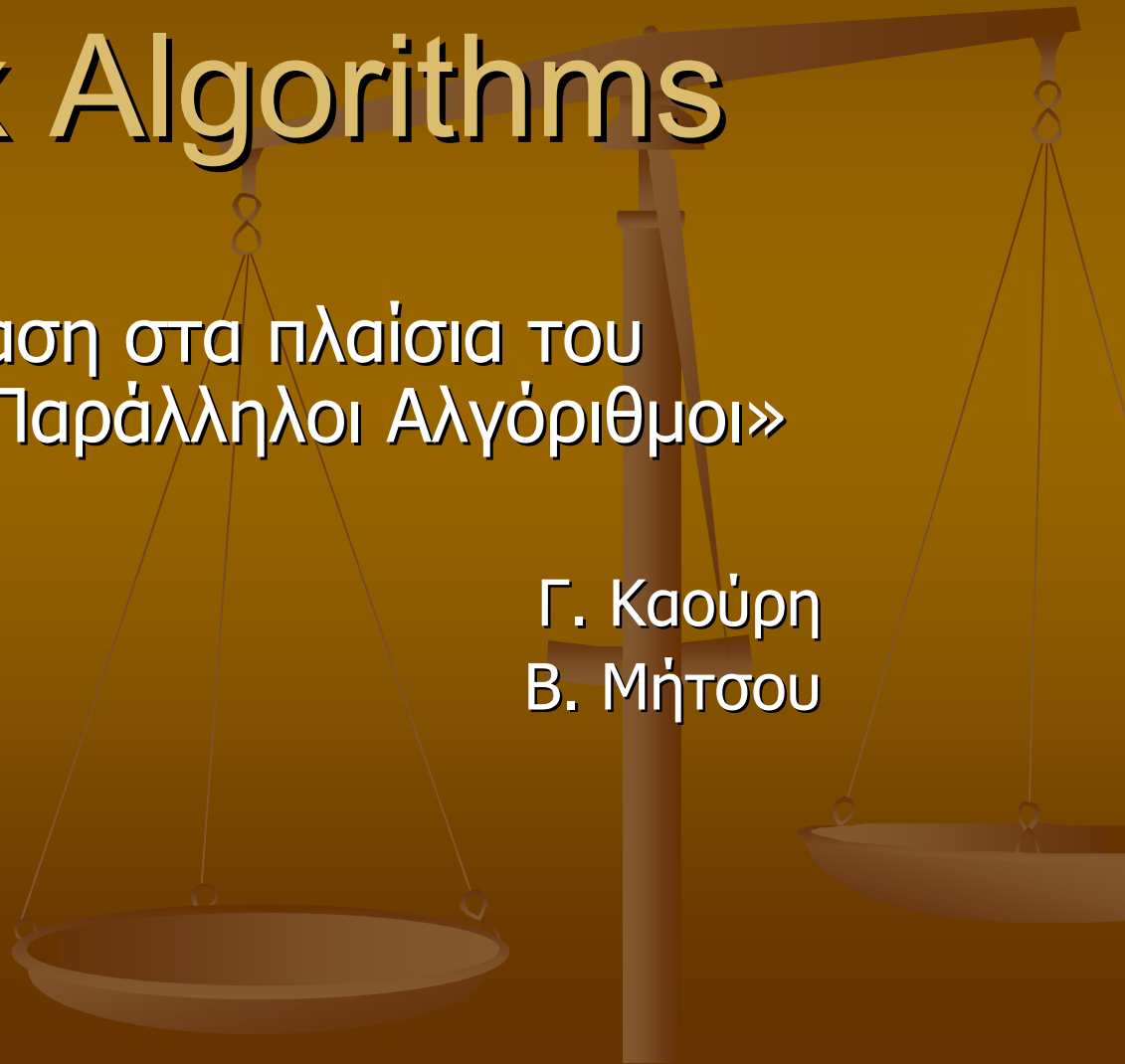
Matrix Algorithms

Παρουσίαση στα πλαίσια του μαθήματος «Παράλληλοι Αλγόριθμοι»

Γ. Καούρη
Β. Μήτσου



Περιεχόμενα παρουσίασης

- Πολλαπλασιασμός πίνακα με διάνυσμα
- Πολλαπλασιασμός πινάκων
- Επίλυση τριγωνικού συστήματος γραμμικών εξισώσεων (αντιστροφή πίνακα)
- Επίλυση τριδιαγώνιου συστήματος εξισώσεων
- Επίλυση τυχαίου συστήματος εξισώσεων με άμεσες μεθόδους
- Επίλυση τυχαίου συστήματος γραμμικών και διαφορικών εξισώσεων με επαναληπτικές μεθόδους

Πολλαπλασιασμός πίνακα με διάνυσμα (1/2)

Πρόβλημα: **A** = (a_{ij}) NxN πίνακας

x = (x_j) διάνυσμα N διαστάσεων

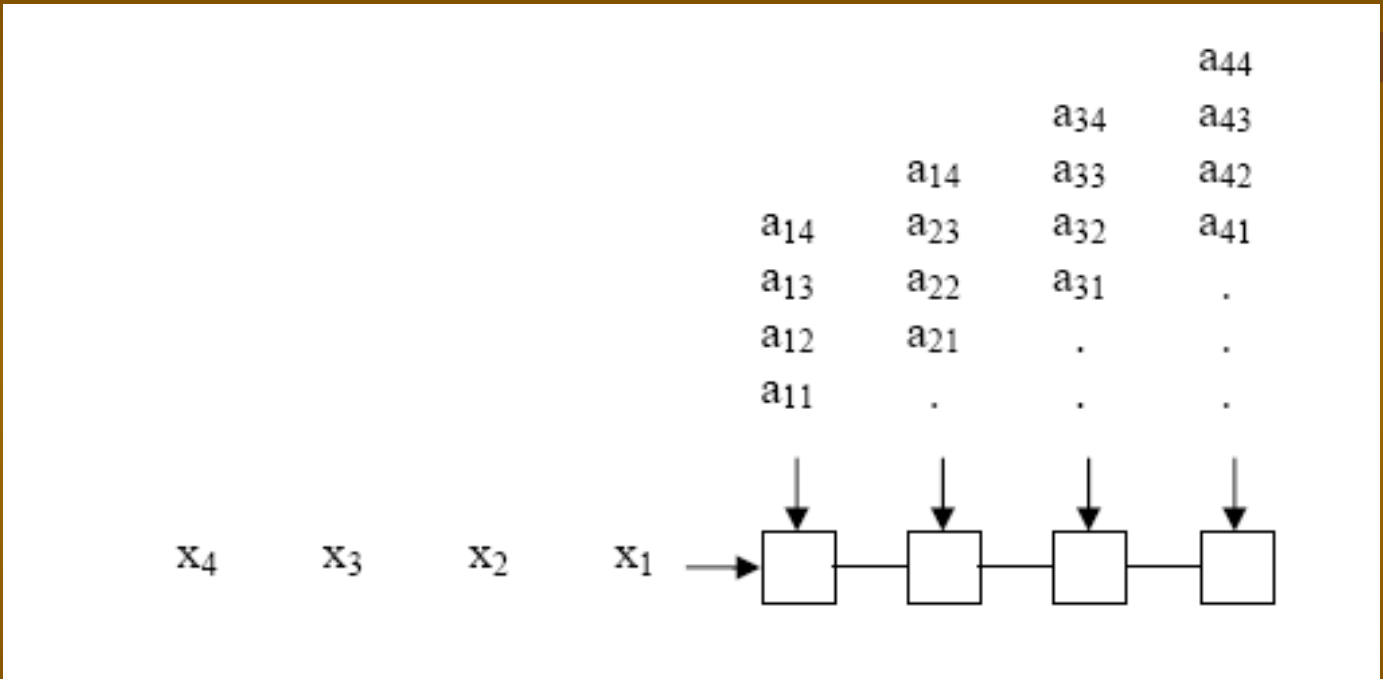
Να υπολογιστεί το γινόμενο **y** = **Ax**,όπου

y = (y_i) και $y_i = \sum_{j=1}^{N} a_{ij} x_j$, $1 \le i \le N$

sequential algorithm: $2N^2 - N$ steps

parallel algorithm: $2N - 1$ steps

Πολλαπλασιασμός πίνακα με διάνυσμα (2/2)



Υπολογισμός γινομένου πίνακα με διάνυσμα για N = 4 με τη χρήση N – διάστατου γραμμικού πίνακα

Πολλαπλασιασμός πινάκων (1/2)

Πρόβλημα: **A** = (a_{ij}) NxN πίνακας

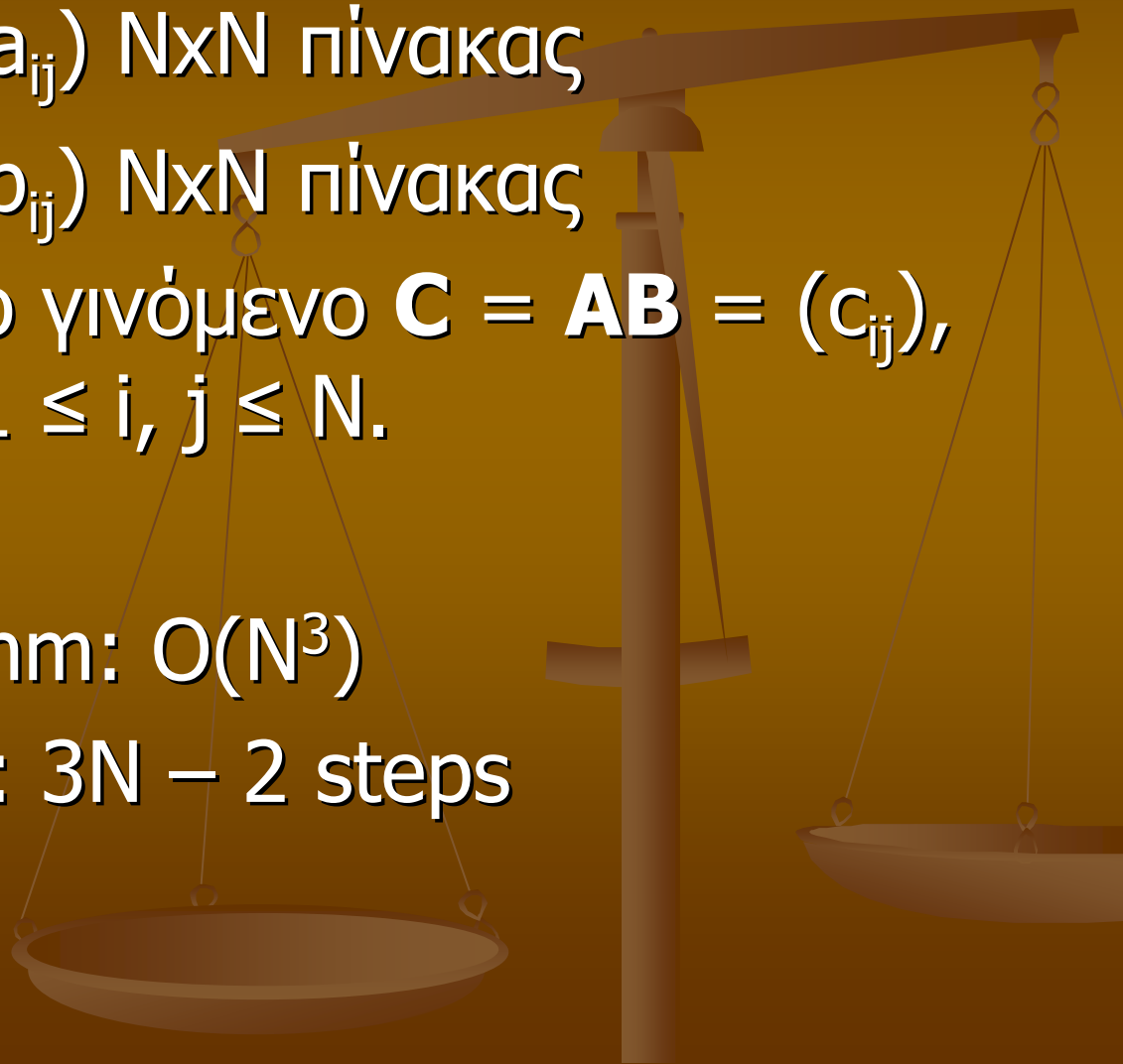
B = (b_{ij}) NxN πίνακας

Να υπολογιστεί το γινόμενο **C** = **AB** = (c_{ij}),

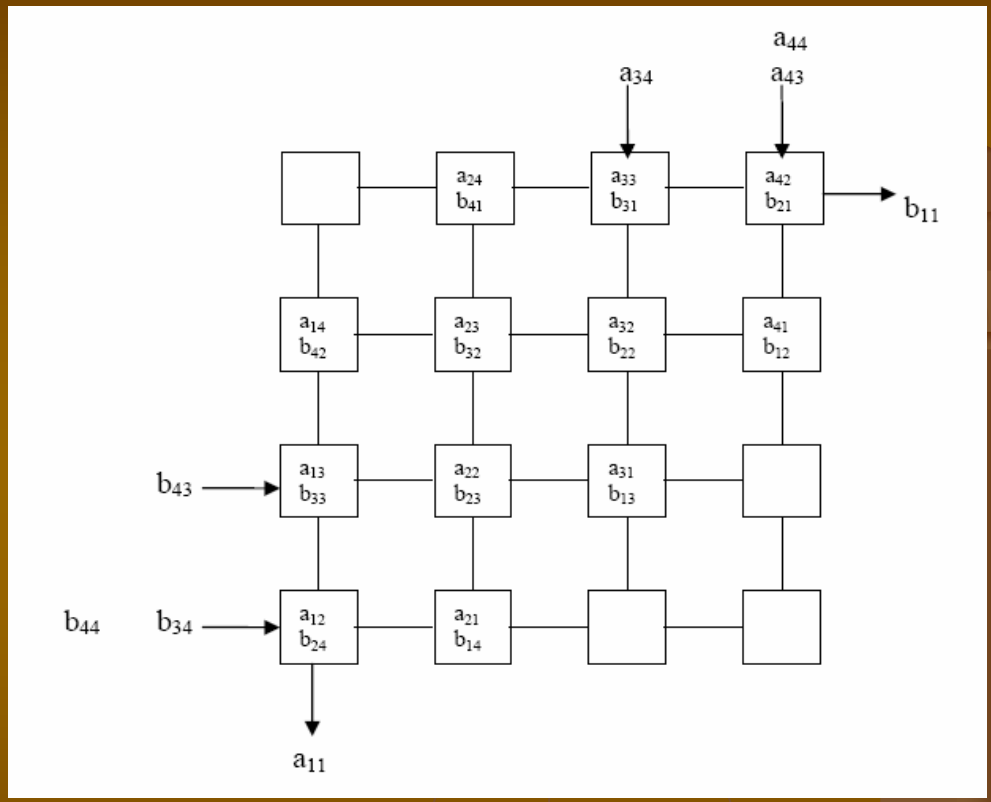
με $c_{ij} = \sum_{k=1}^{N} a_{ik} b_{kj}$, 1 ≤ i, j ≤ N.

sequential algorithm: $O(N^3)$

parallel algorithm: 3N – 2 steps

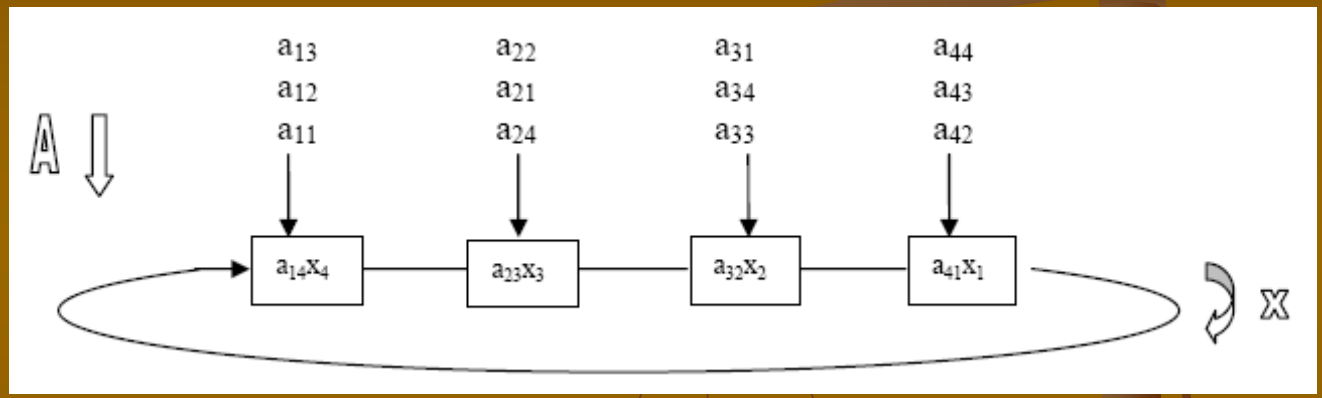


Πολλαπλασιασμός πινάκων (2/2)

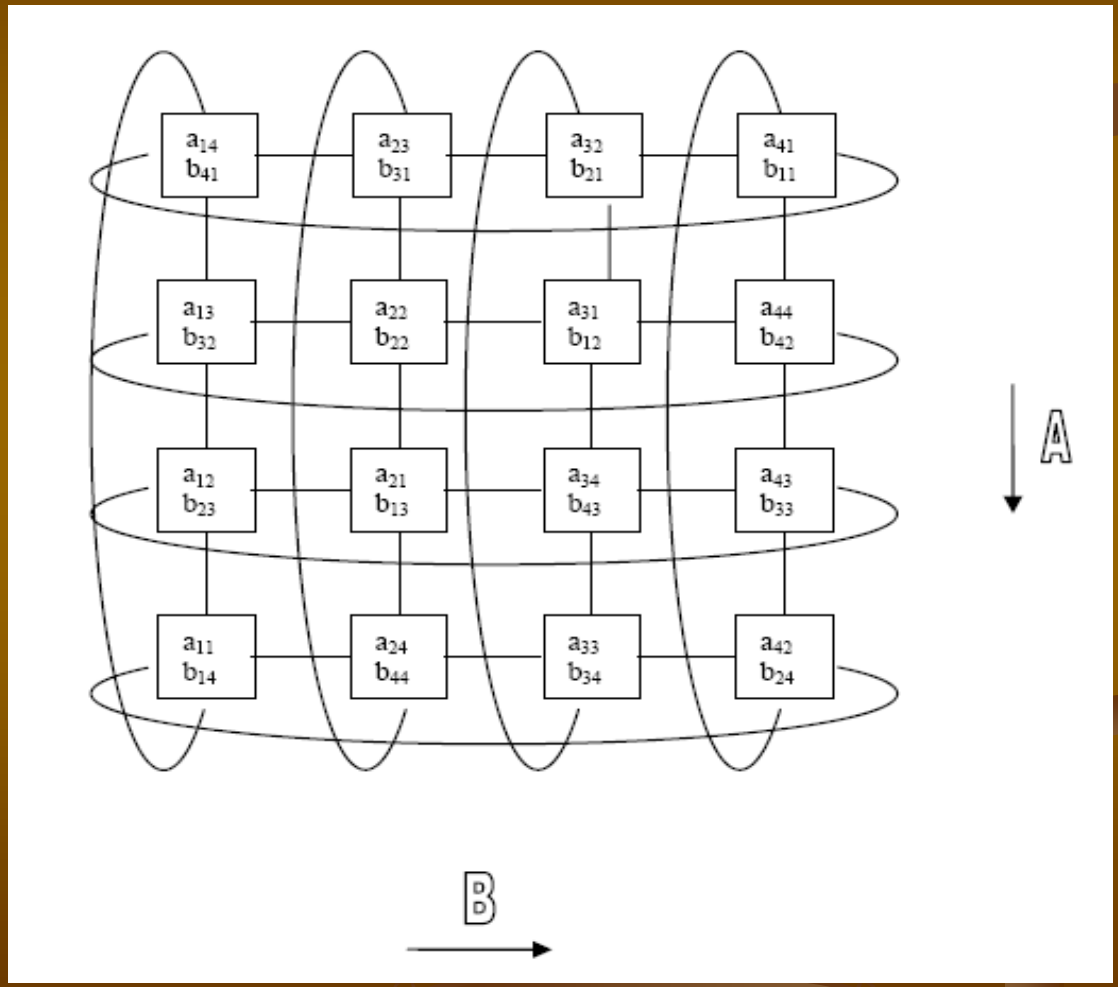


Υπολογισμός γινομένου πινάκων AxB, όπου A, B 4x4 πίνακες. Βρισκόμαστε στο 5ο βήμα του αλγόριθμου, όπου το κελί (i, j) υπολογίζει το $a_{ik}b_{kj}$, με $k = 7 - i - j$ και $1 \leq k \leq 4$.

Βελτίωση της απόδοσης των παραπάνω αλγορίθμων κατά σταθερό παράγοντα



Βελτίωση της απόδοσης των παραπάνω αλγορίθμων κατά σταθερό παράγοντα



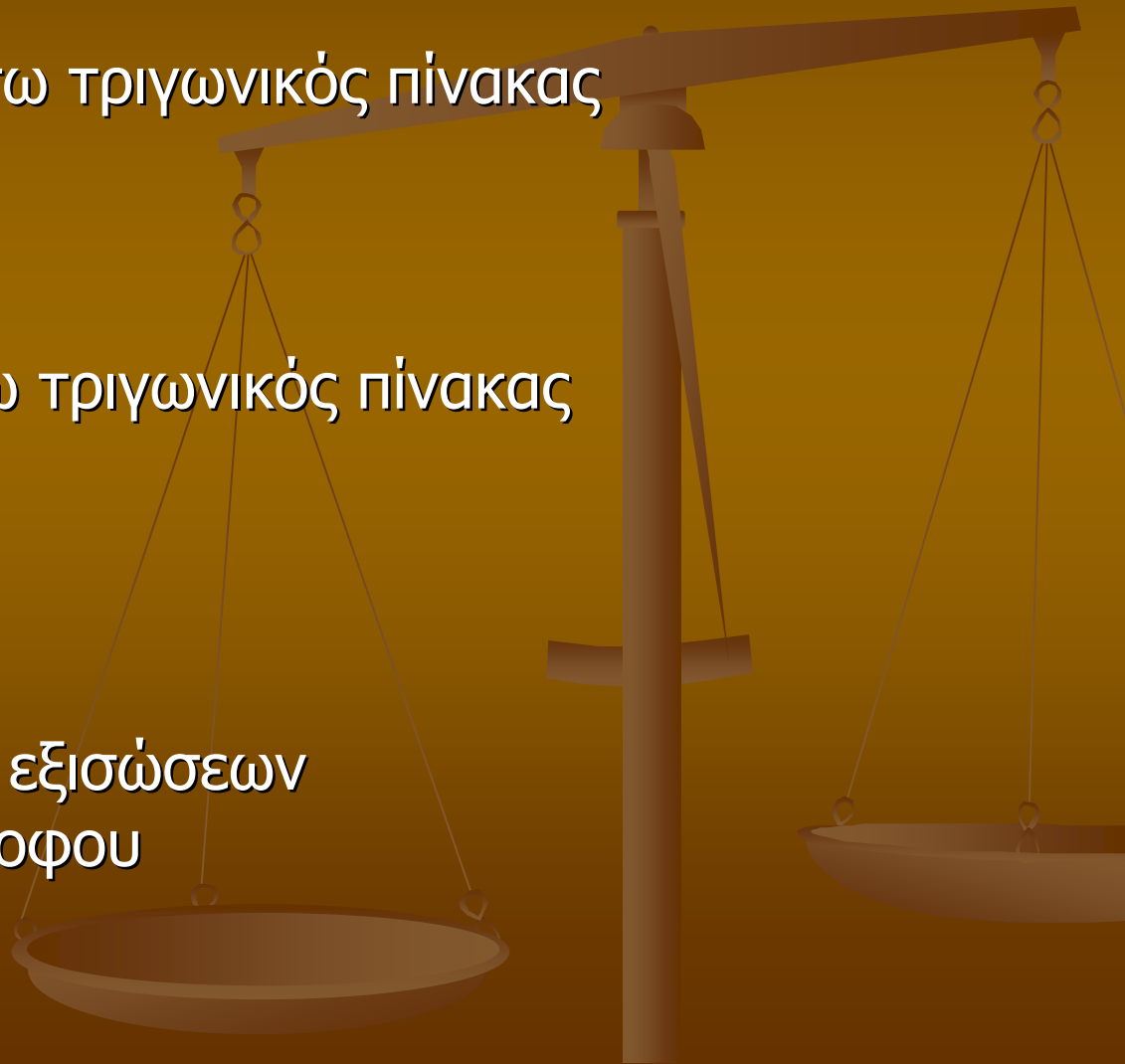
Τριγωνικοί Πίνακες

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

κάτω τριγωνικός πίνακας

$$B = \begin{pmatrix} 2 & 0 & 4 \\ 0 & 1 & 2 \\ 0 & 0 & 5 \end{pmatrix}$$

άνω τριγωνικός πίνακας



- Ορίζουσα
- Επίλυση συστήματος εξισώσεων
- Υπολογισμός αντίστροφου

Επίλυση τριγωνικού συστήματος εξισώσεων (1/3)

Έστω **A** = (a_{ij}) NxN κάτω τριγωνικός πίνακας και N – διάστατο διάνυσμα **b** = (b_i), θέλουμε να βρούμε το **x** = (x_j), όταν **Ax** = **b**.

Πρέπει $a_{ii} \neq 0$, για $1 \leq i \leq N$.

sequential algorithm: Πίσω αντικατάσταση

Επίλυση τριγωνικού συστήματος εξισώσεων (2/3)

parallel algorithm: 2N – 1 steps

Ορίζω σύνολο ενδιάμεσων τιμών $\{t_i\}$ ως εξής:

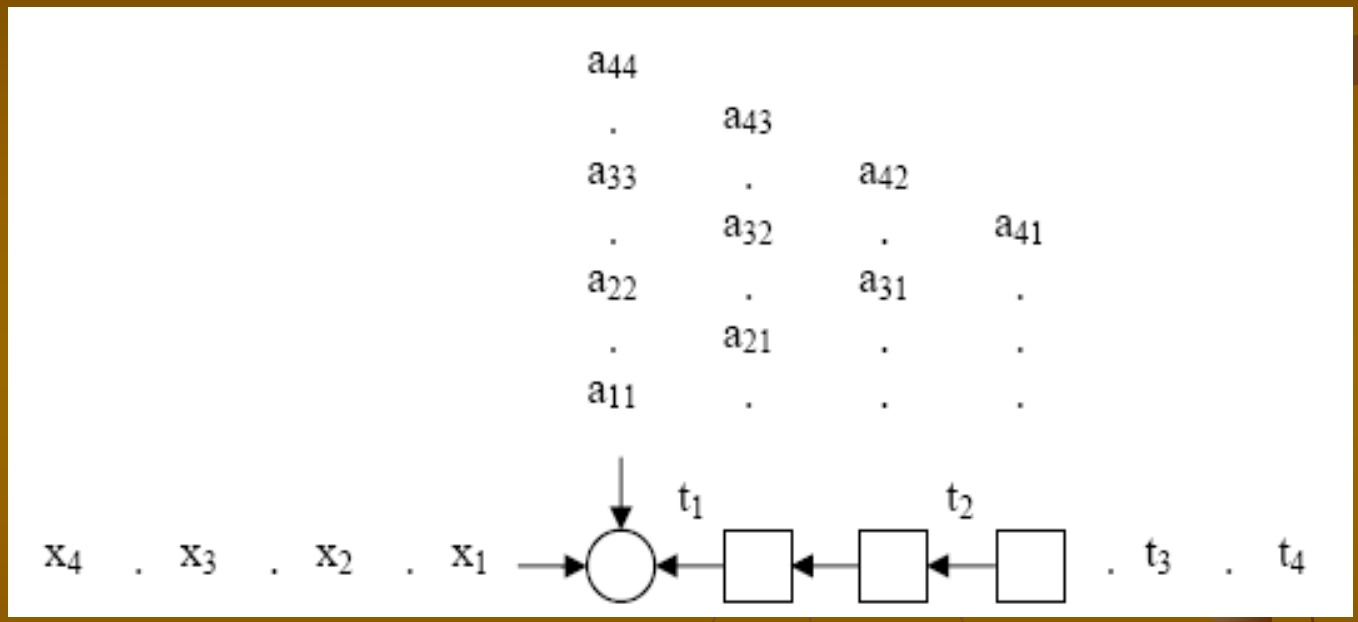
$$t_1 = b_1$$

$$t_i = b_i - \sum_{j=1}^{i-1} a_{ij} x_j$$

Όμως $b_i = \sum_{j=1}^{i} a_{ij} x_j$ οπότε $t_i = a_{ii} x_i$ και συνεπώς:

$$x_i = \frac{t_i}{a_{ii}}$$

Επίλυση τριγωνικού συστήματος εξισώσεων (3/3)



Αρχική τοποθέτηση δεδομένων για την επίλυση 4x4 κάτω τριγωνικού συστήματος εξισώσεων.

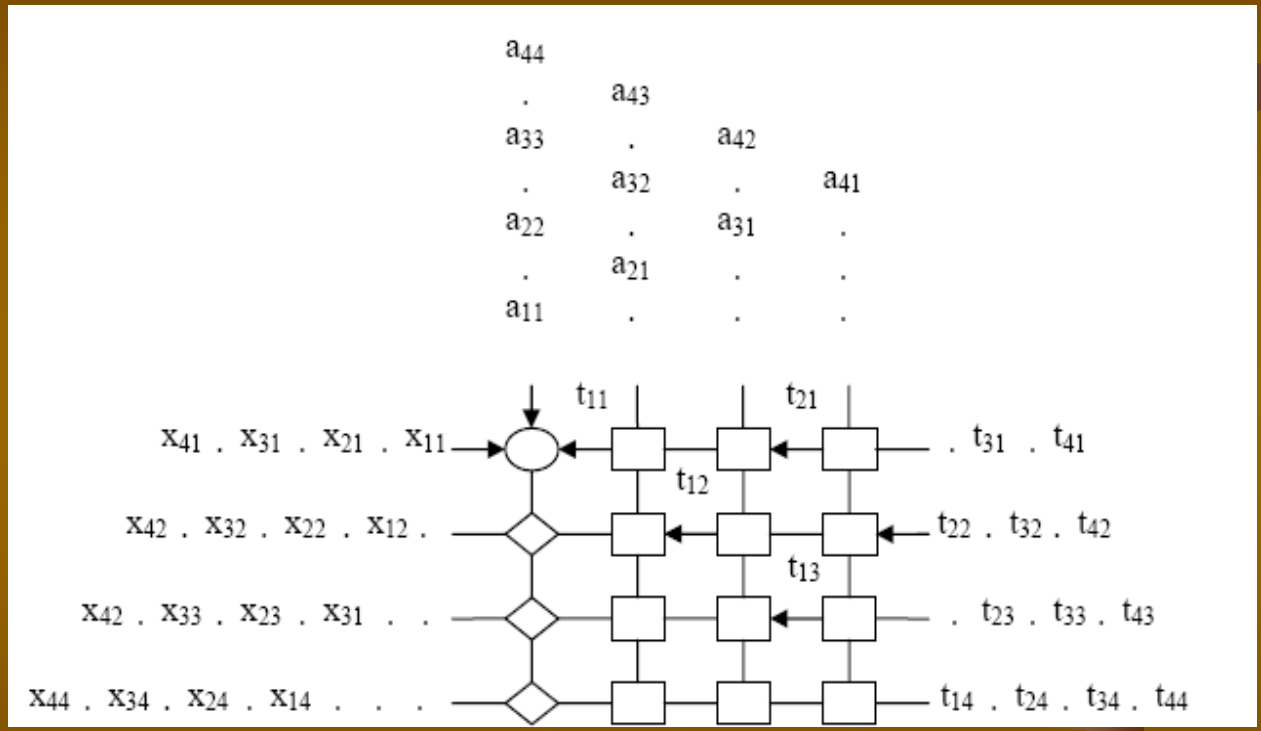
Παρατήρηση: Οι τιμές των t_i αρχικά ορίζονται ίσες με b_i.

Αντιστροφή Τριγωνικών Πινάκων (1/2)

Για την αντιστροφή πίνακα επιλύουμε το σύστημα των εξισώσεων $\mathbf{AX} = \mathbf{I}$.

Ειδικότερα θεωρώντας τα N συστήματα εξισώσεων $\mathbf{Ax}_j = \mathbf{e}_j$, όπου $\mathbf{e}_j = (0, \ldots, 0, 1, 0, \ldots, 0)^T$ και $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ μπορούμε να τα επιλύσουμε ταυτόχρονα σε ένα NxN array σε 3N – 2 βήματα, οπότε να βρεθεί και η λύση του $\mathbf{AX} = \mathbf{I}$.

Αντιστροφή Τριγωνικού Πίνακα (2/2)



Αρχική τοποθέτηση των δεδομένων για την αντιστροφή ενός 4x4 κάτω τριγωνικού πίνακα Α.

t_{ij}=0 αν i ≠ j και 1 αν i = j

Προσοχή: Τα διαφορετικά σχήματα κάνουν διαφορετικές διεργασίες!

Τριδιαγώνιοι Πίνακες

A = (a_{ij}) τριδιαγώνιος αν $a_{ij} = 0$ για κάθε i, j τέτοια ώστε |i - j| > 1

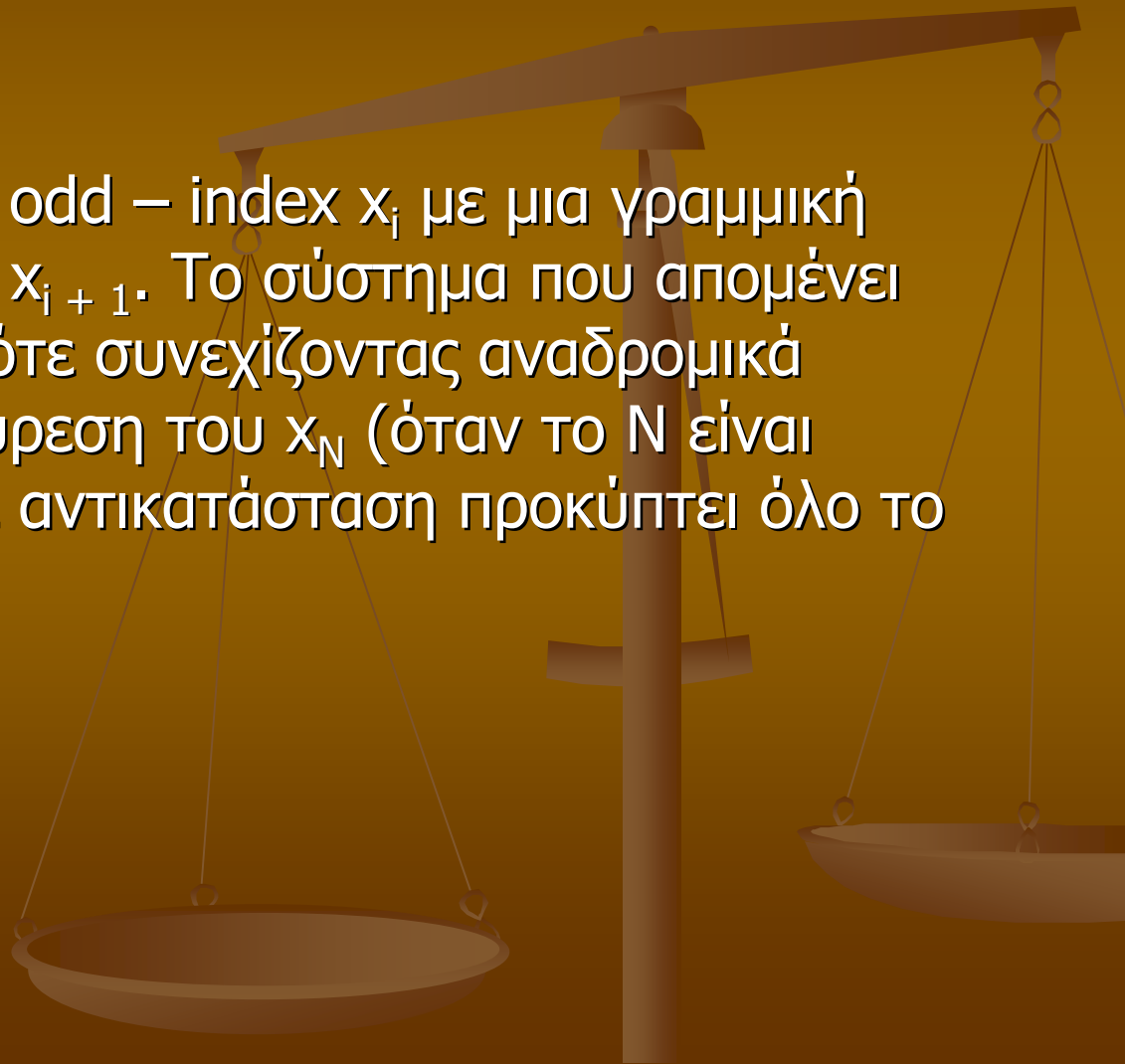
Παράδειγμα:

$$A = \begin{pmatrix} 7 & -2 & 0 & 0 & 0 \\ 5 & 4 & 1 & 0 & 0 \\ 0 & -3 & -3 & -1 & 0 \\ 0 & 0 & 2 & 6 & 4 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Επίλυση τριδιαγώνιου συστήματος εξισώσεων με τη μέθοδο odd – even reduction (1/4)

Γενική ιδέα:

Αντικαθιστούμε κάθε odd – index x_i με μια γραμμική συνάρτηση των x_{i-1}, x_{i+1}. Το σύστημα που απομένει είναι τριδιαγώνιο, οπότε συνεχίζοντας αναδρομικά καταλήγουμε στην εύρεση του x_N (όταν το N είναι δύναμη του 2) και με αντικατάσταση προκύπτει όλο το **x**.



Επίλυση τριδιαγώνιου συστήματος εξισώσεων με τη μέθοδο odd – even reduction (2/4)

Μαθηματική διατύπωση:

Έστω τριδιαγώνιο σύστημα εξισώσεων Ax = b, όπου

$$A=\begin{pmatrix} d_1 & u_1 & & \cdots & & \\ l_2 & d_2 & u_2 & \cdots & & 0 \\ & l_3 & d_3 & \cdots & & \\ & & \cdots & \cdots & \cdots & \\ 0 & & \cdots & l_{N-1} & d_{N-1} & u_{N-1} \\ & & \cdots & & l_N & d_N \end{pmatrix}$$

κάνουμε πρώτα την αντικατάσταση για κάθε odd – index x_i (θεωρούμε οτι $x_0 = 0$):

$$x_i = \frac{1}{d_i}\left(b_i - l_i x_{i-1} - u_i x_{i+1}\right)$$

Επίλυση τριδιαγώνιου συστήματος εξισώσεων με τη μέθοδο odd – even reduction (3/4)

Θεωρώντας οτι $d_i \neq 0$ για κάθε περιττό i, προκύπτει ένα νέο σύστημα εξισώσεων με μόνο even – index x_i. Ειδικότερα:

για $1 \leq i \leq N/2$:

$$l_{2i}^{(1)} x_{2i-1} + d_{2i}^{(1)} x_{2i} + u_{2i}^{(1)} x_{2i+2} = b_{2i}^{(1)}$$

όπου:

$$l_{2i}^{(1)} = -\frac{l_{2i} l_{2i-1}}{d_{2i-1}},$$

$$d_{2i}^{(1)} = -\frac{u_{2i-1} l_{2i}}{d_{2i-1}} + d_{2i} - \frac{u_{2i} l_{2i+1}}{d_{2i+1}},$$

$$u_{2i}^{(1)} = -\frac{u_{2i} u_{2i+1}}{d_{2i+1}},$$

$$b_{2i}^{(1)} = b_{2i} - \frac{l_{2i} b_{2i-1}}{d_{2i-1}} - \frac{u_{2i} b_{2i+1}}{d_{2i+1}},$$

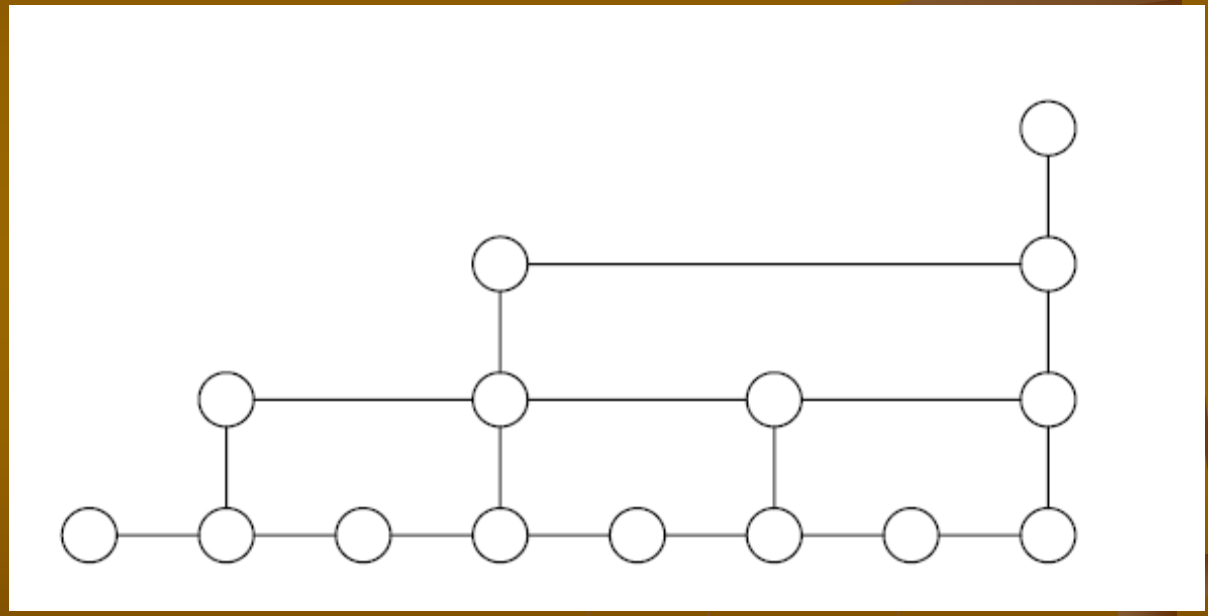
Επίλυση τριδιαγώνιου συστήματος εξισώσεων με τη μέθοδο odd – even reduction (4/4)

Μετά από logN επαναλήψεις απομένει 1 εξίσωση με έναν άγνωστο, οπότε λύνουμε ως προς αυτόν, στη συνέχεια υπολογίζουμε το $x_{N/2}$, μετά το $x_{N/4}$ και το $x_{3N/4}$ κ.ο.κ. μέχρι να υπολογιστούν όλα τα x_i.

Παρατηρήσεις

- Δε δουλεύει όταν κάποιο διαγώνιο στοιχείο είναι ή προκύψει 0.
- Είναι πολύ χρήσιμη για μεγάλες κλάσεις πινάκων, όπως οι συμμετρικοί θετικά ορισμένοι πίνακες και οι πίνακες με αυστηρή διαγώνια υπεροχή. Ιδιαίτερα για αυτές τις κλάσεις ο αλγόριθμος είναι αριθμητικά ευσταθής.

Multigrids



Parallel Prefix Algorithms (1/2)

α) Μετατρέπουμε την i-οστή εξίσωση του συστήματος σαν γινόμενο πίνακα – διανύσματος $l_i x_{i-1} + d_i x_i + u_i x_{i+1} = b_i$

$$\begin{pmatrix} x_{i+1} \\ x_i \\ 1 \end{pmatrix} = \begin{pmatrix} -\frac{d_i}{u_i} & -\frac{l_i}{u_i} & -\frac{b_i}{u_i} \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_i \\ x_{i-1} \\ 1 \end{pmatrix}$$

β) Με επαναλαμβανόμενες αντικαταστάσεις προκύπτει

$$\begin{pmatrix} x_{i+1} \\ x_i \\ 1 \end{pmatrix} = H_i \begin{pmatrix} x_1 \\ 0 \\ 1 \end{pmatrix} \qquad (1)$$

όπου $H_i = G_i G_{i-1} \cdots G_1$ και $G_i = \begin{pmatrix} -\frac{d_i}{u_i} & -\frac{l_i}{u_i} & -\frac{b_i}{u_i} \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Parallel Prefix Algorithms (2/2)

γ) Λύνουμε το 3Χ3 σύστημα εξισώσεων

$$\begin{pmatrix} x_N \\ x_{N-1} \\ 1 \end{pmatrix} = H_{N-1} \begin{pmatrix} x_1 \\ 0 \\ 1 \end{pmatrix},$$

$$l_N x_{N-1} + d_N x_N = b_N$$

δ)Διαδοχικά αντικαθιστούμε στην (1) για την εύρεση των υπολοίπων x_j.

Παρατηρήσεις

- Ο συνολικός χρόνος που απαιτείται είναι O(logN) βήματα σε ένα N-leaf πλήρες δυαδικό δέντρο.
- Ο αλγόριθμος δουλεύει καλά για κάθε τριδιαγώνιο πίνακα.
- Είναι αριθμητικά ασταθής όταν το

  $u_i << d_i, l_i$, οπότε προτιμάται η odd-even reduction.

LU- Παραγοντοποίηση(1/2)

$$A = \begin{pmatrix} d_1 & u_1 & & \cdots & & \\ l_2 & d_2 & u_2 & \cdots & & 0 \\ & l_3 & d_3 & u_3 & \cdots & \\ & & \cdots & \cdots & \cdots & \\ 0 & & & \cdots & l_{N-1} & d_{N-1} & u_{N-1} \\ & & & \cdots & & l_N & d_N \end{pmatrix} =$$

$$= \begin{pmatrix} 1 & & & & \\ p_2 & 1 & & 0 & \\ & p_3 & 1 & & \\ & & \cdots & & \\ 0 & & & p_{N-1} & 1 \\ & & & & p_N & 1 \end{pmatrix} \begin{pmatrix} q_1 & u_1 & & \cdots & \\ & q_2 & u_2 & \cdots & 0 \\ & & q_3 & u_3 & \cdots \\ & & \cdots & \cdots & \cdots \\ 0 & & & \cdots & q_{N-1} & u_{N-1} \\ & & & \cdots & & q_N \end{pmatrix}$$

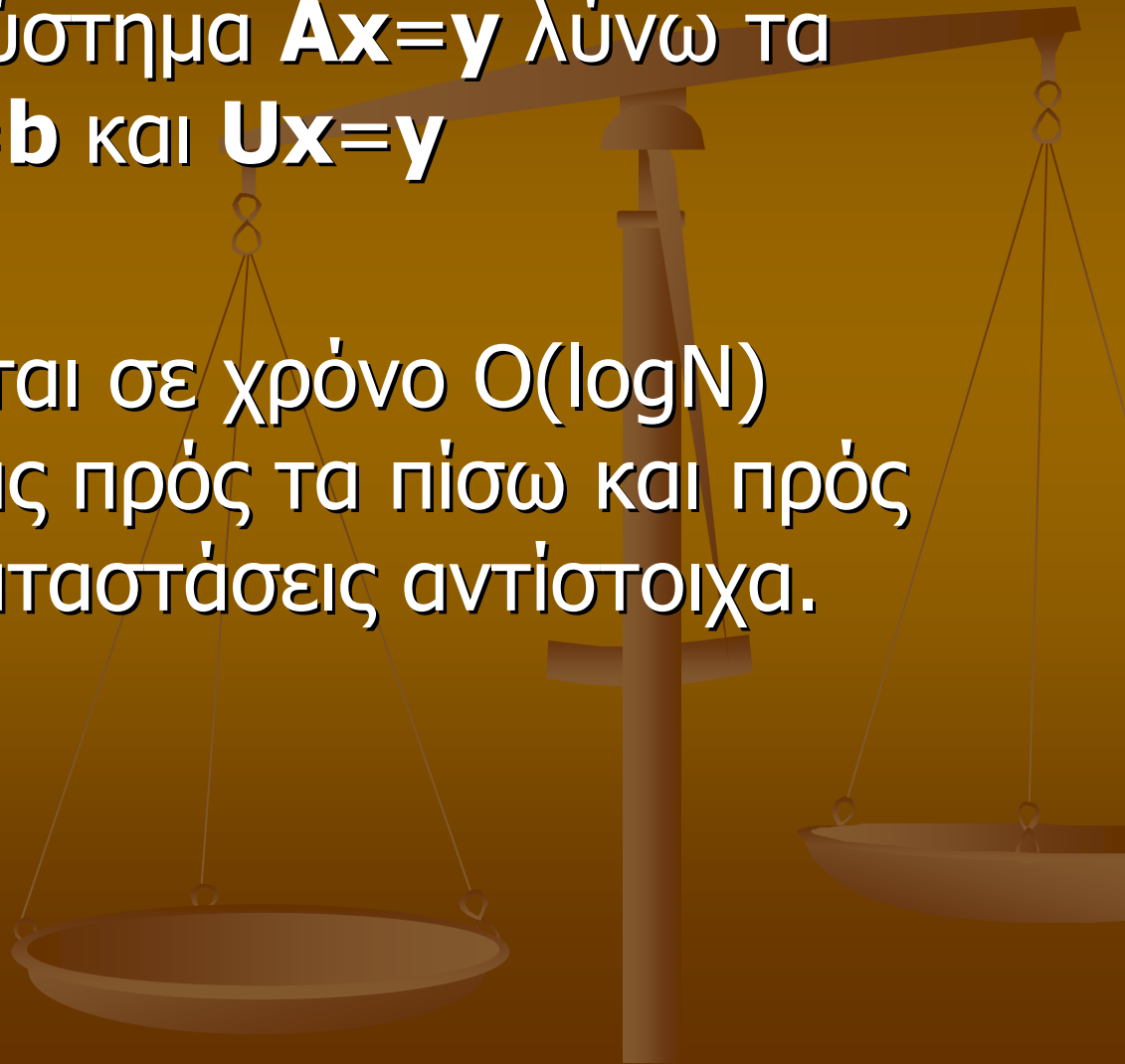
όπου τα q_i είναι μη-μηδενικά

LU-Παραγοντοποίηση(2/2)

Για να λύσω το σύστημα **Ax=y** λύνω τα συστήματα **Ly=b** και **Ux=y**

(όπου **A = LU**)

Και το δύο λύνονται σε χρόνο O(logN) χρησιμοποιώντας πρός τα πίσω και πρός τα μπρός αντικαταστάσεις αντίστοιχα.



Παρατηρήσεις

Η μέθοδος μπορεί να εφαρμοστεί μόνο σε ορισμένες κλάσεις πινάκων (σ' αυτές που μπορούσε να εφαρμοστεί και η odd-even reduction).

Ο υπολογισμός πινάκων **L**, **U** τέτοιων ώστε **A** = **LU** είναι μια διαδικασία που μπορεί επίσης να λυθεί με έναν parallel-prefix αλγόριθμο.

Gaussian Elimination(1/5)

Χρησιμοποιείται στην επίλυση γενικού γραμμικού συστήματος εξισώσεων

Ax = B.

Γενική ιδέα:

Προσπαθούμε να φέρουμε τον επαυξημένο πίνακα [**A|b**] στη μορφή [**I|b'**] εφαρμόζοντας γραμμοπράξεις.

Gaussian Elimination(2/5)

Μέθοδος:

- Για την πρώτη σειρά
  1. Βρίσκουμε την ψηλότερη γραμμή στην οποία το αριστερότερο στοιχείο είναι ≠0 και την τοποθετούμε πρώτη.
  2. Πολλαπλασιάζουμε την πρώτη γραμμή με τον αντίστροφο του στοιχείου $a_{1,1}$ έτσι ώστε το $a_{1,1}$ να γίνει ίσο με 1.
  3. Αφαιρούμε τα κατάλληλα πολλαπλάσια της πρώτης γραμμής από τις υπόλοιπες γραμμές ώστε σε κάθε γραμμή i, το στοιχείο $a_{i,1}$ να προκύψει 0.

Gaussian Elimination(3/5)

- Για τη δεύτερη σειρά

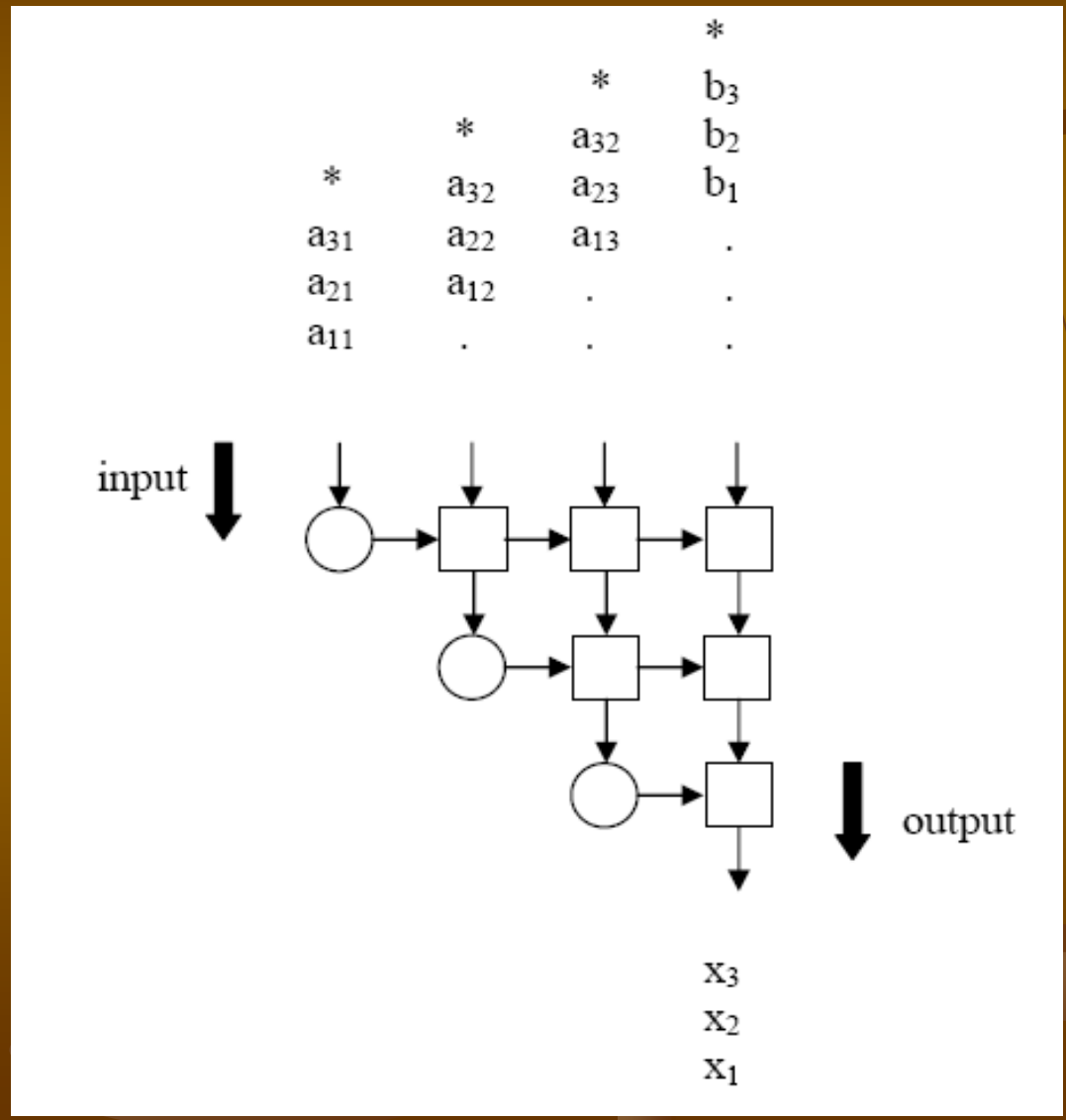
  Επαναλαμβάνουμε κατάλληλα τα βήματα 1 έως 3. Έπειτα αφαιρούμε το κατάλληλο πολλαπλάσιο της δεύτερης γραμμής από την πρώτη έτσι ώστε το στοιχείο $a_{1,2} = 0$.
- Για τις υπόλοιπες σειρές

  Επαναλαμβάνουμε τα τρία βήματα που περιγράφηκαν για τη δεύτερη σειρά μέχρι να σχηματιστεί ο μοναδιαίος πίνακας στις πρώτες Ν στήλες.

Τελικά η λύση του συστήματος είναι η τελευταία στήλη.

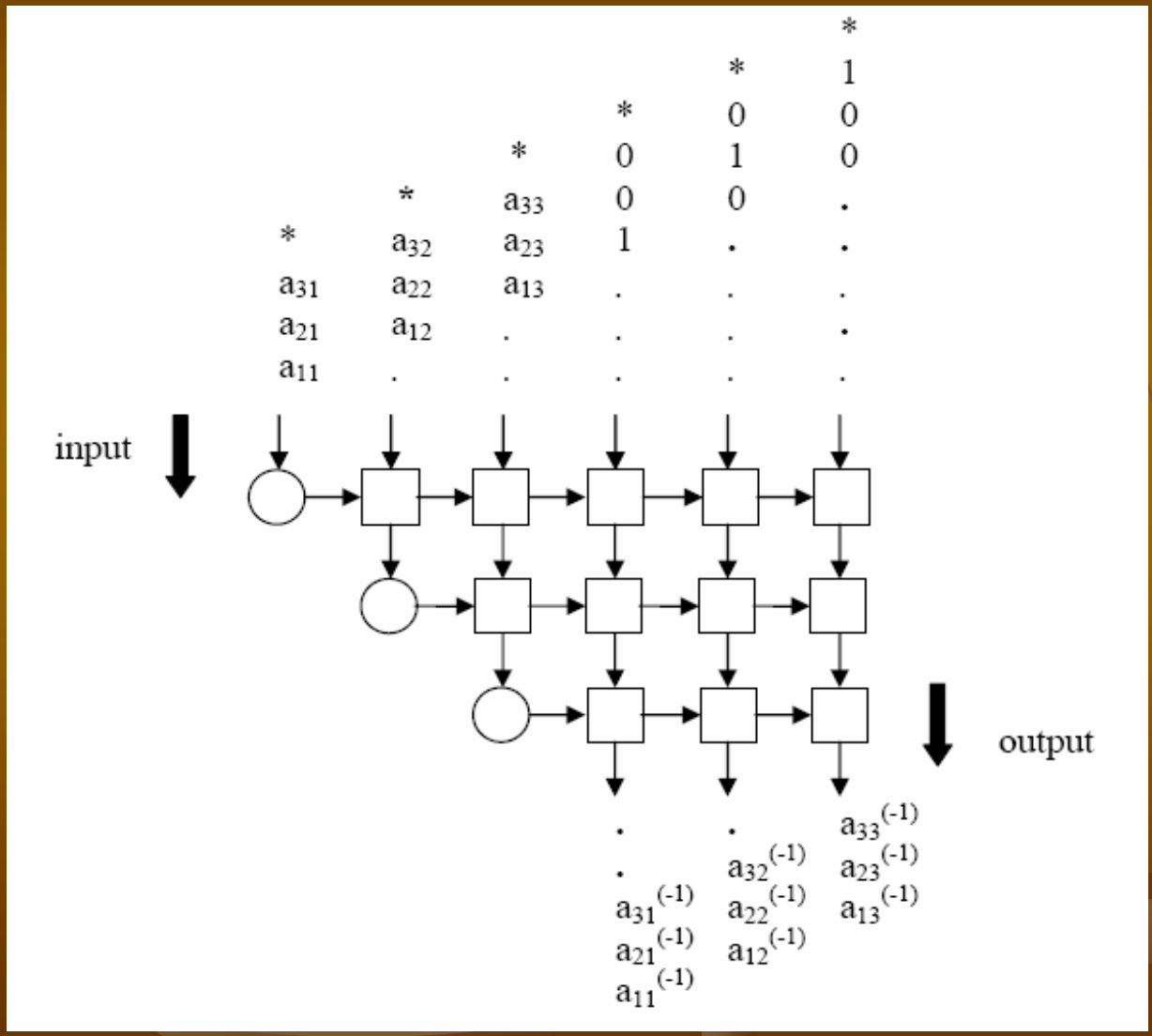
Gaussian Elimination(4/5)

Η διαδικασία που περιγράφεται παραπάνω μοντελοποιείται από mesh of arrays και ολοκληρώνεται σε 4N – 1 βήματα:



Gaussian Elimination(5/5)

Η ίδια διαδικασία μπορεί να μοντελοποιηθεί για να βρεθεί ο αντίστροφος ενός πίνακα **A**, αν στη θέση του διανύσματος **b** τοποθετήσουμε το μοναδιαίο πίνακα I. Η διαδικασία ολοκληρώνεται σε 5N – 2 βήματα.



Επαναληπτικές μέθοδοι

- Jacobi Relaxation
- Gauss – Seidel Relaxation

Όταν το σύστημα που έχουμε να λύσουμε συγκλίνει ($M^t \to \infty$, $M^t = D^{-1}(D - A)$), μπορούμε να χρησιμοποιήσουμε επαναληπτικές μεθόδους.

Jacobi Relaxation (1/2)

Ax = b σύστημα εξισώσεων

A αντιστρέψιμος (μοναδική λύση)

aii ≠ 0

Η i – οστή γραμμή γράφεται ως:

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

Οπότε ένας τρόπος να γραφτεί η λύση είναι:

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

Jacobi Relaxation (2/2)

Ο ζητούμενος υπολογισμός μπορεί να εκφραστεί σαν γινόμενο πίνακα – διάνυσμα ως εξής:

$$\begin{pmatrix} x_1(t+1) \\ x_2(t+1) \\ \dots \\ x_N(t+1) \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & -\frac{a_{12}}{a_{11}} & \dots & -\frac{a_{1N}}{a_{11}} & \frac{b_1}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & \dots & -\frac{a_{2N}}{a_{22}} & \frac{b_2}{a_{22}} \\ & & \dots & & \\ -\frac{a_{N1}}{a_{NN}} & -\frac{a_{N2}}{a_{NN}} & \dots & 0 & \frac{b_N}{a_{NN}} \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \\ \dots \\ x_N(t) \\ 1 \end{pmatrix}$$

Οπότε μπορεί να πραγματοποιηθεί από N – cell linear array σε O(N) βήματα.

Gauss – Seidel Relaxation (1/2)

Πλέον ο τύπος ανανέωσης του $x_i(t+1)$ είναι:

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

Η μέθοδος αυτή συγκλίνει πιο γρήγορα από την Jacobi, αλλά το βασικό της μειονέκτημα είναι οτι είναι πιο σειριακή.

Οπότε, κάθε επανάληψη της μεθόδου χρειάζεται ένα N – cell linear array.

Gauss – Seidel Relaxation (1/2)