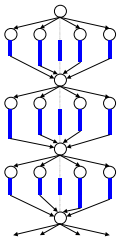
Týden 5
Bariéra a synchronní iterační výpočty.
Bezpečné zasílání zpráv v MPI.

Synchronizace procesů: Bariéra

Vkládá se do místa, kde na sebe musí procesy počkat.

- U procesů se synchronní komunikací není nutná, každá komunikace je zároveň synchronizace. (Transim)
- Použita u obecnějšího modelu PRAM, tzv. Asynchronního PRAM (APRAM), odpadá synchronizace na úrovni instrukcí.
- U asynchronní komunikace v MPI, blokující i neblokující, je pro synchronizaci procesů potřeba. MPI_barrier(Comm) synchronizuje všechny procesy v komunikátoru Comm - čeká se, až všechny procesy zavolají MPI_barrier(Comm).
- Bude rovněž používána u programů se sdílenými proměnnými k synchronizaci vláken.

Synchronizované iterace



- paralelní zpracování stejných podúloh s různými daty (SPMD) nebo nezávislých různých podúloh (MPMD) → různé doby provedení
- po každé iteraci je nutná bariérová synchronizace
- opakuje se až podmínka přestane platit
- účinnost závisí na různosti časů podúloh a na režii bariéry
- také se používá u programování s vlákny (multithreading)

Bariéra s centrálním čítačem

- sdílený čítač je vhodný u **sdílené paměti** a u modelu APRAM
- Kvalita implementace je dána
  - počtem sdílených proměnných a zámků
  - časovou režií v závislosti na počtu procesů
- U **zasílání zpráv** provádí centrální čítání vyčleněný proces Master:

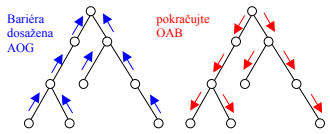
```
for (i=0; i<n; i++)      /*počítej procesy u bariéry v libovolném pořadí*/
   recv(P_any);
for (i=0; i<n; i++)      /*oznam procesům, že mohou pokračovat */
   send(P_i);
```

Kód bariéry u ostatních procesů:

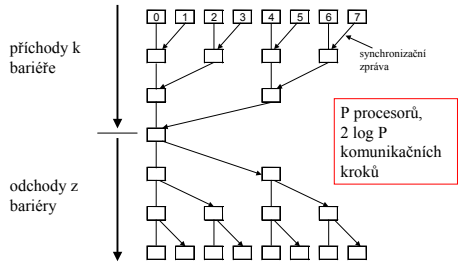
```
send(P_master);   /*jsem u bariéry */
recv(P_master);   /*čekám na povolení pokračovat */
```

Efektivnější implementace bariéry u MP

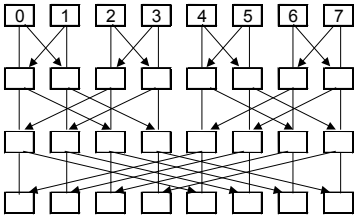
- každý proces pošle (stejnou) zprávu procesu root, AOG
- při tom se použije strom pro rozhlašování: vnitřní uzel kombinuje zprávy od všech svých dětí a pošle jednu zprávu svému rodiči
- po dosažení procesu root vyšle root opačným směrem zprávu „pokračujte“ do všech uzlů (OAB), čímž otevře bariéru



Implementace bariéry úplným binárním stromem, AAB = AOG + OAB



Motýlková bariéra jako AAB



P procesorů, logP komunikačních kroků

Iterativní řešení lineárních rovnic

- *přibližné* řešení, se zadanou přesností **ε**
- husté nebo řídké (např. multi-diagonální) systémy rovnic
- základní myšlenka: začne se s nějakou počáteční aproximací řešení soustavy **Ax** = **b** a iterativně se zlepšuje její přesnost;
  x(t+1) = **f**[**b**, **A**, **x**(t)]
- počet iterací je dán požadovanou přesností nebo časovým omezením
- řešení je *kratší* než u přímých metod v případech speciálních a řídkých systémových matic, jaké se např. získají při numerickém řešení parciálních diferenciálních rovnic (PDE)
- iterativní paralelní algoritmy jsou důležité ve vědě a technice.

Jacobiho iterační metoda

Přepisem rovnic **Ax** = **b** získáme

$$x_i = \frac{1}{a_{ii}}\left[b_i - \sum_{j\neq i} a_{ij}x_j\right] = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{n} a_{ij}x_j\right] + x_i, \qquad a_{ii} < 0$$

Jacobiho metoda: v aktuální iteraci (t +1) se použijí pouze hodnoty $x_i(t)$ jaké byly na konci předchozí iterace

$$x_i(t+1) = \frac{1}{a_{ii}}\left[b_i - \sum_{j\neq i} a_{ij}x_j(t)\right] = \frac{1}{a_{ii}}\left[b_i - \sum_{j=1}^{n} a_{ij}x_j(t)\right] + x_i(t) = \frac{r_i(t)}{a_{ii}} + x_i(t)$$

Postačující podmínka pro konvergenci: **A** je diagonálně dominantní, t.j.

$$\sum_{j\neq i}|a_{ij}| < |a_{ii}|$$

Ukončení výpočtu: $\sum_{i=1}^{n}\left[x_i(t+1) - x_i(t)\right]^2 = \sum_{i=1}^{n}\frac{r_i^2(t)}{a_{ii}^2} < \varepsilon$

Paralelizace Jacobiho metody

Každá iterace obsahuje násobení matice a vektoru a výpočet dosažené přesnosti (paralelní redukce).

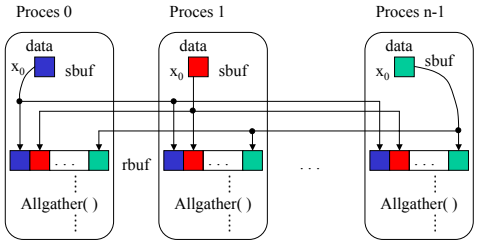
- **Dekompozice po blocích řádků** - každý procesor dostane n/P po sobě jdoucích řádků husté matice A a počítá n/P hodnot neznámých $x_i(t+1)$. K tomu musí znát také hodnoty $x_j(t)$ vypočítané v předchozí iteraci všemi ostatními procesory
  → komunikace AAB;
- **Dekompozice po blocích sloupců** - každý procesor dostane n/P po sobě jdoucích sloupců A a vypočte jejich vektorový příspěvek k $x(t+1)$.
  → all-to-all paralelní redukce (+), má stejnou složitost jako AAB.

Jacobiho metoda, dekompozice po řádcích

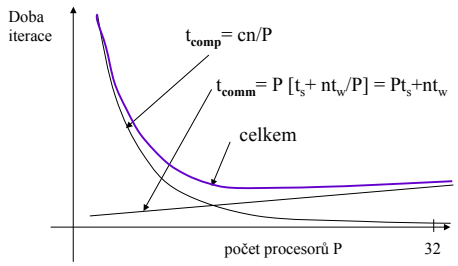
```
/*každý z P procesorů má n/P řádků matice systému*/
/* a celý vektor x[i]; počítá n/P prvků new_x[i]*/
for(i=myid*n/P; i<(myid+1)*n/P; i++){
  x[i]=b[i];        /*inicializace*/
  do {
     iter++;
     sum = 0;
     for(j=0; j<n; j++)
        sum = sum + a[i][j]*x[j];
     new_x[i]= (b[i]-sum)/a[i][i] + x[i]
     MPI_Allgather(&new_x[myid*n/P],n/P, MPI_FLOAT,
                   &x[myid*n/P],n/P,MPI_FLOAT,Comm)
     err = tolerance();
  }while ((err<eps)&&(iter<limit));
)
```

Implicitní bariéra

Komunikace AAB = MPI_Allgather( )



Výpočty a komunikace v Jacobiho iteracích



Gauss-Seidelova iterační metoda

- Nové hodnoty $x_j(t+1)$ aktualizované v probíhající iteraci se ihned použijí ještě v této iteraci pro výpočet $x_i(t+1)$, $j < i$ → rychlejší než sekvenční Jakobiho metoda

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

nově aktualizované v této iteraci

aktualizované v předchozí iteraci

- Z principu sekvenční, nevhodná pro paralelizaci, s výjimkou řídkých a speciálníc matic.

Přerelaxované iterační metody

Kvůli potlačení možných oscilací řešení a zrychlení konvergence se vezme jako nová hodnota váhový průměr staré hodnoty a nové hodnoty vypočtené původním způsobem ($\gamma = 1$); $0 < \gamma < 1$ … relaxační parametr.

Jacobiho přerelaxace (Jacobi overrelaxation, JOR)

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

Gauss-Seidelova přerelaxace (SOR)

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

Diskretizace parciálních diferenciálních rovnic

$$\frac{\partial^2}{\partial x^2}f(x,y) + \frac{\partial^2}{\partial y^2}f(x,y) = 0$$

Laplaceova rovnice (rozložení teploty v oblasti)

Ve (čtvercové) oblasti budeme uvažovat síť $(N+1)^2$ pravidelně rozmístěných bodů i/N, j /N, $0 \le i, j \le N$. Aproximací diferenciálů na síti vnitřních bodů (0< i,j < N, h = 1/N) získáme

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

a podobně i pro y. Dosazením získáme soustavu $(N-1)^2$ rovnic

$$f_{i,j} = \tfrac{1}{4}\left(f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1}\right)$$

tedy obecný bod závisí na svém 4-okolí.

(tzv. metoda konečných diferencí).

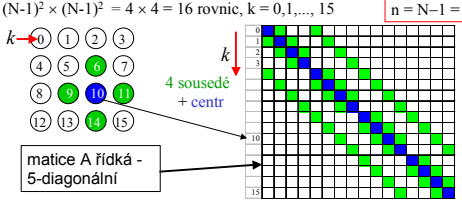
Relace mezi PDE a systémem lin. rovnic

$$f_{i,j} = \tfrac{1}{4}\left(f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1}\right)$$ 2D indexace → 1D indexace

$$f_k = \tfrac{1}{4}\left(f_{k+1} + f_{k-1} + f_{k+n} + f_{k-n}\right) \longrightarrow f_{k-n} + f_{k-1} - 4f_k + f_{k+1} + f_{k+n} = 0$$

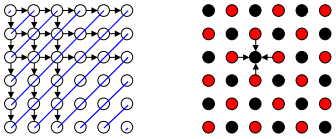
2D mřížka $(N+1) \times (N+1) = 6 \times 6$ bodů včetně hranic, $(N-1)^2 \times (N-1)^2 = 4 \times 4 = 16$ rovnic, k = 0,1,..., 15

N = 5,
n = N−1 = 4



Paralelizace řešení vzniklých systémů rovnic

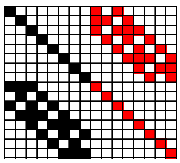
1. Jacobiho metoda bez problémů, na pořadí výpočtu nových hodnot nezáleží. Iteruje se tak dlouho, až je rozdíl hodnot < ε.
2. U Gauss-Seidelovy metody záleží na zvoleném očíslování bodů v rovinné oblasti a tím i na pořadí jejich aktualizace. Jelikož nejde o metodu pro přesné řešení (jako je např. Gauss-Jordanova metoda), ale spíše o iterace až do konvergence, může se použít různých očíslování: anti-diagonální, červeno-černé.



Paralelizace Gauss-Seidelovy metody

- Přirozený postup po řádcích 2D oblasti je vnitřně sekvenční - výpočet $x_i(t+1)$ vyžaduje znalost $x_j(t+1)$ pro všechna j < i . Nelze paralelizovat.
- Antidiagonální postup umožňuje jistou paralelizaci výpočtu - body na antidiagonálách lze aktualizovat paralelně; jejich počet roste od 1 do maxima a pak zas klesá.
- Červeno-černé obarvení lze paralelizovat nejlépe. Střídavě se počítají černé a červené body. Na pořadí aktualizace červených nebo černých bodů nezáleží.

Reprezentace matice A systému:



Diskretizace jiných PDE

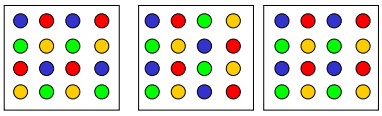
	Systém lineárních rovnic popsaný maticí A $n \times n$
Jednorozměrné simulace:	
1D mřížka, 2 sousedé	3-diagonální
1D mřížka, 4 sousedé	5-diagonální
Dvourozmerné simulace:	
2D mřížka, 4 sousedé	5-diagonální
2D mřížka, 8 sousedů	9-diagonální
2D, 9 sousedů	10-diagonální
Třírozměrné simulace	
3D, 6 sousedů	7-diagonální
3D, 27 sousedů, 9 v každé rovině	28-diagonální

Příklad 10-diagonálního systému

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

Aproximace diferenciálů používá všech 9 okolních bodů a řešení probíhá ve 4 fázích, protože mřížku lze obarvit 4 barvami (třemi způsoby). Body obarvené stejnou barvou lze aktualizovat paralelně.

9-okolí



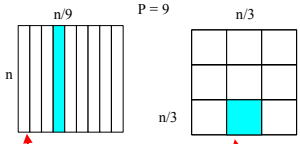
Asynchronní Gauss-Seidelova metoda

- ignoruje závislosti mezi body mřížky v rámci jednoho průchodu
- procesy aktualizující části oblasti postupují shora dolů, zleva doprava
- globální synchronizace procesů se vkládá mezi jednotlivé průchody mřížkou
- proces aktualizuje hodnoty všech přidělených bodů použitím okolních bodů bez ohledu na to, zda byly aktualizovány v probíhající iteraci nebo předchozí
- pro jeden proces vede tato metoda k sekvenční aktualizaci po řádcích
- pro více procesů je uspořádání nepredikovatelné, závisí na rozdělení oblasti mezi procesy

Rozčlenění datové domény n x n bodů

- po proužcích
- po blocích

1 procesor: n^2/P bodů



Komunikace hraničních bodů (1-port, half duplex):

$$t_{comm1} = 4(t_s + nt_w) \qquad t_{comm2} = 8(t_s + nt_w/\sqrt{P})$$

Členění po proužcích je lepší než blokové pro velké časy t_s ,

$t_{comm1} < t_{comm2}$ pro $t_s > nt_w(1 - 2/\sqrt{P})$ a $P > 4$

Výkonnost iteračních výpočtů na mřížce

Mřížka $n \times n$, členění po proužcích
q = počet komunikací, které musí probíhat v jednom procesoru sekvenčně, $q \geq 1$ komunikací nepřekryto výpočtem

$$T_{comm} = q(t_s + t_w n) \qquad T_{comp} = t_c n^2 / P$$

$$T(1) = t_c n^2 \qquad T(P) = t_c n^2 / P + q(t_s + t_w n)$$

$$S = \frac{T(1)}{T(P)} = \frac{t_c n^2}{t_c n^2 / P + q(t_s + t_w n)} \qquad E = \frac{S}{P}$$

$$\lim_{P \to \infty} S = \frac{t_c n^2}{q(t_s + t_w n)} = S_{\max} \quad = \frac{1\mu s 2^{14}}{1(200\mu s + 2\mu s 2^7)} = 36$$

Překrytí komunikace a výpočtů

Po výpočtu hraničních bodů posílá procesor tyto hodnoty sousedům a současně počítá vnitřní body proužku

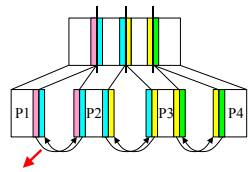
$$S = \frac{T(1)}{T(P)} = \frac{t_c n^2}{t_c n^2 / P + q(t_s + t_w n)}$$

Úplné překrytí (S=P):

$$q(t_s + t_w n) < t_c n(n/P - 2) \quad \rightarrow \quad P \leq 23$$

Proužek bez hraničních sloupců

Komunikace stínových sloupců



Část matice každého procesoru je rozšířena o tzv. stínové sloupce, které se kopírují ze sousedních procesorů.

Vnitřní CPU: 4 komunikace, krajní CPU: 2 komunikace

Pokud všechny procesy napřed posílají a pak příjmají zprávy, není komunikace bezpečná, protože spoléhá na bufer pro send. Není-li jeho kapacita dostatečná, mohlo by dojít k deadlocku.

Bezpečná komunikace - pokrač.

Potlačení deadlocku:

- komunikovat paralelně, je-li možno
- střídat pořadí send() a redcv() u sousedních procesů
- použít kombinovanou komunikaci MPI_Sendrecv(), která vylučuje deadlock
- použít buferovaný send: MPI_Bsend(), kde uživatel nárokuje určitý prostor buferu explicitně
- použít neblokující komunikaci: MPI_Isend() a MPI_Irecv(). Pak je třeba zjistit, zda zpráva byla přijata:
  blokující MPI_Wait(), MPI_Waitall(), MPI_Waitany(), nebo
  neblokující MPI_Test(), MPI_Testall(), MPI_Testany().

Bezpečná komunikace mezi procesory v řadě

```
 ...
if ((myid%2)==0  {
    send (....P_{i-1}..);
    recv (....P_{i-1}..);
    send (....P_{i+1}..);
    recv (....P_{i+1}..);
}  else {
   recv (....P_{i-1}..);
   send(....P_{i-1}..);
   recv (....P_{i+1}..);
   send (....P_{i+1}..);
}
 ...
```

```
...
isend (....P_{i-1}..);
isend (....P_{i+1}..);
irecv (....P_{i-1}..);
irecv (....P_{i+1}..);
waitall (4);
...
```

```
...
sendrecv(....P_{i-1}..);
sendrecv(....P_{i+1}..);
...
```

Vícemřížkové metody

- nejdříve se použije hrubá mřížka bodů, iterační proces začne rychle konvergovat;
- v určité fázi se počet bodů zvýší; původní body zůstanou a mezi nimi přibudou další. Počáteční hodnoty v nových bodech se získají interpolací;
- mřížka se může v průběhu řešení zjemňovat, nebo se může jemná a hrubá mřížka střídat;
- hrubší mřížky zohlední vzdálenější jevy rychleji a poskytnou dobrý výchozí bod pro jemnější mřížku.