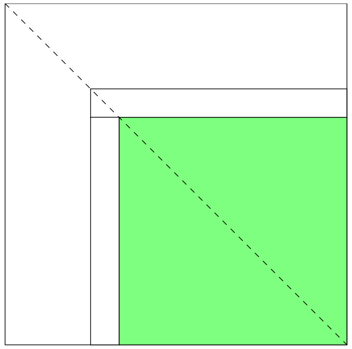
Faze 1



Faze 2



Faze 3



Faze 4

Sekvenční iterační metody pro řešení obecných SLR

- Jacobi

$$x_i(t+1) = -\frac{1}{a_{i,i}}\left[\Sigma_{j\neq i} a_{i,j} x_j(t) - b_i\right] \tag{1}$$

- Gauss-Seidel

$$x_i(t+1) = -\frac{1}{a_{i,i}}\left[\Sigma_{j<i} a_{i,j} x_j(t+1) + \Sigma_{j>i} a_{i,j} x_j(t) - b_i\right]$$

- Jacobiho superrelaxace (JOR)

$$x_i(t+1) = (1-\gamma)x_i(t) - \frac{\gamma}{a_{i,i}}\left[\Sigma_{j\neq i} a_{i,j} x_j(t) - b_i\right]$$

- Gauss-Seidelova superrelaxace (SOR)

$$x_i(t+1) = (1-\gamma)x_i(t) - \frac{\gamma}{a_{i,i}}\left[\Sigma_{j<i} a_{i,j} x_j(t+1) + \Sigma_{j>i} a_{i,j} x_j(t) - b_i\right]$$

Paralelní Jacobiho metoda

Pro účely vysvětlení paralelního řešení, přepíšeme rovnici (1) na tvar

$$x_i(t+1) = \frac{r_i(t)}{a_{i,i}} + x_i(t),$$

kde $\overrightarrow{r(t)} = \overrightarrow{b} - \mathcal{A}\overrightarrow{x(t)}$ = residuální (zbytkový) vektor.

Algoritmus $\textsc{ParallelJacobi}(\mathcal{A}, \overrightarrow{x}, \overrightarrow{b})$

$t := 0$; $\overrightarrow{x(t)} :=$ počáteční hodnota;
$\overrightarrow{r(t)} := \overrightarrow{b} - \mathcal{A}\overrightarrow{x(t)}$; (∗ počáteční residuál: násobení matice vektorem ∗)

while $(||\overrightarrow{r(t)}|| > \varepsilon)$ **do_sequentially** (∗ $||\overrightarrow{r(t)}|| = \sqrt{\overrightarrow{r(t)}^T . \overrightarrow{r(t)}}$ = skalární součin ∗)
  { $t := t + 1$; (∗ paralelní redukce + OAB ∗)
  for $i := 0, \ldots, n-1$ **do_in_parallel**
    $x_i(t) = r_i(t-1)/a_{i,i} + x_i(t-1)$; (∗ $r_i(t-1), x_i(t-1), a_{i,i}$ musí být v 1 procesoru ∗)
  bariérová synchronizace;
  $\overrightarrow{r(t)} = \overrightarrow{b} - \mathcal{A}\overrightarrow{x(t)}$; (∗ nový residuál: násobení matice vektorem ∗)
  bariérová synchronizace;}
$\overrightarrow{x} := \overrightarrow{x(t)}$;

Žádná datová závislost v rámci téže iterace $\Longrightarrow$ krásná paralelizovatelnost.

Paralelní Gauss-Seidelova metoda

- Sekvenční Gauss-Seidelova metoda je rychlejší než sekvenční Jacobiho.
- Je však inherentně sekvenční: Je-li $\mathcal{A}$ hustá nebo nepravidelně řídká matice, pak pro výpočet $x_i(t+1)$ potřebujeme znát hodnoty $x_j(t+1)$ pro všechny nebo skoro všechny $j < i$.
- Tuto nesnáz lze často obejít, je-li matice $\mathcal{A}$ je řídká a speciálního typu.
- Toto je případ většiny matic vzniklých při diskretizaci parciálních dif. rovnic.
- Diskretizaci lze provést metodou
  - konečných diferencí,
  - konečných prvků (FEM).

Parciální diferenciální rovnice (PDE)

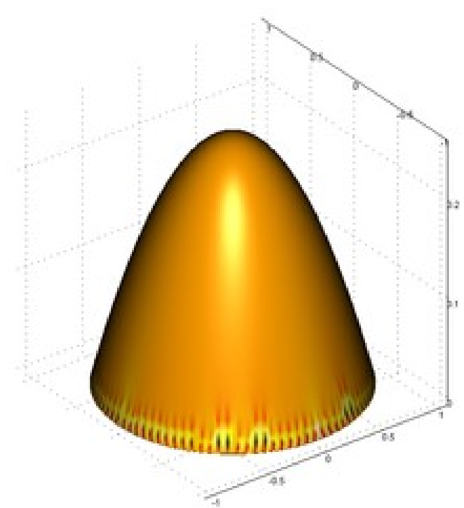
Poissonova rovnice = lineární parciální diferenciální rovnice

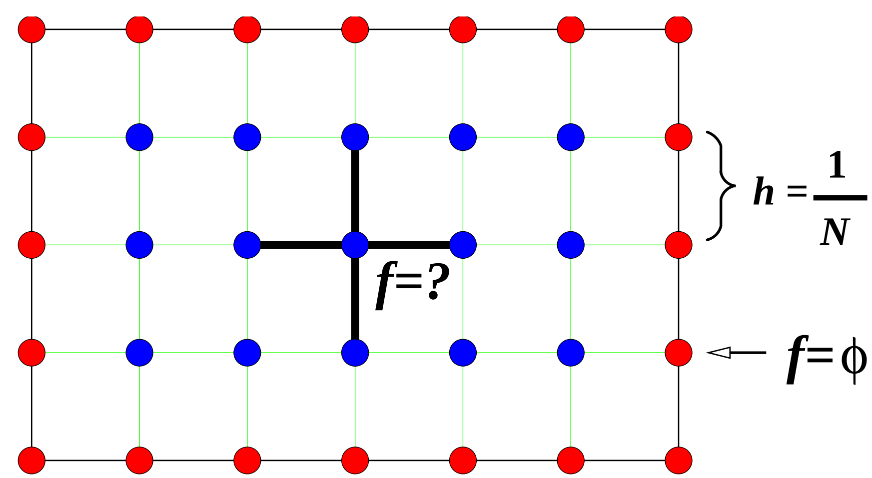
$$\frac{\partial^2 f}{\partial x^2}(x,y) + \frac{\partial^2 f}{\partial y^2}(x,y) = g(x,y), \quad (x,y) \in D = [a,b] \times [c,d], \tag{2}$$

kde $g : D \longrightarrow \Re$ je známá funkce.
Úkolem je nalézt funkci $f : D \longrightarrow \Re$ takovou, že

(a) splní rovnici (2),

(b) na okraji oblasti D má předepsané hodnoty φ.





Diskretizace PDE metodou konečných diferencí

- Síťka $(N+1)^2$ rovnoměrně rozmístěných bodů v oblasti D, $h = 1/N$,

$$f_{i,j} = f\left(\frac{i}{N}, \frac{j}{N}\right), \quad 0 \leq i, j \leq N,$$

$$g_{i,j} = g\left(\frac{i}{N}, \frac{j}{N}\right), \quad 0 < i, j < N.$$

- Aproximace:

$$\frac{\partial^2 f}{\partial x^2}(x,y) \approx \frac{1}{h^2}[f(x+h,y) - 2f(x,y) + f(x-h,y)],$$

$\Longrightarrow$ Rovnice (2) $\rightarrow$ SLR $(N-1)^2$ rovnic s $(N-1)^2$ neznámými $f_{i,j}$

$$f_{i,j} = \frac{1}{4}\left[f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1} - \frac{1}{N} g_{i,j}\right] \tag{3}$$

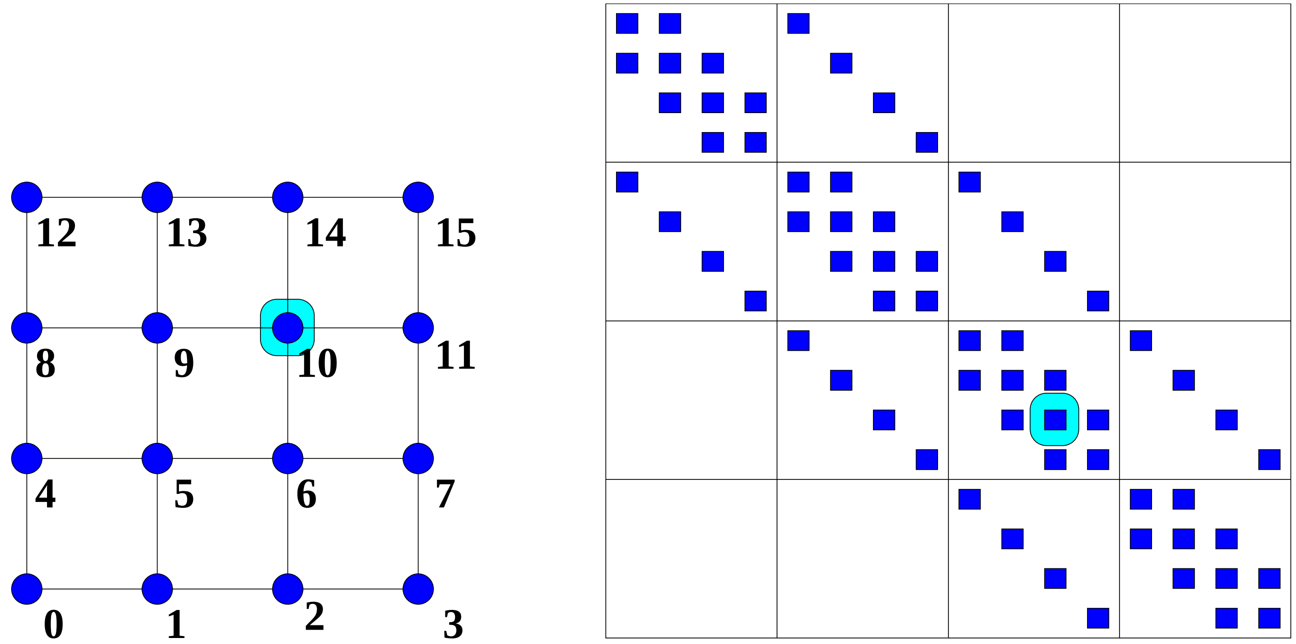
- $\mathcal{A}\overrightarrow{x} = \overrightarrow{b}$, kde
  - $\mathcal{A}$ = matice typu $(N-1)^2 \times (N-1)^2$,
  - $\overrightarrow{x}$ = vektor $(N-1)^2$ neznámých hodnot $f_{i,j}$ na vnitřních bodech síťky,
  - $\overrightarrow{b}$ = vektor hodnot závislých na $g_{i,j}$ a na známých okrajových podmínkách f.

Lexikografické číslování bodů síťky

Matice $\mathcal{A}$ = speciální & řídká = blokově tri-diagonální.

$$\mathcal{A} = \begin{bmatrix} \mathcal{R} & \mathcal{E} & 0 & \dots & \dots & 0 \\ \mathcal{E} & \mathcal{R} & \mathcal{E} & 0 & \dots & \\ 0 & \mathcal{E} & \mathcal{R} & \mathcal{E} & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \\ \vdots & & & \ddots & \ddots & \mathcal{E} \\ 0 & \dots & \dots & 0 & \mathcal{E} & \mathcal{R} \end{bmatrix}, \qquad \mathcal{R} = \begin{bmatrix} -4 & 1 & 0 & \dots & \dots & 0 \\ 1 & -4 & 1 & 0 & \dots & \\ 0 & 1 & -4 & 1 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \\ \vdots & & & \ddots & \ddots & 1 \\ 0 & \dots & \dots & 0 & 1 & -4 \end{bmatrix},$$

a $\mathcal{E}$ = jednotková matice.



(a) (b)

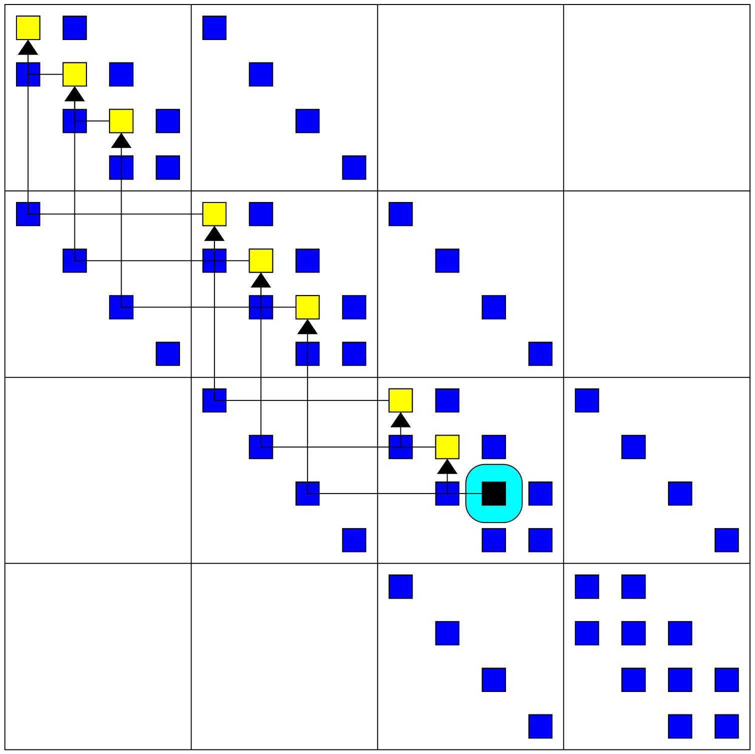
Paralelní Jacobiho algoritmus

Bez problému: číslování bodů mřížky je nepodstatné.

$$f_{i,j}(t+1) = \frac{1}{4}\left[f_{i+1,j}(t) + f_{i-1,j}(t) + f_{i,j+1}(t) + f_{i,j-1}(t) - \frac{1}{N}g_{i,j}\right]. \tag{4}$$

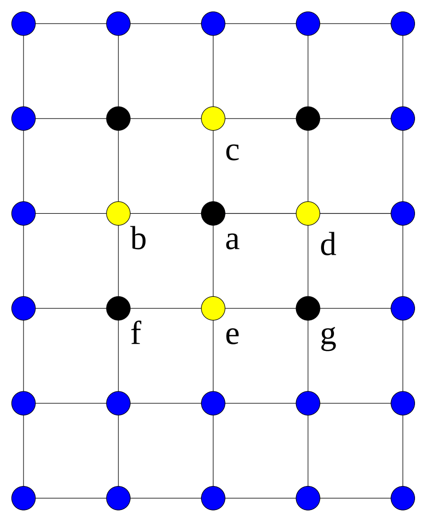
Paralelní Gauss-Seidelův algoritmus

Číslování má dopad na datové závislosti, a tudíž na paralelismus:

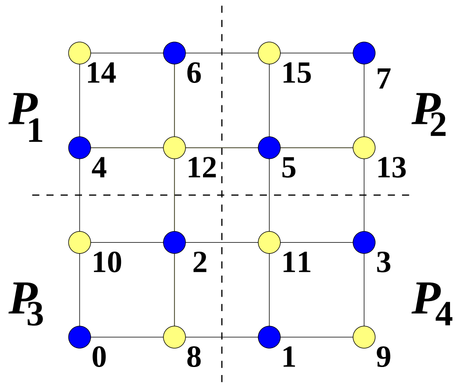


Vysoce paralelní Gauss-Seidelův algoritmus

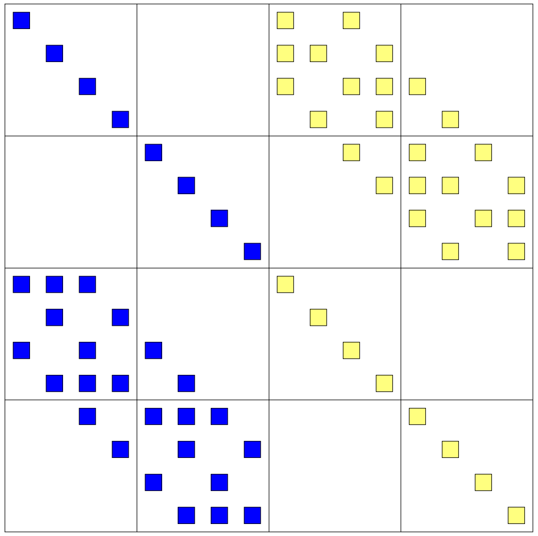
Číslování indukované 2-barvením založeným na paritě



(a)



(b)



(c)

Algoritmus $\textsc{ParallelGS}(\mathcal{A}, \overrightarrow{x}, \overrightarrow{b})$

repeat

  Fáze 0: proveď výpočet (4) na všech **modrých** bodech paralelně;

  Fáze 1: proveď výpočet (4) na všech **žlutých** bodech paralelně;

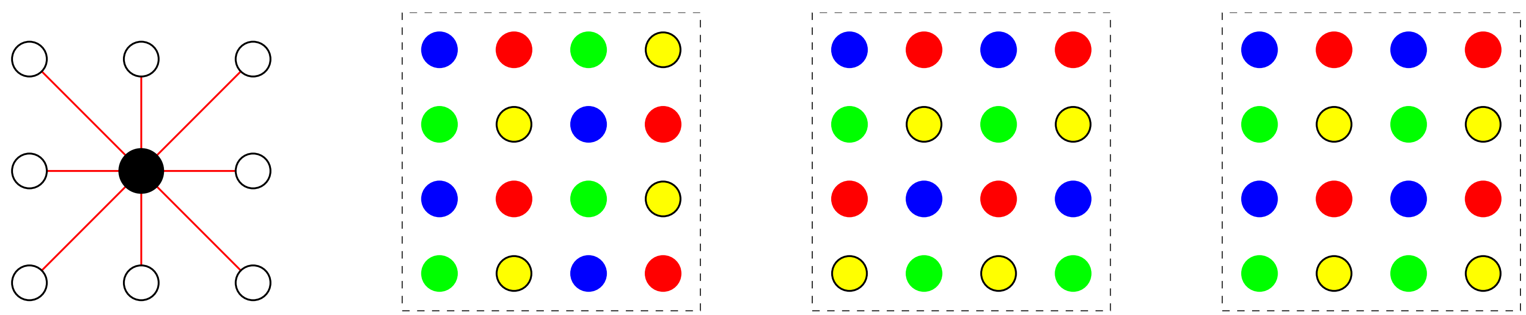
until (není dosaženo dostatečně přesné řešení)

Jiné diskretizační matrice a jejich paralelní řešení

$$\frac{\partial^2 f}{\partial x^2}(x,y) + \frac{\partial^2 f}{\partial y^2}(x,y) + \frac{\partial^2 f}{\partial x \partial y}(x,y) = g(x,y)$$

$\Longrightarrow$ 9-bodová diskretizace

$\exists$ pouze 3 různá barvení 2-D síťky pomocí 4 barev, odpovídající této diskretizaci.



(a) (b)

4-fázový paralelní Gauss-Seidelův algoritmus:

repeat
proveď výpočet na všech **červených** bodech paralelně;
proveď výpočet na všech **modrých** bodech paralelně;
proveď výpočet na všech **zelených** bodech paralelně;
proveď výpočet na všech **žlutých** bodech paralelně;
until (není dosaženo dostatečně přesné řešení)