Algoritmi paraleli
Curs 7

Vlad Olaru

vlad.olaru@fmi.unibuc.ro

Calculatoare si Tehnologia Informatiei

Universitatea din Bucuresti

Sisteme de ecuatii liniare

Fie matricea $\boldsymbol{A \in R^{n \times n}}$***, aflati solutia sistemului:***

$$\boldsymbol{Ax = b; x = \begin{bmatrix} x_1 \\ \dots \\ x_n \end{bmatrix}}$$

Ipoteze de lucru: model de calcul sincron, comunicare instantanee

- **Metode directe**: eliminare gaussiana, factorizare LU etc.
  - afla o ***solutie exacta*** intr-un nr. finit de operatii
  - in general, complexitate $O(n^3)$
  - recomandate pentru dimensiuni mici
  - li se aplica conceptele de complexitate si eficienta discutate

Sisteme de ecuatii liniare

Fie matricea $A \in R^{n \times n}$***, aflati solutia sistemului:***

$$Ax = b; x = \begin{bmatrix} x_1 \\ \dots \\ x_n \end{bmatrix}$$

- **Metode iterative**: Jacobi, Gauss-Seidel
  - nu obtin o solutie exacta in timp finit
  - converg asimptotic catre o solutie exacta
  - in general, dupa un nr mic de pasi (iteratii) ating o solutie cu precizie acceptabila, ceea ce le face preferabile metodelor directe (in special cand n e mare)
  - pt matrici sparse, au cerinte de stocare mai mici decat metodele directe

Sisteme de ecuatii liniare

Fie matricea $\boldsymbol{A \in R^{n \times n}}$***, aflati solutia sistemului:***

$$\boldsymbol{Ax = b; x = \begin{bmatrix} x_1 \\ \cdots \\ x_n \end{bmatrix}}$$

- **Metode iterative**: Jacobi, Gauss-Seidel
  - pt ca pot rula la infinit, nu li se aplica notiunile de complexitate/eficienta
  - o masura mai potrivita este viteza de convergenta a solutiei
  - convergenta e in general *geometrica* (viteza de covergenta e cea a unei *progresii geometrice*)

    secventa de vectori $\{x(t)\}$ a.i. $||x(t) - x^*|| \leq c\rho^t$

    unde x^* este solutia $Ax = b$, iar $c > 0$ si $0 < \rho < 1$ sunt constante
  - cu cat ρ este mai mic, cu atat convergenta e mai rapida

Sisteme de ecuatii liniare cu structura speciala

- sisteme triunghiulare
  - superior: $A_{ij} = 0, i > j$

$$A_{11}x_1 + A_{12}x_2 \cdots + A_{1n}x_n = b_1$$
$$A_{22}x_2 + \cdots A_{2n}x_n = b_2$$
$$\dots$$
$$A_{nn}x_n = b_n$$

  - inferior: $A_{ij} = 0, i < j$

$$A_{11}x_1 = b_1 \Rightarrow x_1 = \frac{b_1}{A_{11}}$$
$$A_{21}x_1 + A_{22}x_2 = b_2 \Rightarrow x_2 = \frac{1}{A_{22}}(b_2 - A_{21}x_1)$$
$$\dots$$
$$A_{n1}x_1 + A_{n2}x_2 \cdots + A_{nn}x_n = b_n \Rightarrow x_n = \frac{1}{A_{nn}}(b_n - A_{n1}x_1 - A_{n2}x_2 - \cdots - A_{nn-1}x_{n-1})$$

Matrici triunghiulare inferioare

- pp A invertibila, vrem sa calculam A^{-1}
- pp $A_{ii} = 1$ si vom generaliza ulterior
- daca $A = I - L$ unde $L_{ij} = 0$ pt. $i \leq j$ => $L^n = 0$
- mai mult

$$A^{-1} = (I + L + L^2 + \ ... + L^{n-1})$$

pt. ca $(I + L + L^2 + \ ... + L^{n-1})(I - L) = I - Ln = I$

- algoritm
  - calculam in paralel $L^2,\ldots,L^{n-1}$
  - adunam rezultatele
  - complexitate $O(log^2 n)$ folosind n^4 procesoare

Matrici triunghiulare inferioare

- algoritm mai eficient

$$A^{-1} = (I + L^{2^{\lceil \log n \rceil - 1}})(I + L^{2^{\lceil \log n \rceil - 2}}) \cdots (I + L^4)(I + L^2)(I + L).$$

pt. ca efectuand inmultirile succesiv ramane $(I + L + L^2 + \ ... + L^{n-1})$

- algoritm:
  - calculam in paralel L^2, L^4,..., $L^{2^{\log n - 1}}$
  - adunam matricea identitate la fiecare matrice de mai sus $O(1)$
  - calculam inmultirile (sunt $O(\log n)$ inmultiri)
  - complexitate $O(log^2 n)$ folosind n^3 procesoare (fara cost de comunicatie)

Matrici triunghiulare cu elemente diagonale diferite de 1

- fie D matrice diagonala a.i. elementele ei sunt elementele de pe diagonala matricii A
- atunci $D^{-1}A$ e matrice triunghiulara cu elemente unitare pe diagonala
- algoritm
  - transformam A in $D^{-1}A$ $O(1)$ cu n^2 procesoare
  - inversam $D^{-1}A$ pt a obtine $A^{-1}D$ $O(log^2 n)$ cu n^3 procesoare
  - inmultim rezultatul din pasul anterior cu D^{-1} la dreapta pt. a obtine A^{-1} $O(1)$ cu n^2 procesoare
- complexitate: $O(log^2 n)$ cu n^3 procesoare

Metoda divide & conquer

- partitionam

$$A = \begin{matrix} A_1 & 0 \\ A_2 & A_3 \end{matrix} \text{ unde } A_1 \text{ are dimensiunea } ceil(\frac{n}{2}) \times ceil(\frac{n}{2})$$

iar A_1 si A_3 sunt triunghiular inferioare

- in aceste conditii

$$A^{-1} = \begin{bmatrix} A_1^{-1} & 0 \\ -A_3^{-1}A_2A_1^{-1} & A_3^{-1} \end{bmatrix}$$

- algoritm (pt $n > 1$)
  - inversam in paralel A_1 si A_3
  - inmultim $A_3^{-1}A_2$ — $O(\log n)$ cu n^3 procesoare
  - inmultim rezultatul pasului anterior cu A_1^{-1} — $O(\log n)$ cu n^3 procesoare

Complexitate metoda divide & conquer

- $T(n) = timp\ inversare\ matrice\ n\ x\ n$

=> $T(n) = T\left(ceil\left(\frac{n}{2}\right)\right) + O(\log n)$

=> $T(n) = O(log^2 n)$ folosind n^3 procesoare

- demonstratie: divide & conquer in $O(\log n)$ pasi

$$T(n) - T\left(ceil\left(\frac{n}{2}\right)\right) = O(\log n)$$

$$T\left(ceil\left(\frac{n}{2}\right)\right) - T\left(ceil\left(\frac{n}{4}\right)\right) = O(\log ceil(\frac{n}{2}))$$

…

Problemele metodelor bazate pe inversare de matrici

- exista solutii mai simple fara sa fie nevoie sa aflam inversa matricii?
- daca luam in calcul costul comunicatiei, complexitatea creste mult peste $O(log^2 n)$
- daca tinem cont si de nr cubic de procesoare => solutii interesante teoretic, dar nepractice

Eliminare gaussiana (inferior triunghiular), “back substitution”

- ecuatia i din sistemul triunghiular inferior:

$$A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{ii}x_i = b_i$$

- pp. folosirea a n procesoare
- pp ca la inceputul fazei i valorile $\{x_1, \cdots, x_{i-1}\}$ si expresiile

$$A_{j1}x_1 + A_{j2}x_2 + \cdots + A_{ji-1}x_{i-1}$$

sunt cunoscute pt. fiecare $j \geq i$ din ec. precedente

=> procesorul i calculeaza

$$x_i = \frac{1}{A_{ii}}(b_i - A_{i1}x_1 - \cdots - A_{ii-1}x_{i-1})$$

cf. formulei de mai sus

Eliminare gaussiana (inferior triunghiular), “back substitution”

- apoi, fiecare procesor j, unde $j \geq i + 1$, calculeaza

$$A_{j1}x_1 + A_{j2}x_2 + \cdots + A_{ji}x_i$$

adunand $A_{ji}x_i$ la expresia anterior cunoscuta

$$A_{j1}x_1 + A_{j2}x_2 + \cdots + A_{ji-1}x_{i-1}$$

- algoritmul se opreste la sf fazei n, cand toate variabilele $x_1, x_2, \ldots xn$ sunt calculate
- complexitate:

- calculul x_i necesita $O(1)$ => complexitate $O(n)$ cu n procesoare si comunicatie instantanee

Analiza comparativa

- eficienta

$$E(n) = \frac{T^*(n)}{pT_p(n)}$$

unde p e nr de procesoare iar n dimensiunea problemei

- obs: $Ax = b$ necesita cel putin n^2 pasi (A are n^2 elemente)
- implementarea seriala back substitution este $O(n^2)$

$\Rightarrow$ $T^*(n) = \Theta(n^2)$

$\Rightarrow$ eficienta algoritmilor bazati pe inversare de matrici este $O(\frac{1}{nlog^2n})$ iar pt back substitution este $\Theta(1)$

$\Rightarrow$ back substitution e mai lent, dar mai eficient !

Celelalte metode necesita un nr disproportionat de mare de procesoare fata de imbunatatirea pe care o produc asupra accelerarii.

Back substitution pt invesarea de matrici triunghiulare

- $AA^{-1} = I$ => x^i, coloana i a matricii A^{-1} satisface

$$Ax^i = e^i$$

unde e^i este al i -lea vector unitate

- algoritm: se rezolva n sisteme $Ax^i = ei$
- complexitate
  - $O(n)$ folosind n^2 procesoare pt rezolvarea celor n probleme in paralel
  - $O(\mathrm{n}^2)$ folosind n procesoare pt rezolvarea celor n probleme una dupa alta

Implementarea back substitution pe topologii particulare

- consideram un lant liniar cu n procesoare
- procesorul i primeste linia i a matricii A si componenta i a vectorului b
- pipelining-ul calculului cu computatia permite un timp de executie de $O(n)$ => costul comunicatiei poate cel mult afecta timpul de executie in limita unui factor constant
- daca acest factor e mare (comparabil cu n), se poate mari granularitatea prin asignarea mai multor linii unui nr mai mic de procesoare
- pe hipercub penalizarea comunicatie va fi mai mica, asa cum am vazut la maparea lanturilor liniare pe hipercuburi

Back substitution pe lant liniar

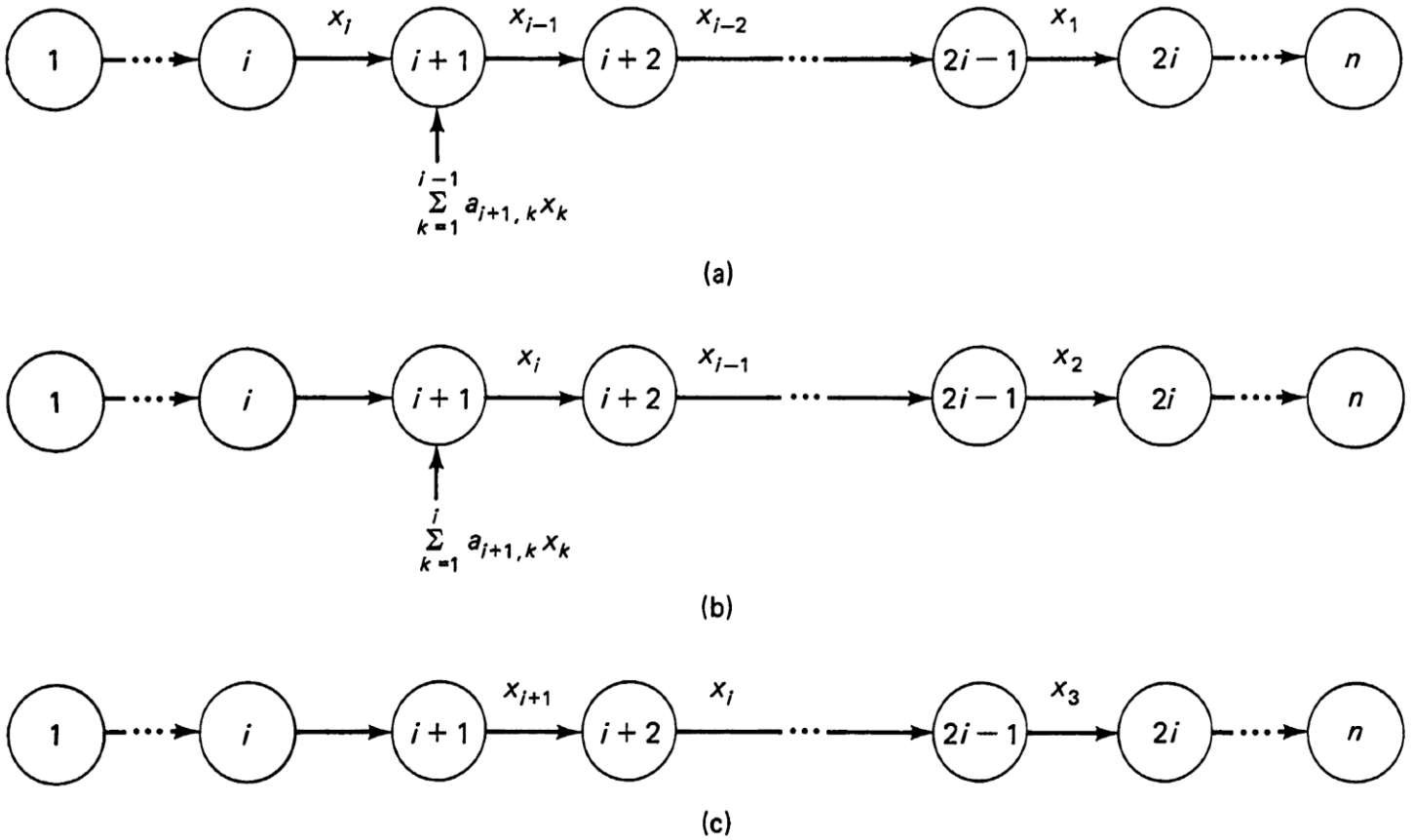
(a) procesorul $i + 1$ a primit x_1, …, x_{i-1}
si a evaluat suma partiala

(b) cand x_i a fost primit de catre procesorul $i + 1$ este trimis la dreapta (procesorul $i + 2$) si noua suma partiala este calculata

Apoi se calculeaza x_{i+1}

(c) x_{i+1} se transmite procesorului $i + 2$

Calculul x_i si x_{i+1} este *O(1)* iar timpul total al algoritmului este proportional cu n



Sisteme tridiagonale

$$A = \begin{bmatrix} 1 & -1 & 0 \\ -2 & 2 & 1 \\ 0 & -2 & 3 \end{bmatrix}; g = [A_{11} \cdots A_{nn}]; h = [A_{12} \cdots A_{n-1n}];$$

$$f = [A_{21} \cdots A_{nn-1}];$$

- Sisteme tridiagonale $Ax = b$
  - $A_{ij} = 0, |i - j| > 1$

$$g_1 x_1 + h_1 x_2 = b_1$$
$$f_i x_{i-1} + g_i x_i + h_i x_{i+1} = b_i, \quad i = 2, 3, \cdots, n-1$$
$$f_n x_{n-1} + g_n x_n = b_n$$

unde g_i sunt elementele de pe diagonala matricii A, iar f_i (respectiv h_i) sunt elementele lui A de sub (respectiv de deasupra) diagonalei

Sisteme tridiagonale rezolvate prin metoda reducerii par-impare

Idee: daca $g_i \neq 0$ putem rezolva x_i pentru fiecare i impar si substituim expresia lui x_i in restul ecuatiilor => un sistem de ecuatii x_i cu i par

$\Rightarrow$ sistem tridiagonal cu jumatate din variabile

$\Rightarrow$ apoi aplicam procedura recursiv

Conventie: $x_0 = x_{n+1} = 0$ pt a putea calcula x_i ca mai jos si pt $i = 1$ respectiv $i = n$

$$x_i = \frac{1}{g_i}(b_i - f_i x_{i-1} - h_i x_{i+1})$$

Deci

$$x_{i-1} = \frac{1}{g_{i-1}}(b_{i-1} - f_{i-1} x_{i-2} - h_{i-1} x_i)$$

$$x_{i+1} = \frac{1}{g_{i+1}}(b_{i+1} - f_{i+1} x_i - h_{i+1} x_{i+2})$$

Sisteme tridiagonale rezolvate prin metoda reducerii par-impare

$$g_1 x_1 + h_1 x_2 = b_1$$
$$f_i x_{i-1} + g_i x_i + h_i x_{i+1} = b_i, \qquad i = 2, 3, \cdots, n-1$$
$$f_n x_{n-1} + g_n x_n = b_n$$

Consideram $x_0 = x_{n+1} = 0$

$$x_i = \frac{1}{g_i}(b_i - f_i x_{i-1} - h_i x_{i+1})$$

Deci

$$x_{i-1} = \frac{1}{g_{i-1}}(b_{i-1} - f_{i-1} x_{i-2} - h_{i-1} x_i)$$
$$x_{i+1} = \frac{1}{g_{i+1}}(b_{i+1} - f_{i+1} x_i - h_{i+1} x_{i+2})$$

Inlocuim in a doua ecuatie de mai sus:

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

Sisteme tridiagonale rezolvate prin metoda reducerii par-impare

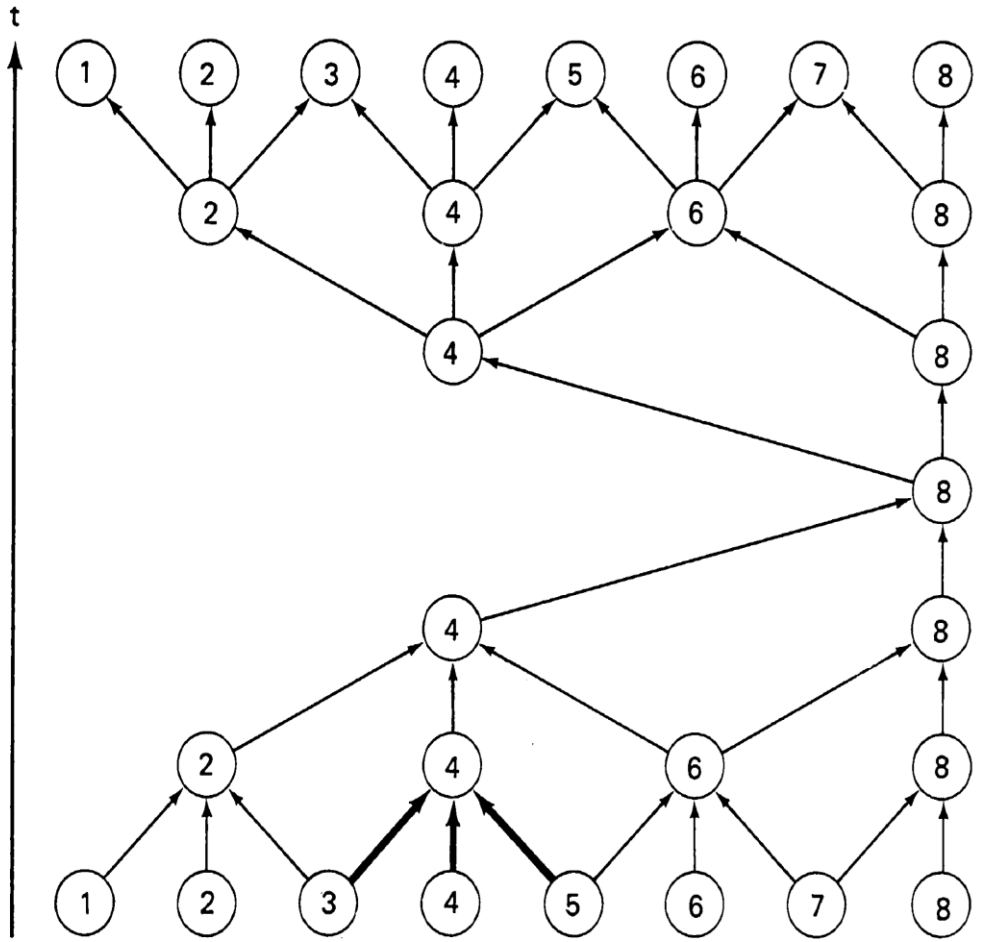
Prin simplificare rezulta:

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

- obtinem un alt sistem tridiagonal cu $n/2$ variabile: $x_2, \cdots, x_{2*floor(\frac{n}{2})}$!
- prin eliminari succesive injumatatim recursiv dimensiunea pana obtinem un sistem cu o singura variabila care se rezolva direct
- apoi, se parcurge procesul in sens invers pt a calcula valorile variabilelor eliminate

Vizualizarea metodei reducerii par-impare

- n = 8
- prima faza: eliminare variabile x_1, x_3, x_5, x_7
- se obtin valorile x_2, x_4, x_6, x_8
- faza a doua: x_2 si x_6 sunt eliminate
- apoi x_4 e eliminat si ramane o singura ecuatie cu o singura necunoscuta x_8
- dupa evaluarea lui x_8, celelalte variabile se evalueaza in sens invers
- ex: odata ce s-au evaluat x_2, x_4, x_6, se obtin imediat si valorile lui x_1, x_3, x_5, x_7
- obs: arcele din diagrama indica dependenta de date (ex: arcele ingrosate reprezinta coeficientii f_i, g_i, h_i pt $i = 3, 4, 5$)



Complexitatea metodei reducerii par-impare pt sisteme tridiagonale

- la fiecare etapa, nr variabilelor se reduce aproxmativ la jumatate
- dupa $O(\log n)$ pasi ramane doar o variabila de evaluat
- in fiecare etapa, e nevoie sa se calculeze coeficientii sistemului redus => 4 pasi cu $O(n)$ procesoare
- similar pentru etapa de evaluare in sens invers
- per total, complexitatea este de $O(\log n)$ cu $O(n)$ procesoare
- Obs: nr total de operatii este $O(n)$ pt. ca in fiecare etapa se injumatatesc calculele

=> exista un algoritm secvential de complexitate $O(n)$ (optimal, de fapt, pt ca sunt necesari cel putin n pasi pt a citi datele)

=> $T^*(n) = \Theta(n)$ iar eficienta este $\Theta(1/\log n)$

Metoda reducerii par-impare pe lant liniar

- n procesoare, fiecare procesor i stocheaza linia i din matrice si componenta i din vectorul b si va calcula variabila x_i
- evaluarea fazei pare se face in $O(1)$ pt. ca procesorul i are nevoie doar de coeficientii procesoarelor vecine
- dupa k reduceri, sistemul contine doar variabile x_i unde i este un multiplu intreg de 2^k => procesorul $i2^k$ trebuie sa comunice cu procesorul $(i+1)2^k$
- in ultima etapa, costul comunicarii este $\Omega(n)$ => complexitate $\Omega(n)$, la fel ca si algoritmul secvential !
- Obs: se putea infera si din faptul ca valoarea lui x_1 depinde de date stocate pe procesoare la distanta $\Theta(n)$

Metoda reducerii par-impare pe lant linear cu cresterea granularitatii

- p procesoare ($p < n$), fiecare procesor i stocheaza linia n/p linii din matrice, componentele corespunzatoare din vectorul b si va calcula n/p variabile
- in primele N = $floor\left(\log\left(\frac{n}{p}\right)\right)$ etape calculele sunt locale=> $O\left(\frac{n}{p}\right)$ operatii aritmetice si f. putina comunicatie
- dupa N etape => un sistem tridiagonal cu $O(p)$ variabile
  - se rezolva trimitand toti coeficientii catre un singur procesor
  - procesorul rezolva sistemul si trimite rezultatele prin broadcast inapoi catre cele p procesoare

  => complexitate $O(p)$ pt calcule si $O(p)$ pt comunicatie
- deci, timpul total este $O\left(\frac{n}{p}\right) + O(p)$
- optimizand dupa p, valoarea optimala este $p = \Theta(\sqrt{n})$ => timp de executie optimal este $O(\sqrt{n})$
  - superior timpului serial $O(n)$ dar inferior valorii pt comunicatie instantanee $O(\log n)$

Metoda reducerii par-impare pe hipercub

- n procesoare
- mapam un lant liniar pe hipercub folosind coduri Gray reflectate (RGC)
- cand folosim maparea RGC, procesoare aflate la distanta logica 2^k in lant se mapeaza la distanta fizica 2 in hipercub

  => toate perechile de procesoare $\left(i2^k, (i+1)2^k\right)$ pot comunica simultan in 2 pasi

$\Rightarrow$ fiecare etapa din reducerea par-impara consuma $O(1)$pt comunicatie

$\Rightarrow$ in total, complexitatea ramane cea teoretica de $O(\log n)$ cand se folosesc n procesoare

Obs: in practica, e posibil ca totusi operatiile de comunicare sa fie dominante comparativ cu calculul (chiar daca sunt $O(1)$)

=> trebuie crescuta granularitatea pt a scadea penalizarea comunicatiei

Metode paralele directe pentru sisteme de ecuatii liniare

- A matrice $n \times n$
- aplica o serie de transformari sistemului de ecuatii liniare a.i. matricea A sa devina triunghiulara si sa aplica *back substitution*
- aceste transformari constau intr-o serie de inmultiri la stanga cu matrici a.i. sistemul devine

$$(M^{(k)}M^{(k-1)} \dots M^{(1)}A)x = (M^{(k)}M^{(k-1)} \dots M^{(1)}b)$$

- matricile M se aleg a.i.
  - inmultirile cu o matrice arbitrara (initial A) sa fie rapide computational
  - produsul final $M^{(k)}M^{(k-1)} \dots M^{(1)}A$ sa fie o matrice triunghiulara
- consecinta: sistemul se poate rezolva in $O(n)$ cu n procesoare
- bonus: $A^{-1} = (M^{(k)}M^{(k-1)} \dots M^{(1)}A)^{-1}(M^{(k)}M^{(k-1)} \dots M^{(1)})$
  - pt. ca $(M^{(k)}M^{(k-1)} \dots M^{(1)}A)$ e triunghiulara, se poate inversa folosind back substitution in $O(n)$ folosind n^2 procesoare sau in $O(n^2)$ folosind n procesoare

Eliminare Gaussiana (cazul general)

- principiu: variabila x_i este exprimata ca o functie de $x_{i+1}, \dots, x_n$ si e eliminata din sistem => dupa $n-1$ pasi, ramane o singura ecuatie in x_n
- definitie recursiva a algoritmului

$$C^{(0)} = A$$

$$C^{(i)} = M^{(i)} \dots M^{(1)} A$$

- pp. $C^{(i-1)}$ calculata pt. $1 \leq i \leq n-1$ si elementele subdiagonale ale coloanei j sunt zero pt. $j < i$
- aratam cum sa construim $M^{(i)}$ a.i.

$$C^{(i)} = M^{(i)} C^{(i-1)}$$

$$C^{(i-1)} = \begin{bmatrix} * & \cdots & \cdots & \cdots & \cdots & * \\ 0 & \ddots & & & & \vdots \\ \vdots & \ddots & \ddots & & & \\ & & 0 & * & \cdots & * \\ \vdots & & \vdots & \vdots & & \\ 0 & \cdots & 0 & * & \cdots & * \end{bmatrix} \begin{matrix} \\ \\ \\ i \\ \\ \\ \end{matrix}$$

$$\begin{matrix} & & & i & & \end{matrix}$$

Eliminare Gaussiana fara pivotare

- pp. “pivotul” $C^{(i-1)}{}_{ii} \neq 0$
- fie $M^{(i)} = I - N^{(i)}$ unde elementele nenule ale lui $N^{(i)}$ sunt

$$N^{(i)}{}_{ji} = C^{(i-1)}{}_{ji} \,/\, C^{(i-1)}{}_{ii} \qquad j > i$$

$$N^{(i)} = \begin{bmatrix} 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \ddots & & & & & \vdots \\ \vdots & \ddots & \ddots & & & & \\ & & 0 & 0 & \cdots & \cdots & 0 \\ & & \vdots & * & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & * & 0 & \cdots & 0 \end{bmatrix} i$$

$$\qquad\qquad\qquad i$$

Eliminare Gaussiana fara pivotare

- pp. “pivotul” $C^{(i-1)}{}_{ii} \neq 0$
- fie $M^{(i)} = I - N^{(i)}$ unde elementele nenule ale lui $N^{(i)}$ sunt

$$N^{(i)}{}_{ji} = C^{(i-1)}{}_{ji} / C^{(i-1)}{}_{ii} \qquad j > i$$

- atunci, $C^{(i)} = (I - N^{(i)})\, C^{(i-1)}$ pt. ca
  - primele $i - 1$ coloane ale lui $N^{(i)}$ sunt zero => primele $i - 1$ coloane ale lui $C^{(i)}$ sunt la fel cu cele ale lui $C^{(i-1)}$ (adica zero)
  - elementele subdiagonale ale lui $C^{(i)}$ din coloana i, $C^{(i)}{}_{ji}$ cu $j > i$ sunt

$$C^{(i)}{}_{ji} = C^{(i-1)}{}_{ji} - \frac{C^{(i-1)}{}_{ii}}{C^{(i-1)}{}_{ii}} C^{(i-1)}{}_{ii} = 0$$

- deci $C^{(i)}$ e de aceeasi forma cu $C^{(i-1)}$ (in particular, superior triunghiulara)
- Obs: metoda esueaza cand pivotul e zero; chiar pt valori mici ale pivotului pot exista probleme numerice

=> metoda se foloseste in special pt. matrici A simetrice (pt care $C^{(i-1)}{}_{ii}$ nu e niciodata zero daca A e inversabila)

Eliminare Gaussiana cu pivotare

- pp. schimbarea a doua linii (sau coloane) ale matricii $C^{(i-1)}$ a.i. elementul diagonal de pe pozitia i este nenul si preferabil mare
- metoda uzuala, pivotarea pe linii
  - calculeaza $C^{(i-1)}$
  - gaseste k a.i. $|C^{(i-1)}{}_{ki}| = \max_{j \geq i} |C^{(i-1)}{}_{ji}|$
  - schimba liniile k si i intre ele
  - apoi calculeaza $C^{(i)}$ ca mai inainte

Comparatie sisteme tridiagonale – eliminare Gaussiana

- metoda reducerii par-impare e un caz particular al eliminarii gaussiene
- se elimina variabile prin inlocuirea lor cu functii dependente de variabilele care raman
- structura speciala tridiagonala permite eliminarea a jumatate din variabile intr-un singur pas
- variabilele nu sunt eliminate in ordine (intai se elimina variabilele impare) => e vorba de o eliminare Gaussiana pe o permutare a matricii A care muta liniile impare la inceputul matricii
  - consecinta: variabilele impare sunt primele eliminate
- reducerea par-impara e echivalenta cu eliminarea gaussiana fara pivotare pt. ca elimina variabilele intr-o anumita ordine
  - daca e nevoie de operatie de pivotare, metoda reducerii par-impare nu mai e aplicabila (pivotarea afecteaza structura tridiagonala)

Implementarea paralela a eliminarii Gaussiene fara pivotare

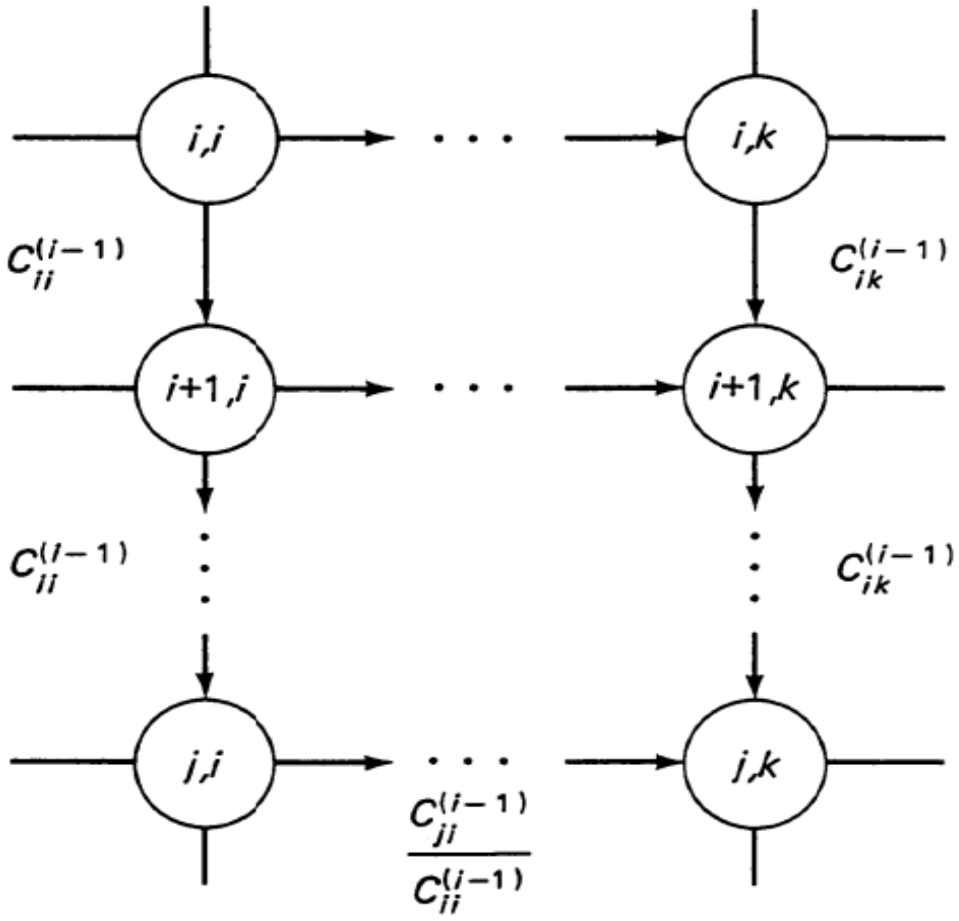
- ignoram costul comunicatiei

$$C_{jk}^{(i)} = C_{jk}^{(i-1)} - \sum_{\ell=1}^{n} N_{j\ell}^{(i)} C_{\ell k}^{(i-1)} = C_{jk}^{(i-1)} - N_{ji}^{(i)} C_{ik}^{(i-1)} = C_{jk}^{(i-1)} - \frac{C_{ji}^{(i-1)}}{C_{ii}^{(i-1)}} C_{ik}^{(i-1)}.$$

- folosind n^2 procesoare, toate elementele $C^{(i)}$ se pot calcula in $O(1)$ (o inmultire, o impartire si o scadere)
- deci $C^{(n-1)}$ se poate calcula in cca $3n$ pasi folosind n^2 procesoare
- cu n procesoare => $O(n^2)$
- eficienta: $\Theta(1)$ pt benchmark serial $T^*(n) = \Theta(n^3)$ si n^2 sau n procesoare

Eliminarea Gaussiana fara pivotare pe grid

- pp. grid patrat cu n^2 procesoare => $O(n)$
- fiecare procesor se asociaza cu un element al matricilor manipulate
- datele se transmit ca in figura alaturata
- o etapa are nevoie de $\Theta(n)$ pasi => daca noua etapa porneste abia dupa ce s-a incheiat etapa anterioara => $\Theta(n^2)$ pasi
- cu interleaving comunicatie/calcul => $\Theta(n)$ pasi



Eliminarea Gaussiana fara pivotare pe grid

- pp. grid patrat cu n^2 procesoare => $O(n)$
- fiecare procesor se asociaza cu un element al matricilor manipulate
- pt a mentine timpul de executie liniar, e nevoie de pipelining-ul comunicatiei cu calculul
  - calculele fazei $i + 1$ incep inainte ca toate mesajele trimise in faza i sa fie primite
  - astfel, timpul de comunicatie este de acelasi ordine de marime cu timpul de calcul
  - altfel, timpul de executie e patratic
  - daca timpul de necesar unei singure comunicatii e semnificativ mai mare decat timpul necesar unei singure operatii de calcul, e nevoie de cresterea granularitatii prin folosirea unui nr mai mic de procesoare

Eliminarea Gaussiana fara pivotare pe hipercub si lant liniar

- pp. hipercub cu $\Theta(n^2)$ procesoare => $O(n)$ pt ca grid-ul se poate mapa pe hipercub
- un lant liniar cu n procesoare poate simula un grid $n \times n$ cu o reducere de $O(n)$ in termeni de viteza => eliminarea Gaussiana se poate executa pe un lant liniar in $O(n^2)$

Finalul metodei de rezolvare bazata pe eliminare Gaussiana

- presupune rezolvarea unui sistem triunghiular
- daca folosim back substitution, metoda seriala e $O(n^2)$ => neglijabil fata de $O(n^3)$ cat necesita eliminarea Gaussiana seriala
- in paralel insa, cand se folosesc n^2 procesoare, back substitution necesita $O(n)$, comparabil cu eliminarea Gaussiana => e importanta o implementare paralela eficienta pt back substitution

Sumar eliminare Gaussiana

TABLE 2.1 Bounds on the timing of Gaussian elimination for several architectures. The upper bounds are the same as those obtained for the same number of processors, if communication is assumed instantaneous.

Number of Processors	Architecture	Without Pivoting	With Pivoting
n^2	Hypercube	$O(n)$	$O(n \log n)$
n^2	Mesh	$O(n)$	$\Omega(n^{4/3})$
$n^2/\log n$	Hypercube	$O(n \log n)$	$O(n \log n)$
n	Hypercube	$O(n^2)$	$O(n^2)$
n	Linear Array	$O(n^2)$	$O(n^2)$

Sisteme de ecuatii liniare
-Metode iterative-

Sisteme de ecuatii liniare

Fie matricea $\boldsymbol{A \in R^{n \times n}}$***, aflati solutia sistemului:***

$$\boldsymbol{Ax = b}; \qquad \boldsymbol{x} = \begin{bmatrix} \boldsymbol{x_1} \\ \dots \\ \boldsymbol{x_n} \end{bmatrix}$$

- **Metode directe**: Eliminare gaussiana, Factorizare LU etc.
  - afla o ***solutie exacta*** intr-un nr. finit de operatii
  - in general, complexitate $O(n^3)$; recomandate pentru dim. mici
- **Metode iterative**: **Jacobi, Gauss-Seidel**
  - afla o solutie inexacta, de precizie acceptabila (converg asimptotic catre una exacta)
  - complexitate = numar de iteratii (din rata de convergenta)
    - ex: $\left|\left|x^{k+1} - x^*\right|\right| \leq \frac{1}{2}\left|\left|x^k - x^*\right|\right| \Rightarrow \left|\left|x^k - x^*\right|\right| \leq \left(\frac{1}{2}\right)^k \left|\left|x^0 - x^*\right|\right|$
    - $\left|\left|x^k - x^*\right|\right| \leq \epsilon \Rightarrow k \geq 0(\log\left(\frac{\left|\left|x^0 - x^*\right|\right|}{\epsilon}\right))$

Sisteme de ecuatii liniare

$$Ax = b$$

Este echivalent cu

$$x_1 = -\frac{1}{A_{11}}(A_{12}x_2 + \cdots + A_{1n}x_n - b_1)$$

$$x_2 = -\frac{1}{A_{22}}(A_{21}x_1 + \cdots + A_{1n}x_n \ - b_2)$$

$$\ldots$$

$$x_n = -\frac{1}{A_{nn}}(A_{n1}x_1 + \cdots + A_{nn-1}x_{n-1} \ - b_n)$$

Sisteme de ecuatii liniare

- daca $A_{ii} \neq 0$ si $\forall j, j \neq i, xj$ cunoscut => x_i
- daca se cunosc insa doar valori aproximative (estimate) ale componentelor x_j, atunci se poate calcula o *valoare estimata* a lui x_i
- calculul valorilor estimate se poate face simultan pt toate componentele lui x
- rezultatul este un algoritm iterativ Jacobi
  - se porneste cu o valoare initiala $x(0) \in R^n$
  - se evalueaza $x(t), t = 1,2, \ldots$ cf iteratiei

$$x_i(t+1) = -\frac{1}{A_{ii}}\left(\sum_{j \neq i} A_{ij} x_i(t) - b_i\right)$$

  - daca secventa converge la o valoare, atunci acea valoare e solutia sistemului de ecuatii liniare
  - Obs: e perfect posibil ca secventa sa nu convearga

Algoritmul Jacobi

$$Ax = b$$

Consideram un sir iterativ $x(0), x(1), \ldots \in R^n$, daca $x(t) \to x^*, t \to \infty$ => x^* e solutia sistemului de ecuatii liniare

La pasul $t + 1$:

$$x_1(t+1) = -\frac{1}{A_{11}}(A_{12}x_2(t) + \cdots + A_{1n}x_n(t) - b_1)$$

$$x_2(t+1) = -\frac{1}{A_{22}}(A_{21}x_1(t) + \cdots + A_{1n}x_n(t) - b_2)$$

...

$$x_n(t+1) = -\frac{1}{A_{nn}}(A_{n1}x_1(t) + \cdots + A_{nn-1}x_{n-1}(t) - b_n)$$

Algoritmul Jacobi pt iteratie convergenta

Fie sistemul: $\begin{cases} 2x_1 - x_2 = 0 \\ -x_1 + 2x_2 = 0 \end{cases}$

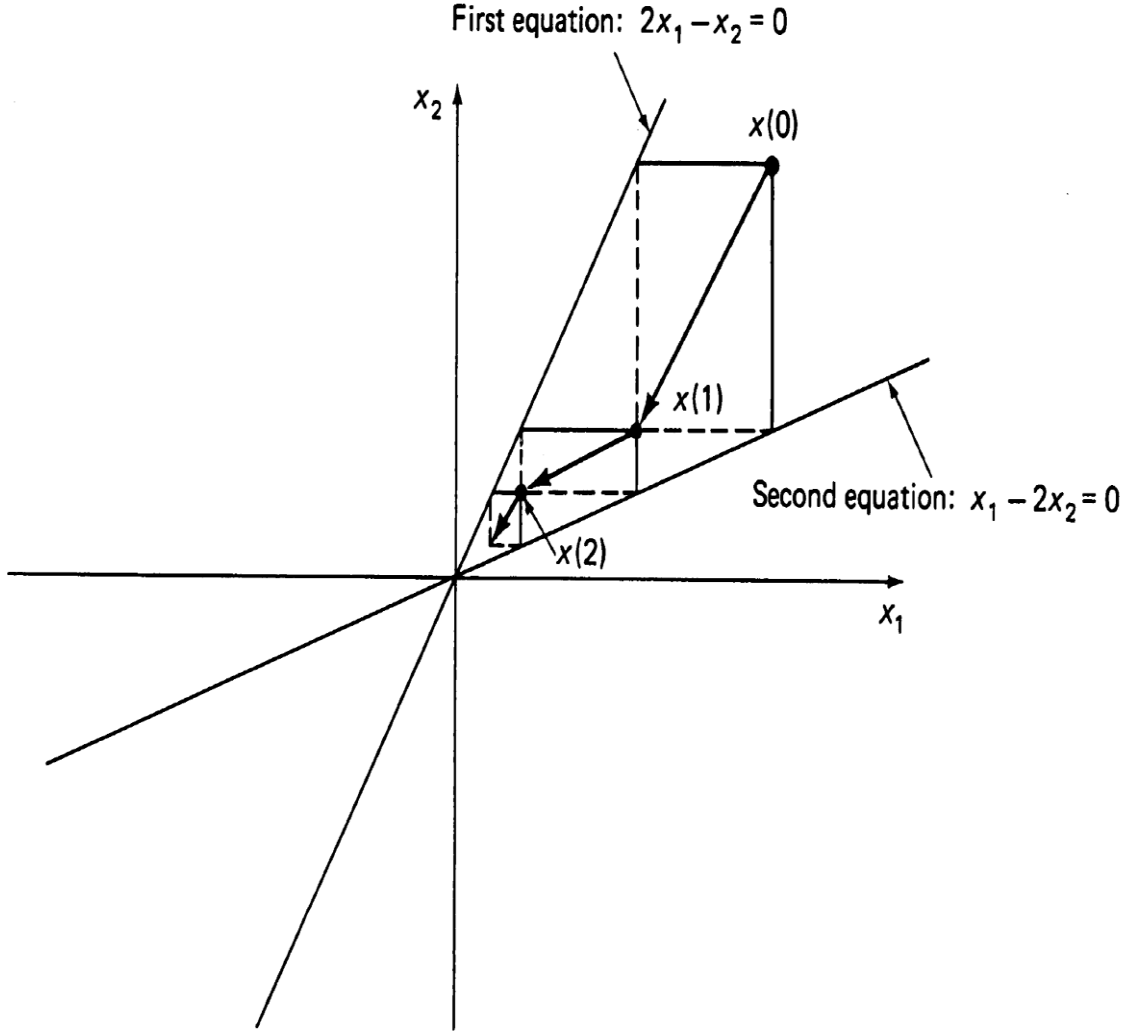
$\begin{cases} x_1 = \frac{x_2}{2} \\ x_2 = \frac{x_1}{2} \end{cases}$ Pornind din $x(0) = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$, iteram:

$$x_1(1) = \frac{x_2(0)}{2} = 2$$

$$x_2(1) = \frac{x_1(0)}{2} = 2$$

$$x_1(t) = \frac{x_2(t-1)}{2} = x_2(0)\left(\frac{1}{2}\right)^t \to 0$$

$$x_2(t) = \frac{x_1(t-1)}{2} = x_1(0)\left(\frac{1}{2}\right)^t \to 0$$



Algoritmul Jacobi pt iteratie divergenta

Fie sistemul: $\begin{cases} -x_1 + 2x_2 = 0 \\ 2x_1 - x_2 = 0 \end{cases}$

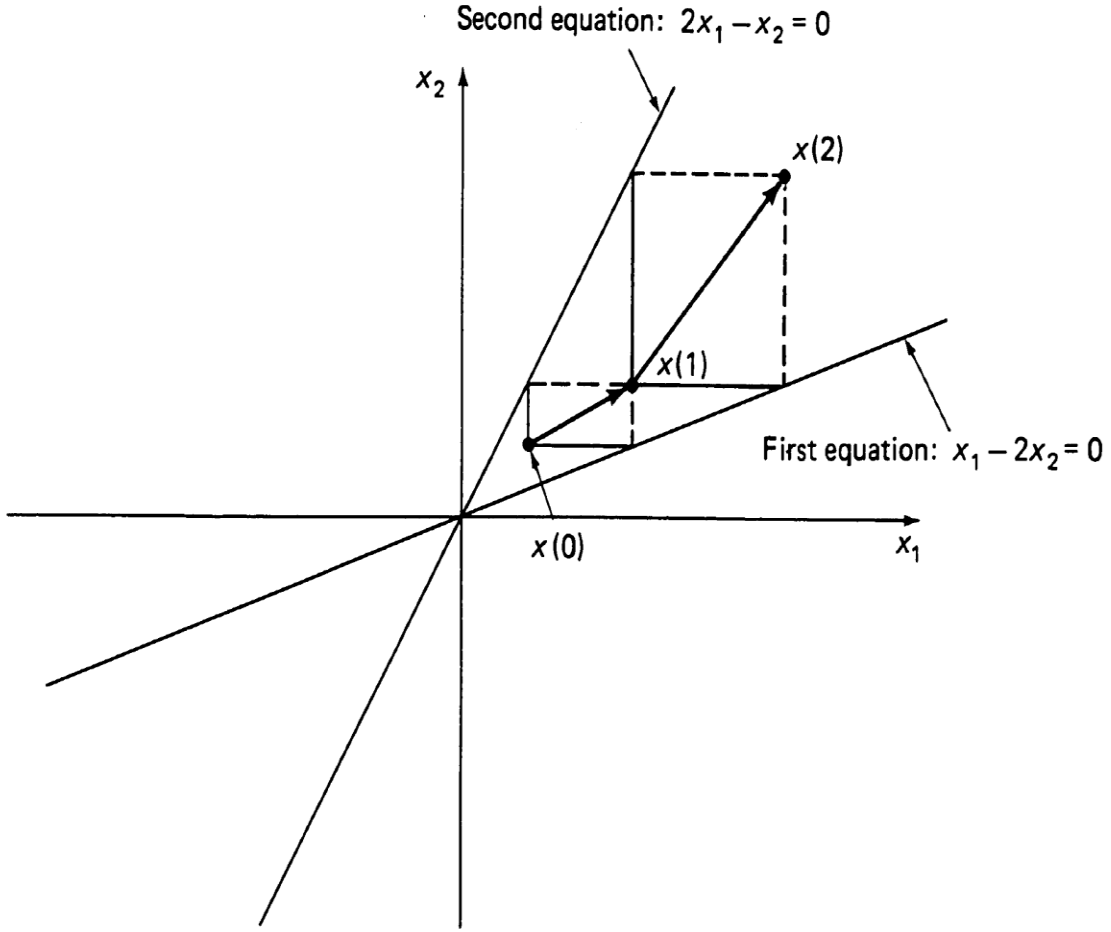
Matriceal: $\begin{bmatrix} -1 & 2 \\ 2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Pornind din $x(0) = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$, iteram:

$$x_1(1) = 2x_2(0) = 8$$
$$x_2(1) = 2x_1(0) = 8$$

$$x_1(2) = 2x_2(1) = 16$$
$$x_2(2) = 2x_1(1) = 16$$

$$x_1(t) = 2x_2(t-1) = 2^t x_2(0)$$
$$x_2(t) = 2x_1(t-1) = 2^t x_1(0)$$



Algoritmul Jacobi

- O matrice A este **diagonal dominanta** (pe linii) daca:

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

- Exemplu: $\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$ este diagonal dominanta, $\begin{bmatrix} -1 & 2 \\ 2 & -1 \end{bmatrix}$ nu este!

Teorema: Daca A este diagonal dominanta, atunci sirul generat de algoritmul Jacobi:

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

converge.

Algoritmul Jacobi

$$Ax = b$$

Este echivalent cu

$$x_i = -\frac{1}{A_{ii}}\left(\sum_{j \neq i} A_{ij}x_j - b_i\right)$$

Idea algoritmului Jacobi:

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

Observati ca la pasul $t + 1$ fiecare $x_i(t + 1)$ poate fi evaluat complet paralel!

Este necesar ca fiecare procesor sa aiba $x(t)$!

Algoritmul Jacobi paralel

- am discutat deja paralelizarea metodelor iterative
- cea mai mare parte a calculului revine la inmultiri de matrici cu vectori
- ne concentram pe sisteme distribuite (message passing)
- ipoteze de lucru
  - pp n procesoare disponibile, procesorul i calculeaza $x_i(t)$ la fiecare iteratie t
  - procesorul i stocheaza linia i a matricii A
- pt a calcula $x_i(t+1)$ procesorul i are nevoie de valorile $x_j(t)$ calculate de procesoarele j la iteratia t pt care $A_{ii} \neq 0$

Algoritmul Jacobi paralel

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

Obs: la pasul $t+1$ fiecare $x_i(t+1)$ poate fi calculat complet paralel!

La fiecare iteratie t:

- P_i calculeaza $x_i(t)$, produs scalar $O(n)$ (pp. A densa, fara structura speciala, sparse)
- difuzare generala (broadcast multinod): P_i difuzeaza $x_i(t)$
  - $O(n)$ pe lant liniar sau grid, $O(\frac{n}{\log n})$ pe hipercub
- procentul de timp consumat in comunicatie scade cu valoarea lui n pt hipercub si ramane constant pt lant liniar
  - daca acest factor constant e mare, crestem granularitatea (mai multe linii stocate pe mai putine procesoare)

Algoritmul paralel Jacobi pt matrici sparse

- matricea A e sparse
- structura rara a matricii e descrisa de un graf $G = (V, E)$ unde

$$E = \{(i,j) | i \neq j \text{ si } Aji \neq 0 \; a.i. \; procesorul \; i \; comunica \; cu \; j\}$$

- acesta e un graf de dependente asa cum am vazut anterior
- implementari paralele eficiente implica maparea eficienta a acestui graf de dependente pe topologii speciale a.i.
  - toate comunicatiile au loc intre procesoare vecine in topologie
  - penalizarea de comunicare e minimala

Algoritmul Gauss-Seidel

$$Ax = b$$

sau echivalent

$$x_i = -\frac{1}{A_{ii}}\left(\sum_{j \neq i} A_{ij} x_j - b_i\right)$$

- daca o iteratie Jacobi

$$x_i(t+1) = -\frac{1}{A_{ii}}\left(\sum_{j \neq i} A_{ij} x_i(t) - b_i\right)$$

se executa secvential, la momentul evaluarii $x_i(t+1)$ exista deja cateva valori estimate noi $x_j(t+1)$ pt $j < i$

Idee: folosirea acestor noi valori estimate x_j pt $j < i$ in estimarea lui x_i

Algoritmul Gauss-Seidel (2)

Idee:

- se porneste cu o valoare initiala $x(0) \in R^n$
- se evalueaza $x(t), t = 1,2, \ldots$ cf iteratiei

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

Obs:

- $x_i(t+1)$ depinde de $x_j(t+1)\ pt. j < i$
- se poate paraleliza doar in cazuri in care matricea A are structura speciala (e.g. A rara)

Algoritmul Gauss-Seidel (3)

$$Ax = b$$

Este echivalent cu

$$x_1 = -\frac{1}{A_{11}}(A_{12}x_2 + A_{13}x_3 \cdots + A_{1n}x_n - b_1)$$

$$x_2 = -\frac{1}{A_{22}}(A_{21}x_1 + A_{23}x_3 \cdots + A_{1n}x_n - b_2)$$

$$x_3 = -\frac{1}{A_{33}}(A_{31}x_1 + A_{32}x_2 \cdots + A_{1n}x_n - b_3)$$

$$\ldots$$

$$x_n = -\frac{1}{A_{nn}}(A_{n1}x_1 + A_{n2}x_2 \cdots + A_{nn-1}x_{n-1} - b_n)$$

Algoritmul Gauss-Seidel (4)

$$Ax = b$$

Este echivalent cu

$$x_1(t+1) = -\frac{1}{A_{11}}(A_{12}x_2(t) + A_{13}x_3(t) \cdots + A_{1n}x_n(t) - b_1)$$

$$x_2(t+1) = -\frac{1}{A_{22}}(A_{21}x_1(t+1) + A_{23}x_3(t) \cdots + A_{1n}x_n(t) - b_2)$$

$$x_3(t+1) = -\frac{1}{A_{33}}(A_{31}x_1(t+1) + A_{32}x_2(t+1) \cdots + A_{1n}x_n(t) - b_3\ )$$

...

$$x_n(t+1) = -\frac{1}{A_{nn}}(A_{n1}x_1(t+1) + A_{n2}x_2(t+1) \cdots + A_{nn-1}x_{n-1}(t+1) - b_n)$$

Algoritmul Gauss-Seidel pt iteratie convergenta

Fie sistemul $\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$$x_1(t+1) = \frac{x_2(t)}{2}$$

$$x_2(t+1) = \frac{x_1(t+1)}{2}$$

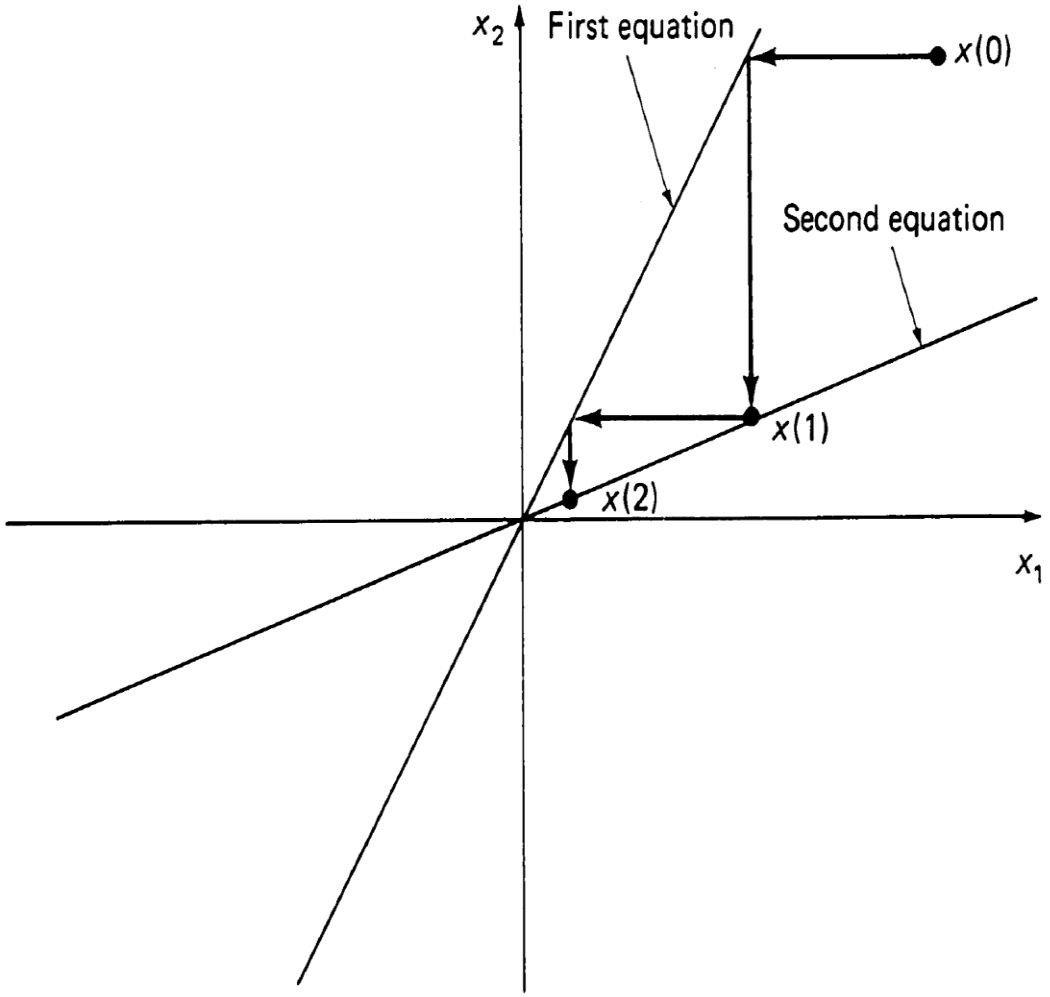
Pornind din $x(0) = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$, iteram:

$$x_1(1) = \frac{x_2(0)}{2} = 2$$

$$x_2(1) = \frac{x_1(1)}{2} = 1$$

$$x_1(2) = \frac{x_2(1)}{2} = \frac{1}{2}$$

$$x_2(2) = \frac{x_1(2)}{2} = \frac{1}{4}$$



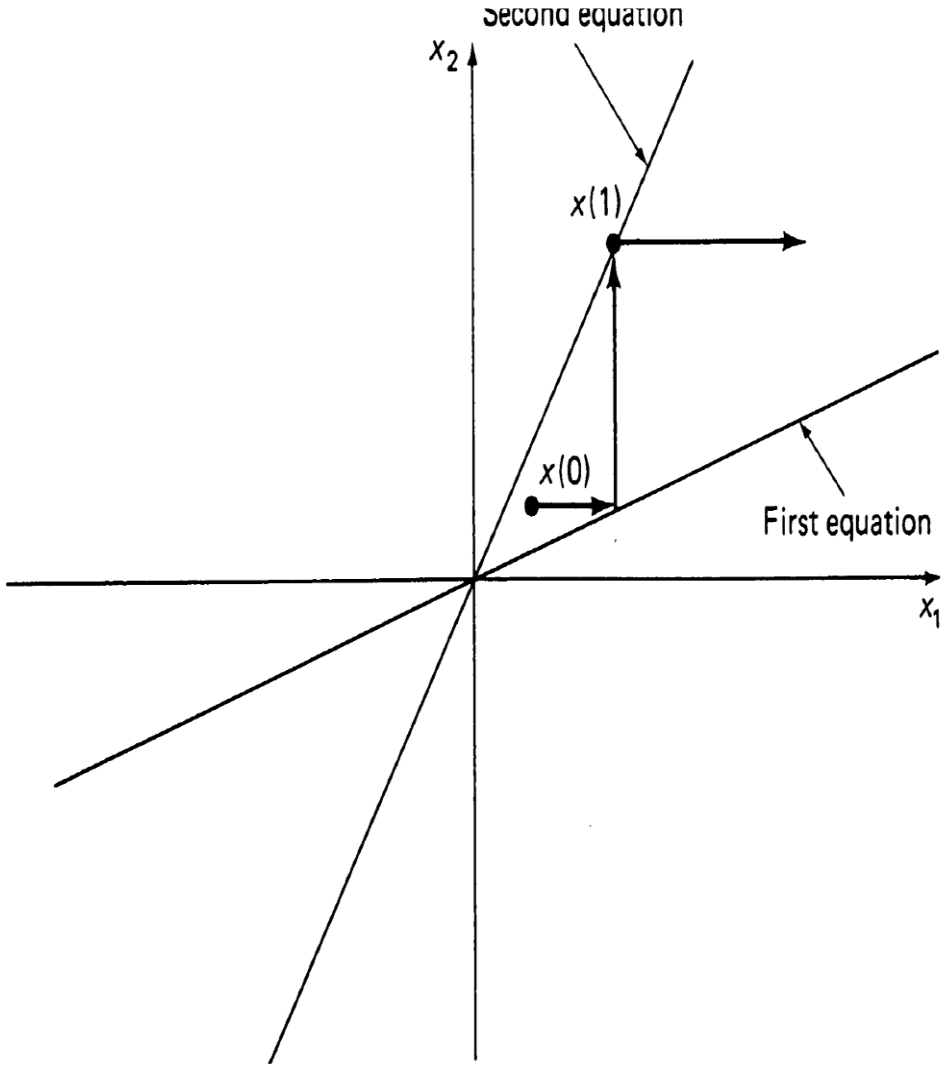
Algoritmul Gauss-Seidel pt iteratie divergenta

Fie sistemul $\begin{bmatrix} -1 & 2 \\ 2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Pornind din $x(0) = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$, iteram:

$$x_1(1) = 2x_2(0) = 8$$
$$x_2(1) = 2x_1(1) = 16$$

$$x_1(2) = 2x_2(1) = 32$$
$$x_2(2) = 2x_1(2) = 64$$



Convergenta algoritmului Gauss-Seidel

Teorema: Daca *A* este diagonal dominanta, atunci sirul generat de algoritmul Gauss-Seidel

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

converge.

Paralelizarea algoritmului Gauss-Seidel

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

- A matrice complet densa $(i.e., A_{ij} \neq 0 \; \forall \; i,j)$
  - cand procesorul i calculeaza $x_i(t+1)$ are nevoie de $x_j(t+1)$ pt $\forall j < i$
  - algoritm inerent secvential, nu se pot calcula in paralel componente x_i diferite
- A matrice rara
  - se pot folosi scheme de colorare a grafurilor
  - reminder: exista o ordine de evaluare a variabilelor in k pasi paraleli $\Leftrightarrow$ exista o colorare optimala cu k culori a grafului de dependenta

Algoritmul Gauss-Seidel paralel

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

La pasul t:

- P_1 calculeaza $x_1(t)$, produs scalar $O(n)$
- difuzare: P_1 difuzeaza pe $x_1(t)$ catre $P_2, \cdots, P_n$
- ...
- P_i primeste $x_{i-1}(t)$
- P_i calculeaza $x_i(t)$, produs scalar $O(n)$
- difuzare: P_i difuzeaza pe $x_i(t)$ catre $P_{i+1}, \cdots, P_n$

Terminarea algoritmilor iterativi

- conditie tipica de terminare

$$\|Ax(t) - b\| < \varepsilon$$

unde ε e o valoare suficient de mica

- pt matrici dense overhead-ul utilizarii normei nu e semnificativ
  - ex: norma infinit
  - la fiecare iteratie t, fiecare procesor evalueaza

$$max_i|[Ax]_i - b_i|$$

peste indicii i ai componentelor asignate acelui procesor

  - aceste valori se compara folosind un arbore de acoperire (spanning tree)

    => fiecare procesor propaga catre root maximul dintre valorile primite si cea calculate local

    => detectarea terminarii are nevoie doar de acumulare intr-un nod (penalizare mica fata de broadcastul/acumularea multinod executate in fiecare iteratie)

Terminarea algoritmilor iterativi pt matrici sparse

- A matrice sparse si variabile asignate procesoarelor a.i. fiecare iteratie nu necesita decat comunicare cu cel mai apropiat vecin in graful de dependente
- timp de comunicare per iteratie proportional cu nr de variabile asignate fiecarui procesor
- timpul de comunicare necesar testarii terminarii algoritmului proportional cu diametrul retelei de interconectare a procesoarelor (topologiei)

$\Rightarrow$ daca diametrul nu e comparabil (sau mai mic) decat nr variabilelor per procesor, testarea terminarii se face din cand in cand

Alternativ, testarea terminarii se poate face in paralel cu executia iteratiilor
=> $Ax - b$ va fi invechita la momentul luarii deciziei de terminare

- in practica, rareori acest lucru are consecinte adverse