Paralelní algoritmy v lineární algebře

Tomáš Oberhuber
tomas.oberhuber@fjfi.cvut.cz

15. dubna 2024

Video na Youtube

Gaussova eliminační metoda

- GEM je základem přímých řešičů lineárních soustav rovnic
- ukážeme si paralelizaci pro případ plné matice lineární soustavy
- sekvenční algoritmus převádí vstupní matici na horní trojúhelníkovou

$$\mathbb{A}\vec{x} = \vec{b} \Rightarrow \mathbb{U}\vec{x} = \vec{d}$$

-

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$\Rightarrow$$

$$\begin{pmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1n} \\ & 1 & u_{23} & \cdots & u_{2n} \\ & & \ddots & & \vdots \\ & & & 1 & u_{n-1n} \\ & & & & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{pmatrix},$$

Gaussova eliminační metoda

- výpočet probíhá podle následujícího schématu

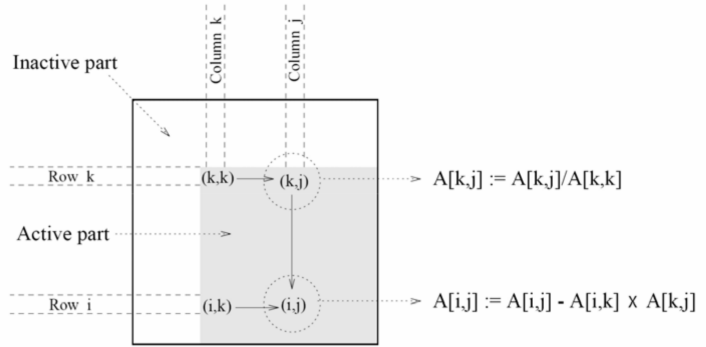


Figure 8.5 A typical computation in Gaussian elimination.

Zdroj: A. Grama, A. Gupta, G. Karypis, V. Kumar, Introduction to Parallel Computing, Pearson/Addison Wesley, 2003

Gaussova eliminační metoda

```
1  void GEM( double A[ n ][ n ], double b[ n ] )
2  {
3     /****
4      * Outer loop
5      */
6     for( int k = 0; k < n; k++ )
7     {
8        /****
9         * Division step
10        */
11       for( int j = k + 1; j < n; j++ )
12          A[ k ][ j ] /= A[ k ][ k ];
13       b[ k ] /= b[ k ] / A[ k ][ k ];
14       A[ k ][ k ] = 1.0;
15       for( int i = k + 1; i < n; i++ )
16       {
17          /****
18           * Elimination step
19           */
20          for( int j = k + 1; j < n; j++ )
21             A[ i ][ j ] -= A[ i ][ k ] * A[ k ][ j ];
22          b[ i ]-=  A[ i ][ k ] * y[ k ];
23          A[ i ][ k ] = 0.0;
24       }
25    }
26 }
```

Gaussova eliminační metoda - zpětný chod

Následně probíhá zpětná substituce pro výpočet řešení:

```
1  for( int k = n-1; k >= 0; k-- )
2  {
3     x[ k ] = b[ k ];
4     for( int j = n-1; j > k; j-- )
5        x[ k ] = x[ k ] - A[ k ][ j ] * x[ j ];
6  }
```

Tento přístup s sebou nese dva problémy:

- během přímého chodu klesá počet prvků matice, které přepočítáváme
  - v k-tém kroku je jich $(n-k)^2$
  - díky tomu ztrácíme možnost paralelizace
- zpětná substituce je obzvlášť nevhodné pro paralelizaci
  - paralelizovat lze jen vnořený cyklus na řádku 4, kde množství výpočtů velmi malé

Gaussova eliminační metoda - zpětný chod

Řešením je nepřevádět matici na horní trojúhelníkovou, ale na diagonální.

- během přímého chodu tedy eliminujeme i prvky nad diagonálou
- tím nám přibude objem výpočtů, které lze provádět paralelně
- odpadá tím kompletně zpětná substituce
- **bohužel, tento postup nelze použít pro výpočet LU rozkladu**

GEM - zpětná substituce

```
1  void GEM2( double A[ n ][ n ], double b[ n ] )
2  {
3     /****
4      * Outer loop
5      */
6     for( int k = 0; k < n; k++ )
7     {
8        /****
9         * Division step
10        */
11       for( int j = k + 1; j < n; j++ )
12          A[ k ][ j ] /= A[ k ][ k ];
13       b[ k ] /= b[ k ] / A[ k ][ k ];
14       A[ k ][ k ] = 1.0;
15       for( int i = 0; i < n; i++ ) // originaly i = k + 1
16       {
17          /****
18           * Elimination step
19           */
20          if( i == k )
21             continue;
22          for( int j = k + 1; j < n; j++ )
23             A[ i ][ j ] -= A[ i ][ k ] * A[ k ][ j ];
24          b[ i ]-=  A[ i ][ k ] * y[ k ];
25          A[ i ][ k ] = 0.0;
26       }
27    }
28 }
```

- paralelizovat lze cykly na řádcích 15 a 22
- tj. výpočet na řádku 23 mohu udělat souběžně pro všechna i, j, kde $i = 0, \ldots, n-1$, $i \neq k$ a $j = k+1, \ldots, n-1$
- nulování prvků na řádku 25 není nutné

GEM - zpětná substituce

Co nás brzdí teď?

- máme-li velký počet vláken/procesů, ne všechny se mohou účastnit dělení pivotem na řádcích 11–13
- nečinná vlákna/procesy musí čekat, což je zdroj neefektivity
- zejména na GPU se vyplací dělen pivotem vynechat a každé vlákno si toto provede samo pro sebe v rámc eliminace

GEM - zpětná substituce

```
1  void GEM2( double A[ n ][ n ], double b[ n ] )
2  {
3     /****
4      * Outer loop
5      */
6     for( int k = 0; k < n; k++ )
7     {
8        for( int i = 0; i < n; i++ )
9        {
10          /****
11           * Elimination step
12           */
13          if( i == k )
14             continue;
15          for( int j = k + 1; j < n; j++ )
16             A[ i ][ j ] -= A[ i ][ k ] * A[ k ][ j ] / A[ k ][ k ];
17          b[ i ]-=  A[ i ][ k ] * y[ k ] / A[ k ][ k ];
18       }
19    }
20    for( int k = 0; k < n; k++ )
21       x[ k ] = b[ k ] / A[ k ][ k ];
22 }
```

- na řádcích 16 a 17 nyní musíme opakovaně dělit pivotem
- navíc musíme opět dělat něco jako zpětnou substituci (řádky 20 a 21), ale tu lze nyní provést zcela paralelně

GEM - složitost

Složitost, pokud mapujeme jedno vlákno/proces na jeden řádek:

- $T_S(n) = \theta(n^3)$
- $T_P(n, p) = \theta(n^3/p)$
- $S(n, p) = p$
- $E(n, p) = 1$
- $C(n, p) = n^3$

- jde tedy o velmi dobrou paralelizaci
- problém nastává pouze tehdy, pokud mapujeme více vláken/procesů na jeden řádek
- ke konci eliminace jsou pak některá vlákna/procesy nevyužita

Algoritmus lze snadno modifikovat pro pivoting, pokud se omezíme pouze na prohazování řádků.

GEM

- uvedené paralelní algoritmy výrazně využívají faktu, že pracujeme s hustými maticemi
- pro řídké matice je potřeba použít mnohem složitější algoritmy
  - `http://www.pardiso-project.org/`
- ukážeme si paralelizaci řešiče pro tridiagonální lineární systémy rovnic

Thomasův algoritmus

- mějme následující soustavu se silně regulární maticí

$$\begin{pmatrix} a_1 & b_1 & & & & \\ c_2 & a_2 & b_2 & & & \\ & c_3 & a_3 & b_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & c_{n-1} & a_{n-1} & b_{n-1} \\ & & & & c_n & a_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n-1} \\ d_n \end{pmatrix}$$

Thomasův algoritmus

- pomocí GEM ji lze převést na tvar

$$\begin{pmatrix} 1 & \mu_1 & & & & \\ & 1 & \mu_2 & & & \\ & & 1 & \mu_3 & & \\ & & & \ddots & \ddots & \\ & & & & 1 & \mu_{n-1} \\ & & & & & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \\ \vdots \\ \rho_{n-1} \\ \rho_n \end{pmatrix}$$

- a zpětná substituce pak má tvar

$$\begin{aligned} x_n &= \rho_n \\ x_{n-1} &= \rho_{n-1} - \mu_{n-1} x_n \\ &\vdots \\ x_k &= \rho_k - \mu_k x_{k+1} \\ &\vdots \\ x_1 &= \rho_1 - \mu_1 x_2 \end{aligned}$$

Thomasův algoritmus

- algoritmus lze zapsat takto

```
1  void Thomas( double a[ n ], double b[ n ], double c[ n ],
2               double d[ n ],
3               double x[ n ] )
4  {
5     double mu[ n ], rho[ n ];
6     mu[ 0 ] = b[ 0 ] / a[ 0 ];
7     rho[ 0 ] = d[ 0 ] / a[ 0 ];
8     for( int i = 1; i < n - 1; i++ )
9     {
10       mu[ i ] = b[ i ] / ( a[ i ] - c[ i ] * mu[ i - 1 ] );
11       rho[ i ] = ( d[ i ] - c[ i ] * rho[ i - 1 ] ) /
12                  ( a[ i ] - c[ i ] * mu[ i - 1 ] ) ;
13    }
14    rho[ n - 1 ] = ( d[ n - 1 ] - c[ n - 1 ] * rho[ n - 2 ] ) /
15                   ( a[ n - 1 ] - c[ n - 1 ] * mu[ n - 2 ] ) ;
16    x[ n - 1 ] = rho[ n - 1 ];
17    for( i = n - 2; i >= 0; i-- )
18       x[ i ] = rho[ i ] - mu[ i ] * x[ i + 1 ];
19
20 }
```

- jde o čistě sekvenční kód
- pomocí šikovného triku ho ale lze paralelizovat

Cyklická redukce

- vezměme si řádek i pro $1 < i < n$ a dva jeho sousedy

$$\begin{aligned} c_{i-1}x_{i-2} + a_{i-1}x_{i-1} \quad & +b_{i-1}x_i & & & = d_{i-1}, \\ c_i x_{i-1} \quad & +a_i x_i \quad & +b_i x_{i+1} & & = d_i, \\ & c_{i+1}x_i \quad & +a_{i+1}x_{i+1} + b_{i+1}x_{i+2} \quad & & = d_{i+1} \end{aligned}$$

- z prostřední rovnice eliminujeme proměnné x_{i-1} a x_{i+1}
- první rovnici vynásobíme číslem $\beta_i = \frac{c_i}{a_{i-1}}$ a třetí číslem $\gamma_i = \frac{b_i}{a_{i+1}}$ a obě odečteme od prostřední
- dostaneme

$$c_i^{(1)} x_{i-2} + a_i^{(1)} x_i + b_i^{(1)} x_{i+2} = d_i^{(1)},$$

pro

$$\begin{aligned} c_i^{(1)} &= -\beta_i c_{i-1} \\ a_i^{(1)} &= a_i - \beta_i b_{i-1} - \gamma_i c_{i+1} \\ b_i^{(1)} &= -\gamma_i b_{i+1} \\ d_i^{(1)} &= d_i - \beta d_{i-1} - \gamma d_{i+1}. \end{aligned}$$

Cyklická redukce

- dodefinováním $c_1 = 0$ a $b_n = 0$ lze tuto úpravu provést pro každou sudou rovnici
- původní soustavu tak převedeme na redukovaný tvar

$$\begin{pmatrix} a_1^{(1)} & b_1^{(1)} & & & & & \\ c_2^{(1)} & a_2^{(1)} & b_2^{(1)} & & & & \\ & c_3^{(1)} & a_3^{(1)} & b_3^{(1)} & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & c_{n/2-1}^{(1)} & a_{n/2-1}^{(1)} & b_{n/2-1}^{(1)} \\ & & & & c_{n/2}^{(1)} & a_{n/2}^{(1)} \end{pmatrix} \begin{pmatrix} x_2 \\ x_4 \\ x_6 \\ \vdots \\ x_{n-2} \\ x_n \end{pmatrix} = \begin{pmatrix} d_1^{(1)} \\ d_2^{(1)} \\ d_3^{(1)} \\ \vdots \\ d_{n/2-1}^{(1)} \\ d_{n/2}^{(1)} \end{pmatrix}$$

- pokud tuto soustavu vyřešíme, dokážeme pak ze znalosti x_{i-2}, x_i a x_{i+2} dopočítat i x_{i-1} a x_{i+1} ze vztahů původní soustavy

$$\begin{aligned} c_{i-1}x_{i-2} + a_{i-1}x_{i-1} \quad & +b_{i-1}x_i & & = d_{i-1}, \\ & c_{i+1}x_i \quad & +a_{i+1}x_{i+1} + b_{i+1}x_{i+2} \quad & = d_{i+1} \end{aligned}$$

- tak tedy získáme i všechna lichá x_i
- všimneme si, že redukovanou soustavu můžeme řešit stejným postupem...

Cyklická redukce

Příklad:

- mějme úlohu o pěti neznámých

$$\left(\begin{array}{ccccc|c} a_1 & b_1 & & & & d_1 \\ c_2 & a_2 & b_2 & & & d_2 \\ & c_3 & a_3 & b_3 & & d_3 \\ & & c_4 & a_4 & b_4 & d_4 \\ & & & c_5 & a_5 & d_5 \end{array}\right)$$

- budeme na sudých řádcích budeme eliminovat koeficienty odpovídající proměnným s lichým indexem, tj. indexy podél diagonály
  - od druhého řádku odečteme c_2/a_1 násobek prvního a b_2/a_3 násobek třetího
  - od čtvrtého řádku odečteme c_4/a_3 násobek třetího a b_4/a_5 násobek pátého

Cyklická redukce

- dostaneme

$$\left(\begin{array}{ccccc|c} a_1 & b_1 & 0 & 0 & 0 & d_1 \\ 0 & a_2 - \frac{c_2}{a_1}b_1 - \frac{b_2}{a_3}c_3 & 0 & -\frac{b_2}{a_3}b_3 & 0 & d_2 - \frac{c_2}{a_1}d_1 - \frac{b_2}{a_3}d_3 \\ 0 & c_3 & a_3 & b_3 & 0 & d_3 \\ 0 & -\frac{c_4}{a_3}c_3 & 0 & a_4 - \frac{c_4}{a_3}b_3 - \frac{b_4}{a_5}c_5 & 0 & d_4 - \frac{c_4}{a_3}d_3 - \frac{b_4}{a_5}d_5 \\ 0 & 0 & 0 & c_5 & a_5 & d_5 \end{array}\right)$$

- což přeznačíme na

$$\left(\begin{array}{ccccc|c} a_1^0 & b_1^0 & 0 & 0 & 0 & d_1^0 \\ 0 & b_2^1 & 0 & c_2^1 & 0 & d_2^1 \\ 0 & c_3^0 & a_3^0 & b_3^0 & 0 & d_3^0 \\ 0 & a_4^1 & 0 & b_4^1 & 0 & d_4^1 \\ 0 & 0 & 0 & c_5^0 & a_5^0 & d_5^0 \end{array}\right) \tag{1}$$

- a sudé rovnice lze extrahovat

$$\left(\begin{array}{cc|c} b_2^1 & c_2^1 & d_2^1 \\ a_4^1 & b_4^1 & d_4^1 \end{array}\right)$$

Cyklická redukce

- to je obecně opět tridiagonální soustava
- pokud by byla větší, lze na ní celý postup opakovat, až dostaneme soustavu pro jednu nebo dvě neznámé
- tu lze snadno vyřešit $\rightarrow x_2, x_4$
- soustavu 1 lze upravit na tvar

$$\left(\begin{array}{ccccc|c} a_1^0 & b_1^0 & 0 & 0 & 0 & d_1^0 \\ 0 & 1 & 0 & 0 & 0 & x_2 \\ 0 & c_3^0 & a_3^0 & b_3^0 & 0 & d_3^0 \\ 0 & 0 & 0 & 1 & 0 & x_4 \\ 0 & 0 & 0 & c_5^0 & a_5^0 & d_5^0 \end{array}\right)$$

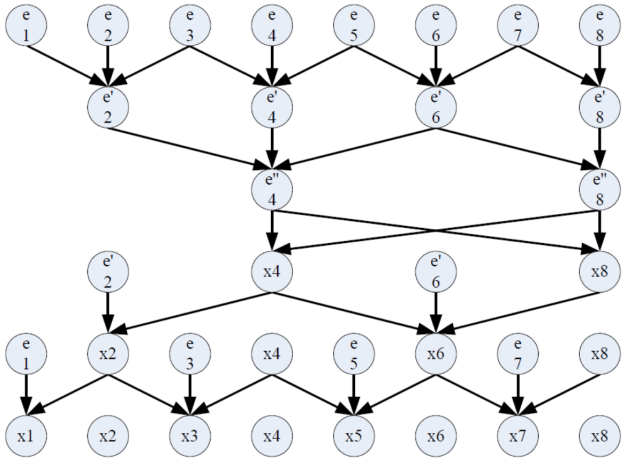
- ze znalosti x_4 lze dopočítat x_5
- ze znalosti x_2 a x_4 lze dopočítat x_3
- ze znalosti x_2 lze dopočítat x_1

Cyklická redukce

Celý výpočet pak probíhá takto:

1: $a_i^{(0)} = a_i$ pro $i = 1, \ldots, n$
2: $c_i^{(0)} = c_i$ pro $i = 2, \ldots, n$ a $c_1^{(0)} = 0$
3: $b_i^{(0)} = b_i$ pro $i = 1, \ldots, n-1$ a $b_n^{(0)} = 0$
4: **for** $k = 1 \ldots \lfloor \log_2 n \rfloor$ **do**
5: $\quad s = 2^k$
6: $\quad$ **for** $i = 1, \ldots, \lfloor n/2^k \rfloor$ **do**
7: $\quad\quad \beta_i = \frac{c_i^{(k-1)}}{a_{i-1}^{(k-1)}}, \gamma_i = \frac{b_i^{(k-1)}}{a_{i+1}^{(k-1)}}$
8: $\quad\quad c_{i/2}^{(k)} = -\beta_i c_{i-1}^{(k-1)}$
9: $\quad\quad a_{i/2}^{(k)} = a_i^{(k-1)} - \beta_i b_{i-1}^{(k-1)} - \gamma_i c_{i+1}^{(k-1)}$
10: $\quad\quad b_{i/2}^{(k)} = -\gamma_i b_{i+1}^{(k-1)}$
11: $\quad\quad d_{i/2}^{(k)} = d_i^{(k-1)} - \beta_i d_{i-1}^{(k-1)} - \gamma_i d_{i+1}^{(k-1)}$
12: $\quad$ **end for**
13: **end for**
14: vyřeš výslednou rovnici (nebo dvě)
15: **for** $k = \lfloor \log_2 n \rfloor \ldots 1$ **do**
16: $\quad s = 2^k$
17: $\quad$ **for** $i = 1, \ldots, \lfloor n/2^k \rfloor$ **do**
18: $\quad\quad$ ze znalosti x_{is} dopočítej $x_{is-s/2}$ a $x_{is+s/2}$
19: $\quad$ **end for**
20: **end for**

Cyklická redukce



Jednotlivé kroky cycklické redukce pro tridiagonální soustavu s osmi neznámými.
e značí indexy řádkům x indexy neznámých.

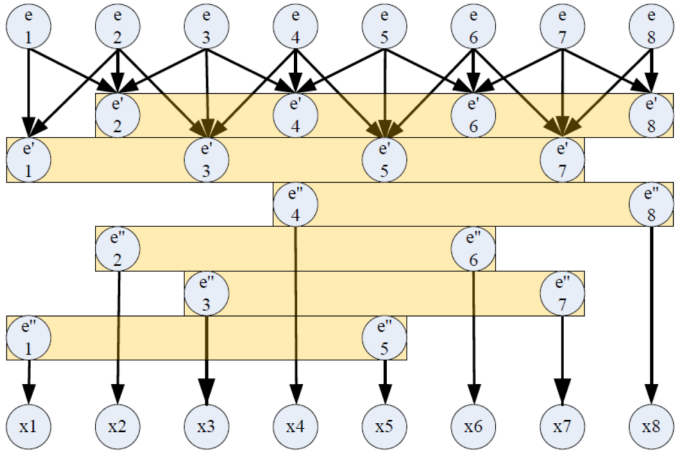
Cyklická redukce

- všimneme si, že jednotlivé kroky cyklické redukce lze provádět paralelně
- počet aktivních procesů se v každém kroku redukuje na polovinu
- snadno vidíme, že pro $p = n$
  - $T_S(n) = \theta(n)$
  - $T_P(n, p) = \theta(\log n)$
  - $C(n, p) = \theta(n \log n)$
  - $S(n, p) = \theta(\frac{n}{\log n})$
  - $E(n, p) = \theta(\frac{1}{\log n})$
- paralelizace není ideální, lze ji o trochu vylepšit

Cyklická redukce

- v prvním kroku jsme eliminovali liché řádky soustavy
- nyní uděláme to samé i se sudými řádky
- dostaneme dvě nezávislé soustavy poloviční velikosti, jednu pro sudé neznámé a druhou pro liché neznámé
- tak postupujeme dále
- v dalším kroku získáme čtyři čtvrtinové soustavy atd.
- po $\lfloor \log_2 n \rfloor$ krocích získáme $n/2$ až n nezávislých soustav o jedné nebou dvou neznámých, které lze vyřešit paralelně
- ušetřím si tak druhou fázi algoritmu o $\log_2 n$ krocích
- doba běhu se sníží na polovinu a stále udržujeme všechny procesy aktivní
- i tento algoritmus ale zůstává nákladově neoptimální

Cyklická redukce



Pruhy na obrázku znázorňují jednotlivé nezávislé soustavy rovnic.

SOR metoda

Super-overrelaxation method

- jde o iterativní metodu pro řešení soustav lineárních rovnic
- při napočítávání nové iterace $x^{(k+1)}$ z $x^{(k)}$ využívá již napočítané složky vektoru $x^{(k+1)}$

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}x_j^{(k+1)} - \sum_{j>i} a_{ij}x_j^{(k)}\right)$$

SOR metoda

```
1  bool sorMethod( const Matrix& A,
2                  const Vector& b,
3                  Vector& x,
4                  double eps,
5                  double omega )
6  {
7     double normB = norm( b );
8     double r = eps + 1.0;
9     while( r > eps )
10    {
11       /***
12        * SOR iterace
13        */
14       for( int i = 0; i < n; i++ )
15       {
16          double s = 0.0;
17          for( int j = 0; j < n; j++ )
18             if( j != i )
19                s = s + A[ i ][ j ] * x[ j ];
20          x[ i ] += omega * ( b[ i ] - s ) / A[ i ][ i ];
21       }
22
23       /***
24        * Vypocet rezidua
25        */
26       r = 0.0;
27       for( int i = 0; i < n; i++ )
28       {
29          double Ax_i = 0.0;
30          for( int j = 0; j < n; j++ )
31             Ax_i = Ax_i + A[ i ][ j ] * x[ j ];
32          r = r + ( Ax_i - b_i ) * ( Ax_i - b_i );
33       }
34       r = sqrt( r ) / normB;
35    }
36 }
```

SOR metoda

- budeme se zabývat paralelizací této smyčky

```
1   /****
2    * SOR iterace
3    */
4
5   for( int i = 0; i < n; i++ )
6   {
7      double s = 0.0;
8      for( int j = 0; j < n; j++ )
9         if( j != i )
10           s = s + A[ i ][ j ] * x[ j ];
11     x[ i ] += omega * ( b[ i ] - s ) / A[ i ][ i ];
12  }
```

- řádky 8-10 vlastně odpovídají násobení i-tého řádku matice $\mathbb{A}$ s vektorem x s vynecháním diagonálního prvku
- to lze snadno paralelizovat s využitím paralelní redukce
- pro matici $A \in \mathbb{R}^{n,n}$ lze využít maximálně n procesů
- pokud bude matice A řídká, může se stát, že naprostá většina procesů nebude mít nic na práci
- vlastně tak nemáme dost práce pro celkem n procesů

SOR metoda

- v případě řídkých matic lze provést rozklad na skupiny nezávislých řádků a provést paralelizaci cyklu na řádku 15
- vysvětlíme si to na matici aproximující Laplaceův operátor konečnými diferencemi na strukturované síti
- řešíme tedy úlohu na $\Omega \equiv [0,1] \times [0,1]$

$$\begin{aligned} -\Delta u &= f \text{ na } \Omega, \\ u\,|_{\partial\Omega} &= g, \end{aligned}$$

SOR metoda

- volíme pravidelnou čtvercovou síť

$$\omega_h \equiv \{x_{ij} = (ih, jh) \mid i = 1, \cdots N-1, j = 1, \cdots N-1\}$$

- na vnitřních uzlech pro $0 < i, j < N$ aproximujeme operátor pomocí konečných diferencí

$$\Delta u_{ij} \approx \frac{u^h_{i-1,j} - 2u^h_{ij} + u^h_{i+1,j}}{h^2} + \frac{u^h_{i,j-1} - 2u^h_{ij} + u^h_{i,j+1}}{h^2}$$

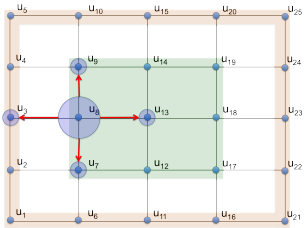
- tj. pro vnitřní uzly $0 < i, j < N$ řešíme soustavu rovnic

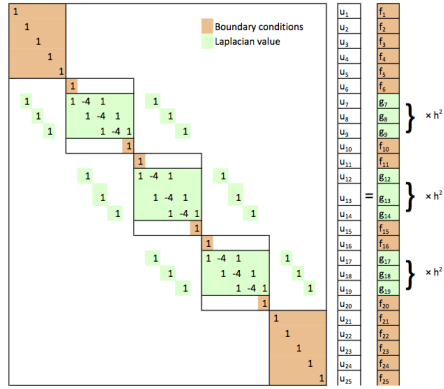
$$-u^h_{i,j-1} + u^h_{i-1,j} - 4u^h_{ij} + u^h_{i+1,j} + u^h_{i,j+1} = h^2 f_{ij}$$

- a pro okrajové

$$u_{ij} = g_{ij}.$$

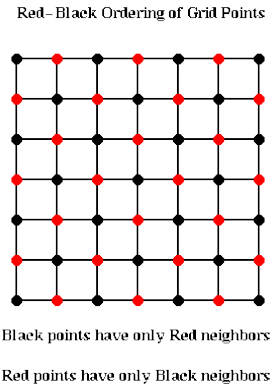
SOR metoda





Zdroj: http://math.stackexchange.com/questions/329765/discretization-of-inhomogeneous-dirichlet-boundary-conditions-for-2d-poissons-e

SOR metoda



Zdroj: http://www.cs.berkeley.edu/~demmel/cs267/lecture24/lecture24.html

SOR metoda

- obecně si sestrojíme graf k němuž bude daná matice tvořit adjacenční matici
- provedeme vrcholové obarvení
- to odpovídá tomu, že každému sloupci matice přiřadíme jednu barvu a to tak, že na žádném řádku se nesmí vyskytnout nenulový prvek stejné barvy jako diagonální prvek
- SOR iteraci pak rozdělíme na bloky řádků se stejnou barvou na diagonále

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}x_j^{(k+1)} - \sum_{j>i} a_{ij}x_j^{(k)}\right)$$

- protoze všechny takové řádky jsou na sobě nezávislé, lze je zpracovat současně