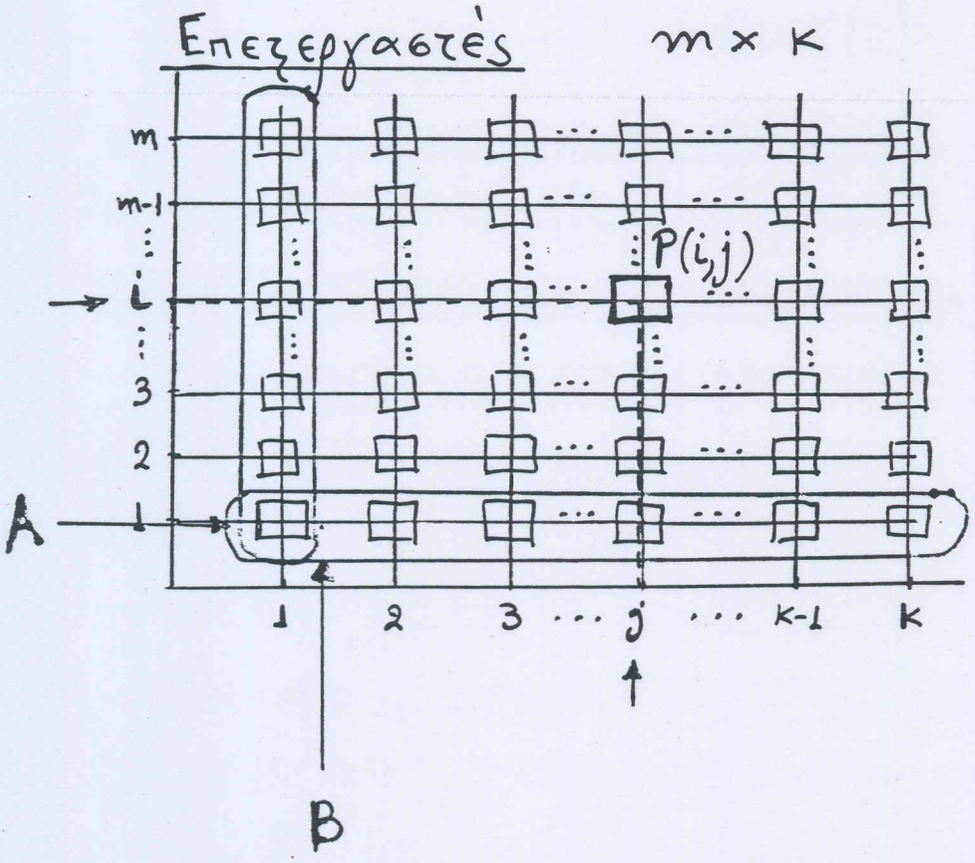
2.1. Πολλαπλασιασμός πινάκων σε ορθογώνιο πλέγμα επεξεργαστών

$m \times n$ $n \times k$ $m \times k$

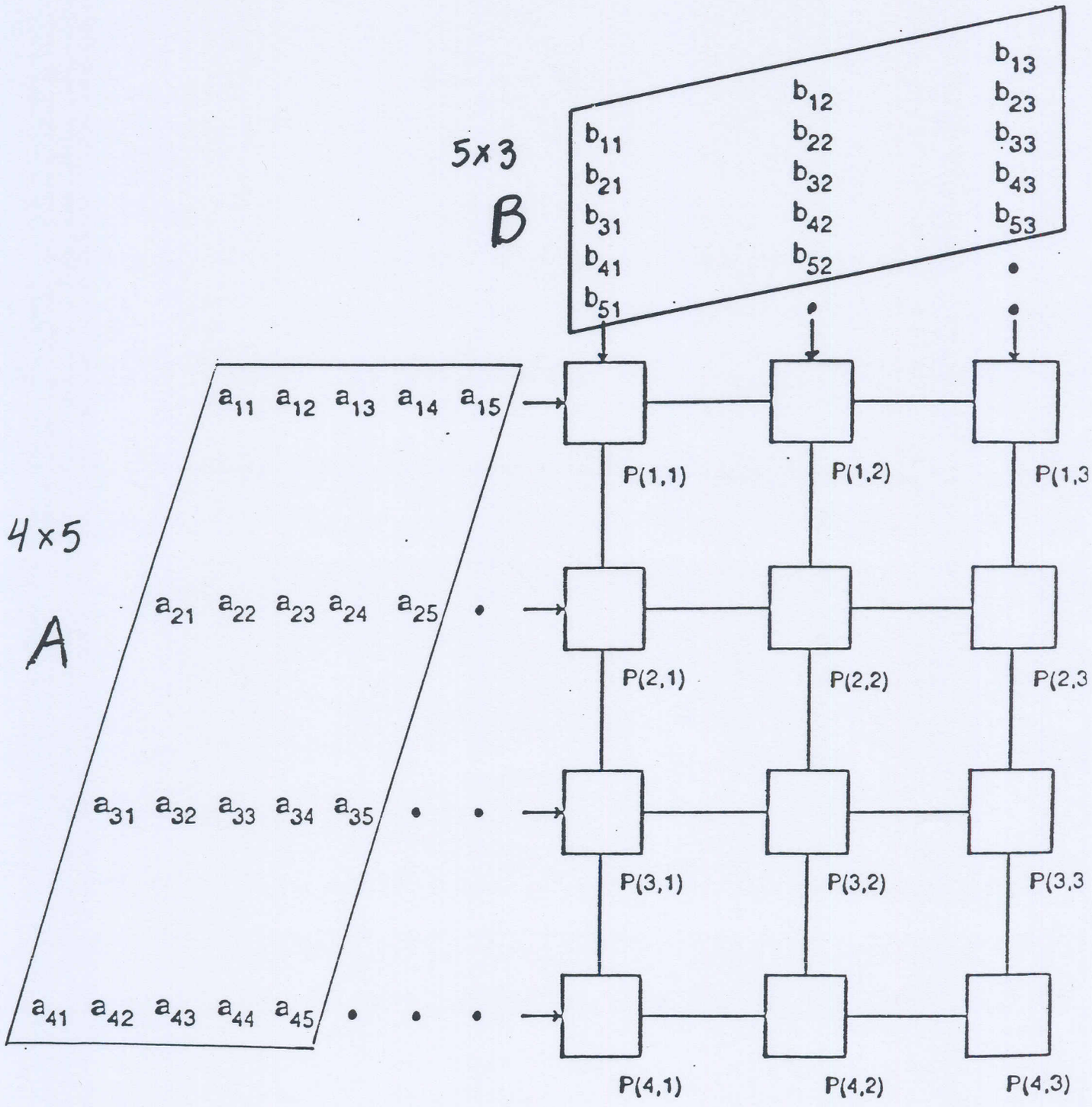
$A \quad B \longrightarrow C$



Ο πίνακας A τροφοδοτεί τους επεξεργαστές στην 1η στήλη.

Ο πίνακας B » » στην 1η γραμμή.

παράδειγμα $m=4$, $n=5$, $k=3$



- καθυστέρηση εισόδου κατά μία χρονική στιγμή

Έτσι εξασφαλίζεται η συνάντηση των a_{is} και b_{sj} την ίδια ακριβώς χρονική στιγμή στον επεξεργαστή P_{ij}.

- Η γραμμή i του πίνακα A καθυστερεί την είσοδό της κατά μια χρονική στιγμή μετά την γραμμή $i-1$, $2 \le i \le m$
- Η στήλη j του πίνακα B καθυστερεί την είσοδό της κατά μια

Αλγόριθμος

Αρχικά $c_{ij} \leftarrow 0$

Όταν ο επεξεργαστής P_{ij} δέχεται δύο στοιχεία a και b τότε:

1. Πολλαπλασιάζει αυτά $a \times b$
2. Προσθέτει το αποτέλεσμα στο c_{ij}

$$c_{ij} \leftarrow c_{ij} + (a \times b)$$

3. Στέλνει το a στον $P_{i,j+1}$ εκτός αν $j = k$

$$P_{i,j+1} \leftarrow a$$

4. Στέλνει το b στον $P_{i+1,j}$ εκτός αν $i = m$

$$P_{i+1,j} \leftarrow b$$

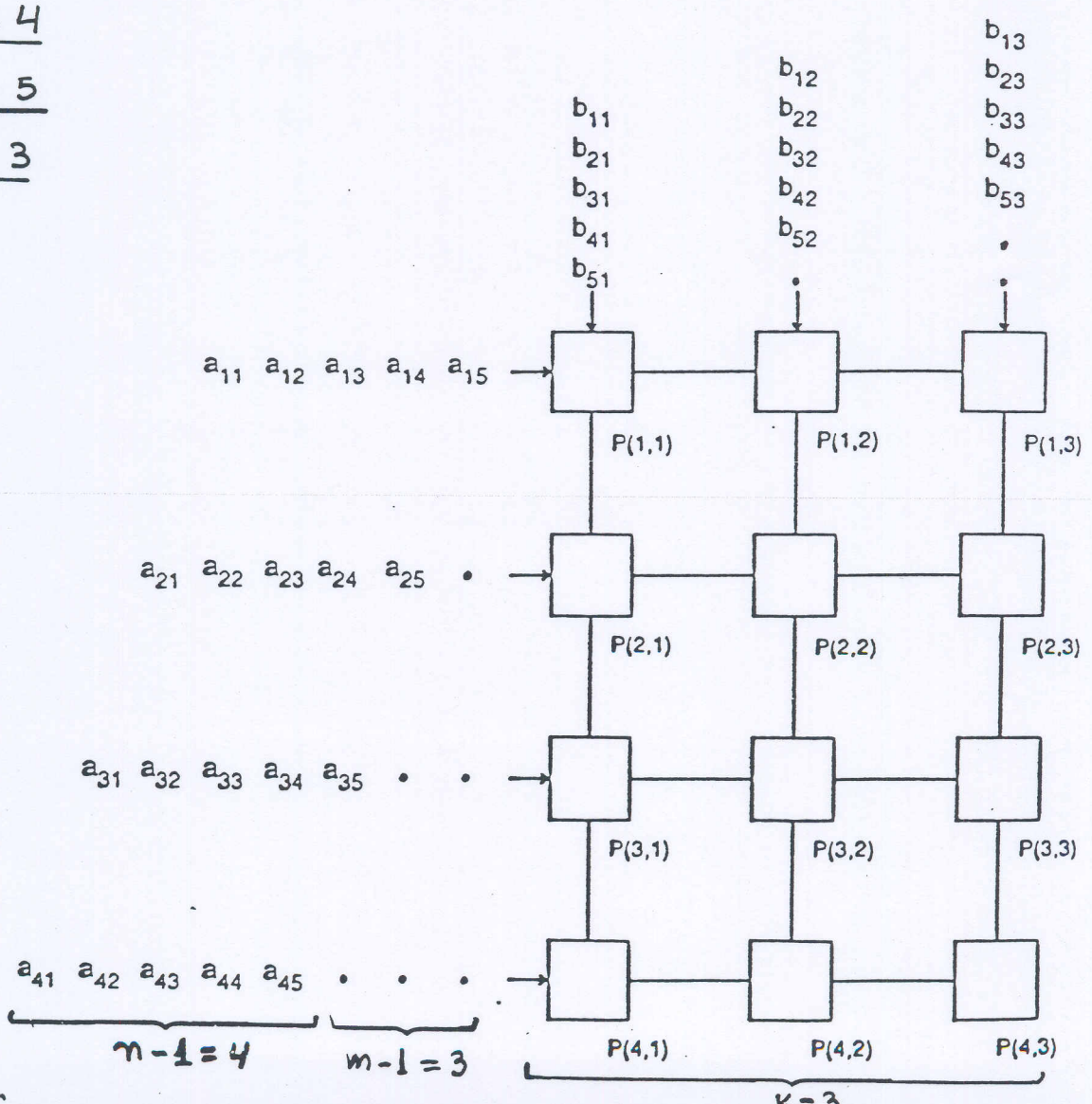
procedure MESH MATRIX MULTIPLICATION (A, B, C)

for $i = 1$ to m do in parallel
  for $j = 1$ to k do in parallel
    (1) $c_{ij} \leftarrow 0$
    (2) while $P(i, j)$ receives two inputs a and b do
      (i) $c_{ij} \leftarrow c_{ij} + (a \times b)$
      (ii) if $i < m$ then send b to $P(i+1, j)$
        end if
      (iii) if $j < k$ then send a to $P(i, j+1)$
        end if
    end while
  end for
end for.

$m = 4$

$n = 5$

$k = 3$



Η τελεία • σημαίνει καθυστέρηση εισόδου κατά 1 χρονική στιγμή

Ανάλυση

Τα στοιχεία a_{m1} και b_{1k} απαιτούν $\underline{m+k+n-2}$ βήματα από την αρχή του υπολογισμού μέχρι να φθάσουν στον επεξεργαστή P_{mk}.

Επειδή ο επεξεργαστής P_{mk} τελειώνει τελευταίος αυτά τα βήματα απαιτούνται για να υπολογισθεί το γινόμενο των πινάκων.

Αν $m \leq n$ και $k \leq n$ τότε:

Χρόνος εκτέλεσης $t(n) = O(n)$ (ο ταχύτερος δυνατός σε δίκτυο)

αριθμός επεξεργαστών $p(n) = n^2$

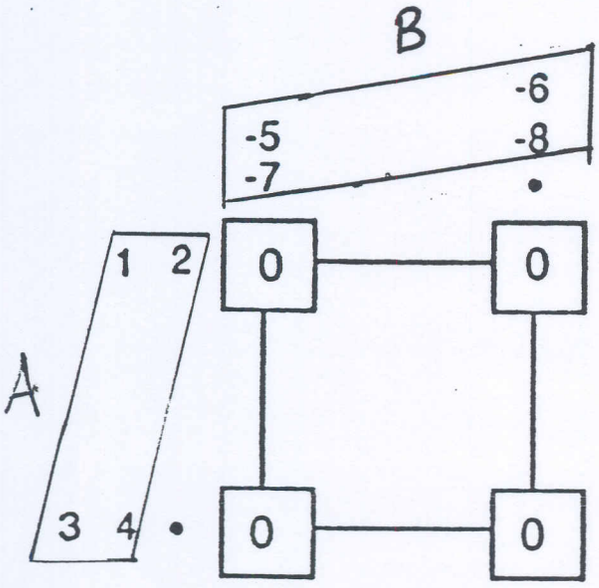
Υπολογιστικό κόστος $c(n) = O(n^3)$

Ο χρόνος εκτέλεσης του procedure MESH MATR MULT είναι ο ταχύτερος δυνατός σε ένα δίκτυο επεξεργαστών, με την υπόθεση ότι μόνο οι συνοριακοί επεξεργαστές διαχειρίζονται τις λειτουργίες εισόδου/εξόδου.

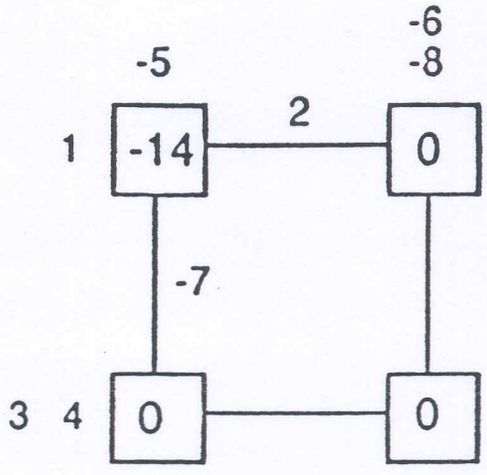
Πράγματι με την προϋπόθεση αυτή απαιτούνται $\Omega(n)$ βήματα για να διαβασθούν από την είσοδο (από τους επεξεργαστές στην 1η γραμμή και 1η στήλη) και/ή για να παραχθούν από την έξοδο (από τους επεξεργαστές στη γραμμή m και στήλη k).

Παράδειγμα $m = n = \kappa = 2$

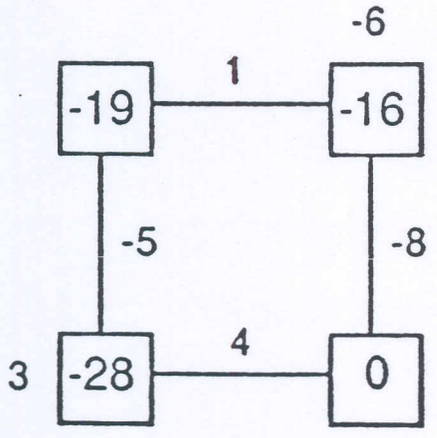
$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad B = \begin{bmatrix} -5 & -6 \\ -7 & -8 \end{bmatrix} \qquad C = A \cdot B = \begin{bmatrix} -19 & -22 \\ -43 & -50 \end{bmatrix}$$



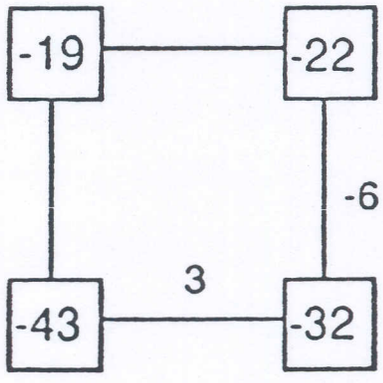
(a)



(b)

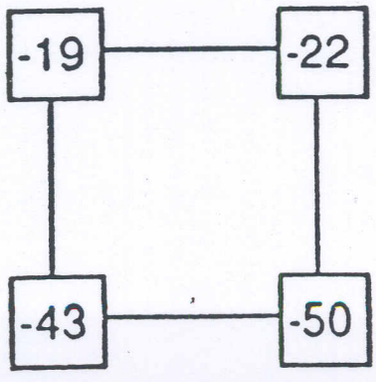


(c)



(d)

C =



(e)

2.2 Πολλαπλασιασμός πινάκων σε κύβο

Αναζητώντας να πετύχουμε ένα ταχύτερο αλγόριθμο στρεφόμαστε σε μια άλλη αρχιτεκτονική.
Το μοντέλο που επιλέγουμε είναι ένας SIMD υπολογιστής του οποίου οι επεξεργαστές συνδέονται με ένα δίκτυο μορφής κύβου.

Έστω $N = 2^q$ επεξεργαστές

$$P_0, P_1, \dots, P_{2^q-1}, \quad q \geq 1$$

και

i, $i^{(b)}$ ακέραιοι $\quad 0 \leq i, i^{(b)} \leq 2^q - 1$

με δυαδικές παραστάσεις που διαφέρουν μόνο σε μια θέση $0 \leq b < q$

Δηλαδή, αν η δυαδική παράσταση του i είναι

$$i_{q-1} \dots i_{b+1} \underline{i_b} \, i_{b-1} \dots i_1 i_0$$

τότε η δυαδική παράσταση του $i^{(b)}$ είναι

$$i_{q-1} \dots i_{b+1} \underline{i'_b} \, i_{b-1} \dots i_1 i_0$$

όπου i'_b το δυαδικό συμπλήρωμα του bit i_b

Η διασύνδεση μεταξύ των επεξεργαστών σε κύβο καθορίζεται ως εξής:

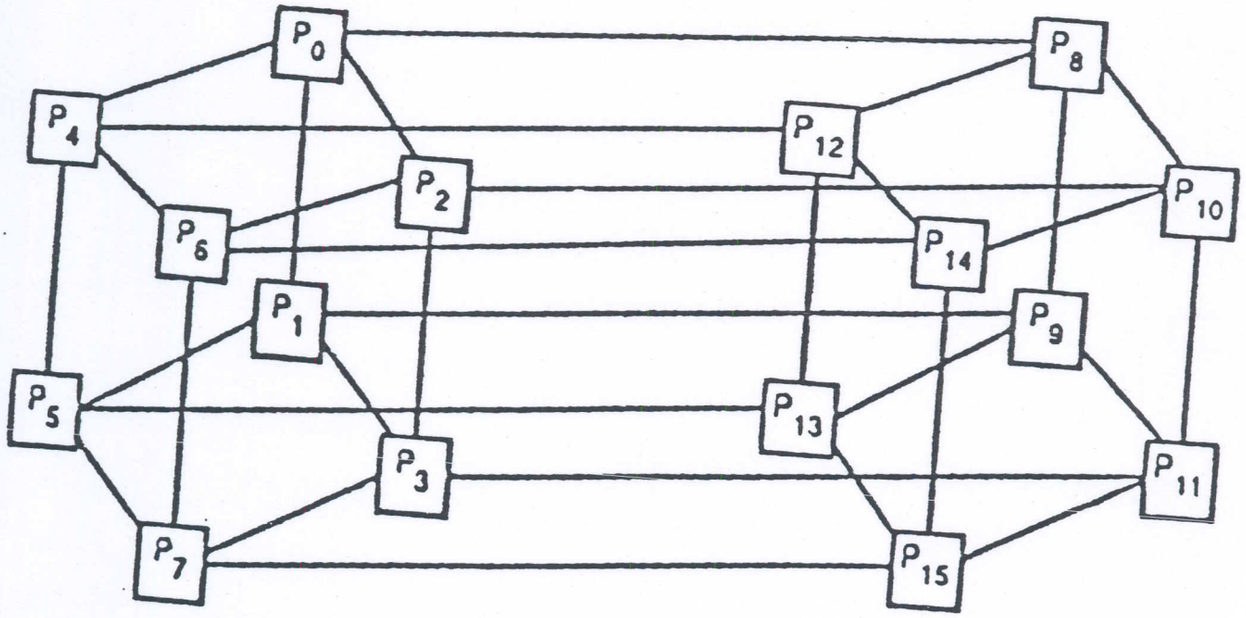
Ο P_i συνδέεται με τον $P_{i^{(b)}}$ με δύο συνδέσμους για κάθε $0 \leq b < q$.

Οι g επεξεργαστές που συνδέονται με τον P_i λέγονται γειτονικοί του P_i.

Παράδειγμα

$g = 4$

$N = 2^g = 16$ επεξεργαστές



Διασύνδεση τύπου κύβου με 16 επεξεργαστές

Εστω $n = 2^q$

$N = n^3 = 2^{3q}$ επεξεργαστές

Υποθέτουμε για λόγους απλής παρουσίασης ότι

A, B $n \times n$

Απεικονίζουμε τους επεξεργαστές σε ένα τριδιάστατο $n \times n \times n$ πίνακα έτσι ώστε:

ο επεξεργαστής P_r κατέχει την θέση (i, j, k)

όπου $r = in^2 + jn + k$

και $0 \le i, j, k \le n-1$ (row-major order)

Έτσι λοιπόν, αν η δυαδική παράσταση του r είναι η

$r_{3q-1} r_{3q-2} \cdots r_0$

τότε οι δυαδικές παραστάσεις των i, j και k είναι:

$i \rightarrow r_{3q-1} \cdots r_{2q}$

$j \rightarrow r_{2q-1} \cdots r_q$

$k \rightarrow r_{q-1} \cdots r_0$

Κάθε επεξεργαστής P_r έχει 3 καταχωρητές A_r, B_r και C_r που συμβολίζονται αντίστοιχα $A(i,j,k)$, $B(i,j,k)$ και $C(i,j,k)$.

Αρχικά, ο επεξεργαστής P_s στη θέση $(0,j,k)$, $\begin{matrix} 0 \le j < n \\ 0 \le k < n \end{matrix}$ περιέχει τα στοιχεία a_{jk} και b_{jk} στους καταχωρητές A_s και B_s αντίστοιχα.

Οι καταχωρητές όλων των άλλων επεξεργαστών είναι αρχικά 0.

Στο τέλος του υπολογισμού ο C περιέχει τα c_{jk} όπου

$$c_{jk} = \sum_{i=0}^{n-1} a_{ji}\, b_{ik}$$

Ο αλγόριθμος σχεδιάζεται για να εκτελέσει ταυτόχρονα τους n^3 πολλαπλασιασμούς που επιτελούνται στον υπολογισμό των n^2 στοιχείων του C. Αυτό γίνεται σε 3 φάσεις.

Φάση 1 Τα στοιχεία των πινάκων A και B κατανέμονται στους n^3 επεξεργαστές. Άρα έχουμε $A(i,j,k) = a_{ji}$, $B(i,j,k) = b_{ik}$

Φάση 2 Υπολογίζονται τα γινόμενα $C(i,j,k) = A(i,j,k) * B(i,j,k)$

Φάση 3 Υπολογίζονται τα αθροίσματα $\sum_{i=0}^{n-1} C(i,j,k)$

Συμβολίζουμε με $\{N, r_m = d\}$ το σύνολο των ακεραίων r $0 \le r < N-1$ με δυαδική παράσταση $r_{3q-1} \cdots r_{m+1}\, \underline{d}\, r_{m-1} \cdots r_0$

procedure CUBE MATRIX MULTIPLICATION (A, B, C)

Φάση 1

Step 1: for $m = 3q - 1$ downto $2q$ do
  for all r in $\{N, r_m = 0\}$ do in parallel
    (1.1) $A_{r^{(m)}} = A_r$
    (1.2) $B_{r^{(m)}} = B_r$
  end for
end for.

Step 2: for $m = q - 1$ downto 0 do
  for all r in $\{N, r_m = r_{2q+m}\}$ do in parallel
    $A_{r^{(m)}} \leftarrow A_r$
  end for
end for.

Step 3: for $m = 2q - 1$ downto q do
  for all r in $\{N, r_m = r_{q+m}\}$ do in parallel
    $B_{r^{(m)}} \leftarrow B_r$
  end for
end for.

Φάση 2

Step 4: for $r = 1$ to N do in parallel
  $C_r \leftarrow A_r \times B_r$
end for.

Φάση 3

Step 5: for $m = 2q$ to $3q - 1$ do
  for $r = 1$ to N do in parallel
    $C_r \leftarrow C_r + C_{r^{(m)}}$
  end for
end for.

Τα βήματα 1, 2, 3 και 5 αποτελούνται από q επαναλήψεις σταθερού χρόνου, ενώ το βήμα 4 εκτελείται σε σταθερό χρόνο.

Άρα το procedure CUBE MATR MULT εκτελείται σε χρόνο $O(q)$ δηλαδή

$$\underline{t(n) = O(\log n)}$$

Θα αποδείξουμε τώρα ότι αυτός ο χρόνος εκτέλεσης είναι ο ταχύτερος που μπορούμε να πετύχουμε για ένα παράλληλο αλγόριθμο πολ/σμού δύο $n \times n$ πινάκων στον κύβο.

Παρατηρούμε ότι κάθε c_{ij} είναι άθροισμα n στοιχείων. Απαιτούνται $\Omega(\log n)$ βήματα για να υπολογιστεί αυτό το άθροισμα σε ένα δίκτυο ενδοεπικοινωνίας με n (ή περισσότερους) επεξεργαστές. Έστω S ο μικρότερος αριθμός βημάτων που απαιτείται σε ένα δίκτυο για τον υπολογισμό του αθροίσματος n αριθμών.

Κατά τη διάρκεια του τελικού βήματος, χρειάζεται ένας το πολύ επεξεργαστής για να εκτελέσει την τελευταία πρόσθεση και να παράγει το αποτέλεσμα.

- Κατά τη διάρκεια του βήματος $S-1$ χρειάζονται το πολύ 2 επεξεργαστές, κατά τη διάρκεια του βήματος $S-2$ χρειάζονται το πολύ 4 επεξεργαστές, κ.ο.κ.

Επομένως μετά από S βήματα, ο μέγιστος αριθμός των συνόλων προσθέσεων που μπορεί να εκτελεστούν είναι:

$$\sum_{i=0}^{S-1} 2^i = 2^S - 1$$

Η Φάση 1 του αλγορίθμου υλοποιείται στα βήματα 1-3.

Στο Βήμα 1 τα αρχικά δεδομένα στους $A(0,j,k)$ και $B(0,j,k)$ αντιγράφονται στους επεξεργαστές $P(i,j,k)$, όπου $1 \le i < n$ έτσι ώστε στο τέλος του βήματος 1 έχουμε:

$$A(i,j,k) = a_{jk} \quad \text{και} \quad B(i,j,k) = b_{jk}$$

$$0 \le i < n$$

Στο Βήμα 2 αντιγράφονται τα περιεχόμενα του $A(i,j,i)$ στους επεξεργαστές $P(i,j,k)$, έτσι ώστε στο τέλος του βήματος 2 έχουμε:

$$A(i,j,k) = a_{ji}, \qquad 0 \le k < n$$

Στο Βήμα 3 αντιγράφονται τα περιεχόμενα του $B(i,i,k)$ στους επεξεργαστές $P(i,j,k)$, έτσι ώστε στο τέλος του βήματος 3 έχουμε:

$$B(i,j,k) = b_{ik}, \qquad 0 \le j < n$$

Στο Βήμα 4 υπολογίζεται το γινόμενο $C(i,j,k) = A(i,j,k) \times B(i,j,k)$ ταυτόχρονα από τους επεξεργαστές $P(i,j,k)$, $0 \le i,j,k < n$

Στο Βήμα 5 υπολογίζονται ταυτόχρονα τα n^2 αθροίσματα

$$C(0,j,k) = \sum_{i=0}^{n-1} C(i,j,k)$$

Δοθέντος ότι χρειάζονται ακριβώς $n-1$ προσθέσεις για τον υπολογισμό του αθροίσματος n αριθμών έχουμε:

$$n-1 \leq 2^s - 1 \iff 2^s \geq n \iff \underline{s \geq \log n}$$

Επειδή $p(n) = n^3$ ο αλγόριθμος CUBE MATR MULT έχει

Κόστος $\underline{c(n) = O(n^3 \log n)}$

που είναι μεγαλύτερο από το χρόνο εκτέλεσης του ακολουθιακού αλγορίθμου.

Επομένως, ενώ ο πολλαπλασιασμός πινάκων στο κύβο είναι ταχύτερος από εκείνον στο διδιάστατο δίκτυο, το κόστος του είναι υψηλότερο λόγω του μεγάλου αριθμού επεξεργαστών που χρησιμοποιεί.

Παράδειγμα $n = 2^2$

$$A = \begin{bmatrix} 17 & 23 & 27 & 3 \\ 9 & 1 & 14 & 16 \\ 31 & 26 & 22 & 8 \\ 15 & 4 & 10 & 29 \end{bmatrix} \qquad B = \begin{bmatrix} -7 & -25 & -19 & -5 \\ -18 & -30 & -28 & -12 \\ -13 & -21 & -11 & -32 \\ -20 & -2 & -6 & -24 \end{bmatrix}$$

$N = 2^6$ επεξεργαστές $P_0, P_1, P_2, \ldots, P_{63}$

SIMD κύβος

Οι επεξεργαστές διατάσσονται σε ένα <u>3-διάστατο πίνακα</u> που στην πράξη είναι ένας <u>6-διάστατος κύβος</u> με συνδέσεις που παραλείπονται για λόγους απλότητας.

Καθένας από τους δείκτες i, j, k μοιράζεται 2 bits από τη δυαδική παράσταση $\underbrace{r_5 r_4}\ \underbrace{r_3 r_2}\ \underbrace{r_1 r_0}$ του δείκτη r του επεξεργαστή P_r δηλαδή είναι:

$\underline{i = r_5 r_4}$ $\underline{j = r_3 r_2}$ $\underline{k = r_1 r_0}$

Αφού $q=2$ το Βήμα 1 επαναλαμβάνεται 2 φορές για $m=5$ και $m=4$

Για $m=5$

Όλοι οι επεξεργαστές P_r με δυαδική παράσταση δείκτη

$r = r_5 r_4 r_3 r_2 r_1 r_0$ με $r_5 = 0$

αντιγράφουν τα περιεχόμενά τους στους επεξεργαστές με δυαδική παράσταση δείκτη $r^{(5)} = r_5' r_4 r_3 r_2 r_1 r_0$ με $r_5' = 1$.

Επομένως οι επεξεργαστές $P_0, P_1, \ldots, P_{15}$ αντιγράφουν τα περιεχόμενά τους στους επεξεργαστές $P_{16}, P_{17}, \ldots, P_{31}$, αντίστοιχα και ταυτόχρονα οι $P_{16}, P_{17}, \ldots, P_{31}$ αντιγράφουν τα αρχικά τους περιεχόμενα (όλα 0) στους $P_{48}, P_{49}, \ldots, P_{63}$, αντίστοιχα.

Για $m=4$

Όλοι οι επεξεργαστές P_r με δυαδική παράσταση δείκτη

$r = r_5 r_4 r_3 r_2 r_1 r_0$ με $r_4 = 0$

αντιγράφουν τα περιεχόμενά τους στους επεξεργαστές με δυαδική παράσταση δείκτη $r^{(4)} = r_5 r_4' r_3 r_2 r_1 r_0$ με $r_4' = 1$

Επομένως οι επεξεργαστές $P_0, P_1, \ldots, P_{15}$ αντιγράφουν τα περιεχόμενα τους στους $P_{16}, P_{17}, \ldots, P_{31}$, αντίστοιχα και ταυτόχρονα οι $P_{32}, P_{33}, \ldots P_{47}$ αντιγράφουν τα νέα περιεχόμενα τους (που απέκτησαν στην προηγούμενη επανάληψη) στους $P_{48}, P_{49}, \ldots, P_{63}$, αντίστοιχα.

Το **Βήμα 2** επαναλαμβάνεται 2 φορές

για **$m=1$** και **$m=0$**

Για **$m=1$**

Όλοι οι επεξεργαστές P_r με δυαδική παράσταση δείκτη

$r = r_5 r_4 r_3 r_2 r_1 r_0$ με **$r_1 = r_5$**

αντιγράφουν τα περιεχόμενα των καταχωρητών A στους καταχωρητές A των επεξεργαστών με δυαδική παράσταση δείκτη **$r_5 r_4 r_3 r_2 r_1' r_0$**.

Για παράδειγμα, οι P_0, P_1 αντιγράφουν τα περιεχόμενα των καταχωρητών A στους καταχωρητές A των P_2 και P_3, αντίστοιχα.

Για **$m=0$**

Όλοι οι επεξεργαστές P_r με δυαδική παράσταση δείκτη

$r = r_5 r_4 r_3 r_2 r_1 r_0$ με **$r_0 = r_4$**

αντιγράφουν τα περιεχόμενα των καταχωρητών A στους καταχωρητές A των επεξεργαστών με δυαδική παράσταση δείκτη **$r_5 r_4 r_3 r_2 r_1 r_0'$**.

Για παράδειγμα, οι P_0, P_2 αντιγράφουν τα περιεχόμενα των καταχωρητών A στους καταχωρητές A των P_1 και P_3.

Στο τέλος του βήματος αυτού κάθε στοιχείο του A έχει γίνει αντίγραφο διαμέσου κάθε γραμμής στο Σχ. (α).

Το <u>Βήμα 3</u> είναι ισοδύναμο με το Βήμα 2 εκτός του ότι σ'αυτό αντιγράφεται ένα στοιχείο του B διαμέσου κάθε στήλης. Τα περιεχόμενα των 64 επεξεργαστών μετά το τέλος των Βημάτων 2 και 3 φαίνονται στο Σχ. (d)

Στο <u>Βήμα 4</u>, όλοι οι επεξεργαστές λειτουργούν <u>ταυτόχρονα</u> και κάθε επεξεργαστής υπολογίζει το γινόμενο των A και B καταχωρητών του και καταχωρεί το αποτέλεσμα στον καταχωρητή τους C.

Το <u>Βήμα 5</u> επαναλαμβάνεται 2 φορές

για <u>m=4</u> και <u>m=5</u>

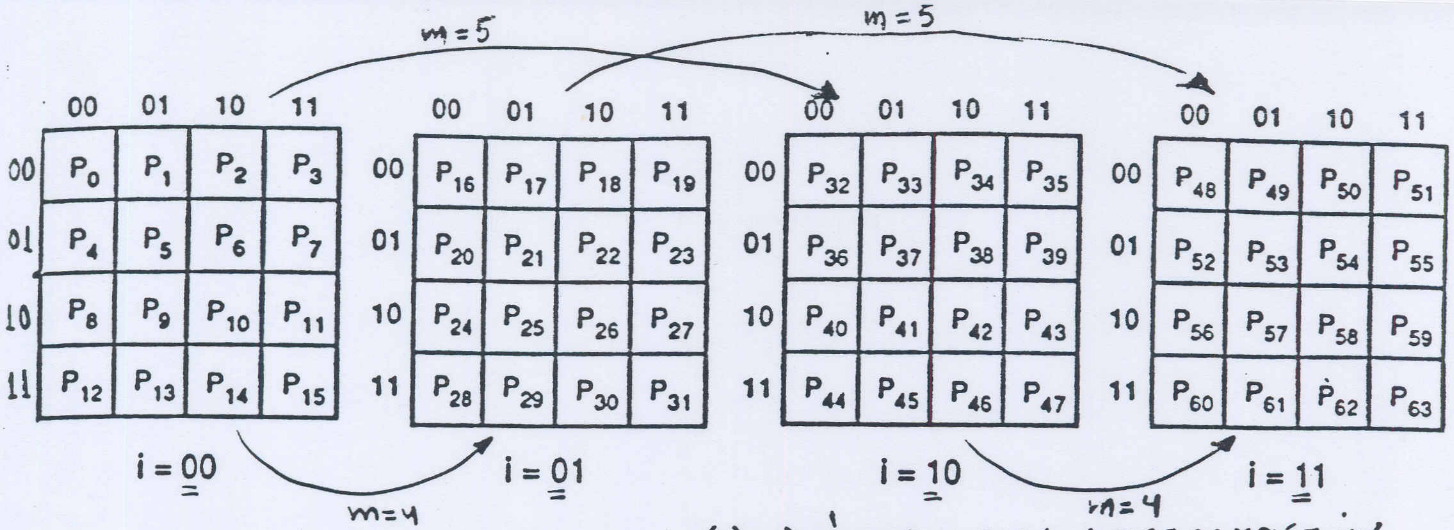
Για <u>m=4</u>

Προστίθενται τα περιεχόμενα των καταχωρητών C των ζευγών επεξεργαστών των οποίων οι δυαδικές παραστάσεις των δεικτών τους διαφέρουν στο <u>bit r_4</u>.
Και οι δυό επεξεργαστές διατηρούν το αποτέλεσμα.

Για <u>m=5</u>

Γίνεται το ίδιο με την προηγούμενη επανάληψη για τους επεξεργαστές με δείκτες που διαφέρουν στο <u>bit r_5</u>.

Το τελικό αποτέλεσμα αποθηκεύεται στους επεξεργαστές $P_0, P_1, \ldots, P_{15}$, όπως φαίνεται στο Σχ. (e)



(a) Διάταξη των επεξεργαστών σε ένα 3-διάστατο πίνακα

17 -7	23 -25	27 -19	3 -5
9 -8	1 -30	14 -28	16 -12
31 -13	26 -21	22 -11	8 -32
15 -20	4 -2	10 -6	29 -24

(b) Αρχική καταχώρηση των A και B στους καταχωρητές $P_0, P_1, \ldots, P_{15}$

17 -7	23 -25	27 -19	3 -5
9 -18	1 -30	14 -28	16 -12
31 -13	26 -21	22 -11	8 -32
15 -20	4 -2	10 -6	29 -24

17 -7	23 -25	27 -19	3 -5
9 -18	1 -30	14 -28	16 -12
31 -13	26 -21	22 -11	8 -32
15 -20	4 -2	10 -6	29 -24

17 -7	23 -25	27 -19	3 -5
9 -18	1 -30	14 -28	16 -12
31 -13	26 -21	22 -11	8 -32
15 -20	4 -2	10 -6	29 -24

17 -7	23 -25	27 -19	3 -5
9 -18	1 -30	14 -28	16 -12
31 -13	26 -21	22 -11	8 -32
15 -20	4 -2	10 -6	29 -24

(c) Τα περιεχόμενα των 64 επεξεργαστών μετά το τέλος του Βήματος 1

17 -7	17 -25	17 -19	17 -5
9 -7	9 -25	9 -19	9 -5
31 -7	31 -25	31 -19	31 -5
15 -7	15 -25	15 -19	15 -5

23 -18	23 -30	23 -28	23 -12
1 -18	1 -30	1 -28	1 -12
26 -18	26 -30	26 -28	26 -12
4 -18	4 -30	4 -28	4 -12

27 -13	27 -21	27 -11	27 -32
14 -13	14 -21	14 -11	14 -32
22 -13	22 -21	22 -11	22 -32
10 -13	10 -21	10 -11	10 -32

3 -20	3 -2	3 -6	3 -24
16 -20	16 -2	16 -6	16 -24
8 -20	8 -2	8 -6	8 -24
29 -20	29 -2	29 -6	29 -24

(d) Τα περιεχόμενα των 64 επεξεργαστών μετά το τέλος των Βημάτων 2 και 3

-944	-1688	-1282	-1297
-583	-581	-449	-889
-1131	-2033	-1607	-1363
-887	-763	-681	-1139

(e)

2.3 Πολλαπλασιασμός πινάκων σε ένα CRCW SM SIMD υπολογιστή

Υποθέτουμε ότι οι αλληλοσυγκρούσεις για γράψιμο αντιμετωπίζονται ως εξής:

Όταν μερικοί επεξεργαστές επιδιώκουν να γράψουν στην ίδια θέση μνήμης τότε το άθροισμα των αριθμών που είναι για γράψιμο αποθηκεύεται σ' αυτή τη θέση.

Ο αλγόριθμος είναι μια άμεση παραλληλοποίηση του ακολουθιακού procedure MATRIX MULTIPLICATION.

Χρησιμοποιεί $n \times n \times k$ επεξεργαστές για τον πολ/σμό ενός $n \times n$ πίνακα A με ένα $n \times k$ πίνακα B.

Συντομία οι επεξεργαστές διατάσσονται σε ένα $n \times n \times k$ πίνακα του οποίου ο κάθε επεξεργαστής έχει τρεις δείκτες

(i, j, s), $1 \le i \le n$, $1 \le j \le n$, $1 \le s \le k$

Αρχικά οι πίνακες A και B βρίσκονται στη κοινή μνήμη.

Όταν ο αλγόριθμος τελειώσει το γινόμενο τους $C = A \cdot B$ βρίσκεται επίσης στη κοινή μνήμη.

procedure CRCW MATRIX MULTIPLICATION (A, B, C)

```
for i = 1 to m do in parallel
  for j = 1 to k do in parallel
    for s = 1 to n do in parallel
      (1) c_ij ← 0
      (2) c_ij ← a_is × b_sj
    end for
  end for
end for.
```

Ανάλυση

Πλήθος επεξεργαστών $p(n) = n^3$

Χρόνος εκτέλεσης $t(n) = O(1)$

Κόστος $c(n) = p(n) \times t(n)$

$= n^3 \times O(1)$

$= O(n^3)$

Το κόστος $c(n)$ είναι το ίδιο με εκείνο του ακολουθιακού αλγορίθμου

Παράδειγμα $n = 4$

$P(n) = 4^3 = 64$ επεξεργαστές

Όλα τα 16 γινόμενα που φαίνονται στο σχήμα (d) υπολογίζονται ταυτόχρονα και αποθηκεύονται (προσθέτονται) σε σημάδες των 4 στη κατάλληλη θέση του C.

Για παράδειγμα, οι P_1, P_2, P_3, P_4 υπολογίζουν αντίστοιχα τα γινόμενα $17 \times (-7)$, $23 \times (-18)$, $27 \times (-13)$ και $3 \times (-20)$ και αποθηκεύουν το αποτέλεσμα στο C_{11} που είναι $C_1 = -944$

3. Πολλαπλασιασμός πίνακα με διάνυσμα

$$(\underset{m\times n}{A}\ ,\ \underset{n\times 1}{U}) \longrightarrow \underset{m\times 1}{V} = AU$$

$$v_i = \sum_{j=1}^{n} a_{ij} u_j\ , \qquad 1 \le i \le m$$

Παράδειγμα : $m = 3$, $n = 4$

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}}_{U} = \underbrace{\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}}_{V}$$

Παράλληλοι αλγόριθμοι

1. Πολλαπλασιασμός σε μονοδιάστατο πίνακα
2. >> σε δέντρο
3. Συνέλιξη (convolution)

3.1 Πολλαπλασιασμός πίνακα με διάνυσμα σε μονοδιάστατο πίνακα (linear array)

Έστω $\underline{A}$ $m \times n$, $\underline{U}$ $n \times 1$ $\Rightarrow \underline{V} = AU$ $m \times 1$

$$V_i = \sum_{j=1}^{n} a_{ij} u_j \,, \qquad 1 \le i \le m$$

Έστω $\underline{m}$ επεξεργαστές $\underline{P_1, P_2, \ldots, P_m}$

Ο επεξεργαστής $\underline{P_i}$ χρησιμοποιείται για τον υπολογισμό του στοιχείου $\underline{V_i}$ του V.

Κάθε επεξεργαστής P_i έχει $\underline{3}$ καταχωρητές

$\underline{\alpha}$ $\underline{u}$ $\underline{v}$

Όταν ο P_i δέχεται δύο εισόδους a_{ij} και u_j

(i) αποθηκεύει το a_{ij} στον α
και το u_j στον u

(ii) πολλαπλασιάζει το α με το u

(iii) προσθέτει το αποτέλεσμα στο V_i και

(iv) στέλνει το u_j στον P_{i-1} εκτός αν $i=1$.

Ανάλυση

Το στοιχείο a_{1n} χρειάζεται $m+n-1$ βήματα για να φθάσει στον P_1. Αφού ο P_1 είναι ο τελευταίος επεξεργαστής που εργάζεται για να τελειώσει ο αλγόριθμος αυτά τα βήματα απαιτούνται για να υπολογιστεί το γινόμενο

Αν $m \leq n$ τότε ο χρόνος εκτέλεσης του αλγορίθμου είναι

$$t(n) = O(n)$$

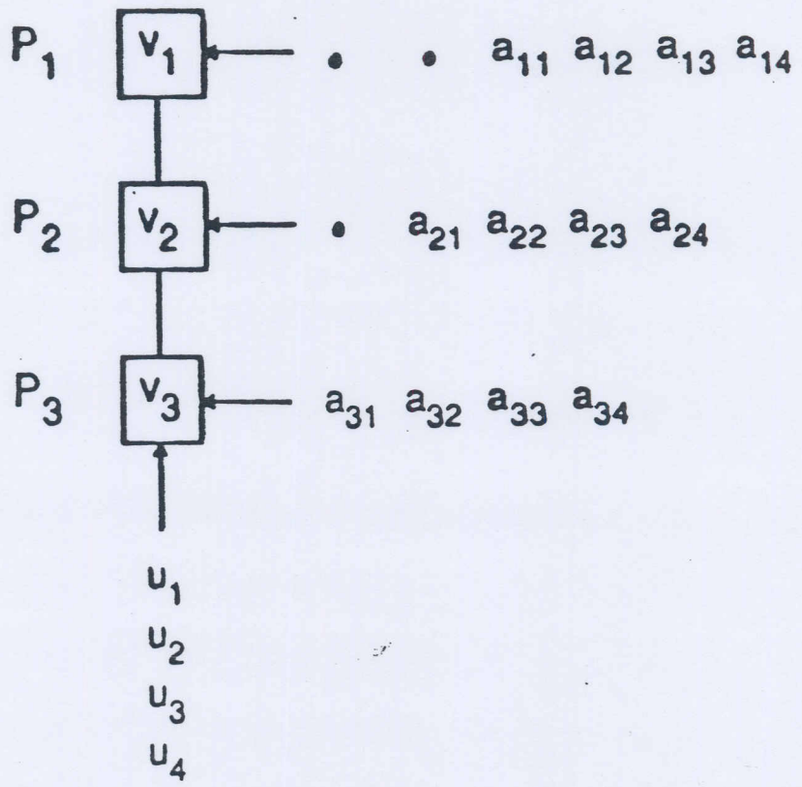
Αφού χρησιμοποιούνται m επεξεργαστές, ο αλγόριθμος έχει κόστος $C(n) = O(n^2)$

που είναι βέλτιστο αφού για να διαβαστεί πίνακας απαιτούνται $\Omega(n^2)$ βήματα.

Παράδειγμα

$$m = n = 2$$

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad U = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$



procedure LINEAR MV MULTIPLICATION (A, U, V)

for $i = 1$ **to** m **do in parallel**

(1) $v_i \leftarrow 0$

(2) **while** P_i receives two inputs a and u **do**

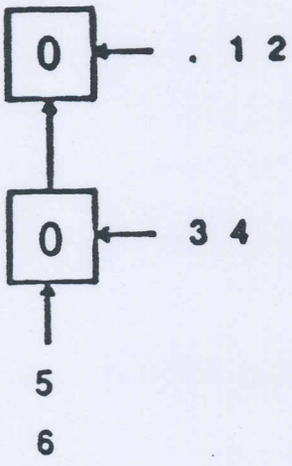
(2.1) $v_i \leftarrow v_i + (a \times u)$

(2.2) **if** $i > 1$ **then send** u **to** P_{i-1}

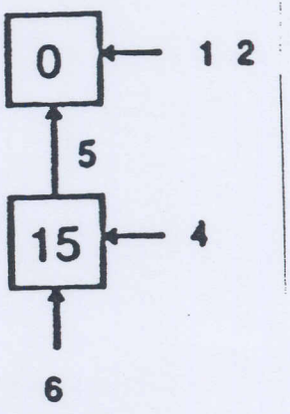
end if

end while

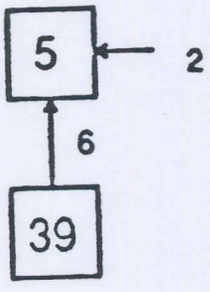
end for.



(a)



(b)



(c)

(d)

3.2 Πολλαπλασιασμός πίνακα με διάνυσμα σε δέντρο.

Έστω n τερματικοί (φύλλα) επεξεργαστές $P_1, P_2, \ldots, P_n$

$n-2$ ενδιάμεσοι $P_{n+1}, P_{n+2}, \ldots, P_{2n-2}$

και 1 επεξεργαστής ρίζα P_{2n-1}.

Οι ακραίοι (τερματικοί) επεξεργαστές P_i, $1 \le i \le n$ αποθηκεύουν το u_i κατά την εκτέλεση του αλγορίθμου.

Ο πίνακας A προωθείται στο δέντρο γραμμή-γραμμή ένα στοιχείο ανά επεξεργαστή (ακραίο).

- Όταν ο ακραίος επεξεργαστής P_i δέχεται το a_{ji} υπολογίζει το γινόμενο $u_i \times a_{ji}$ και στέλνει το γινόμενο στους γονείς του.

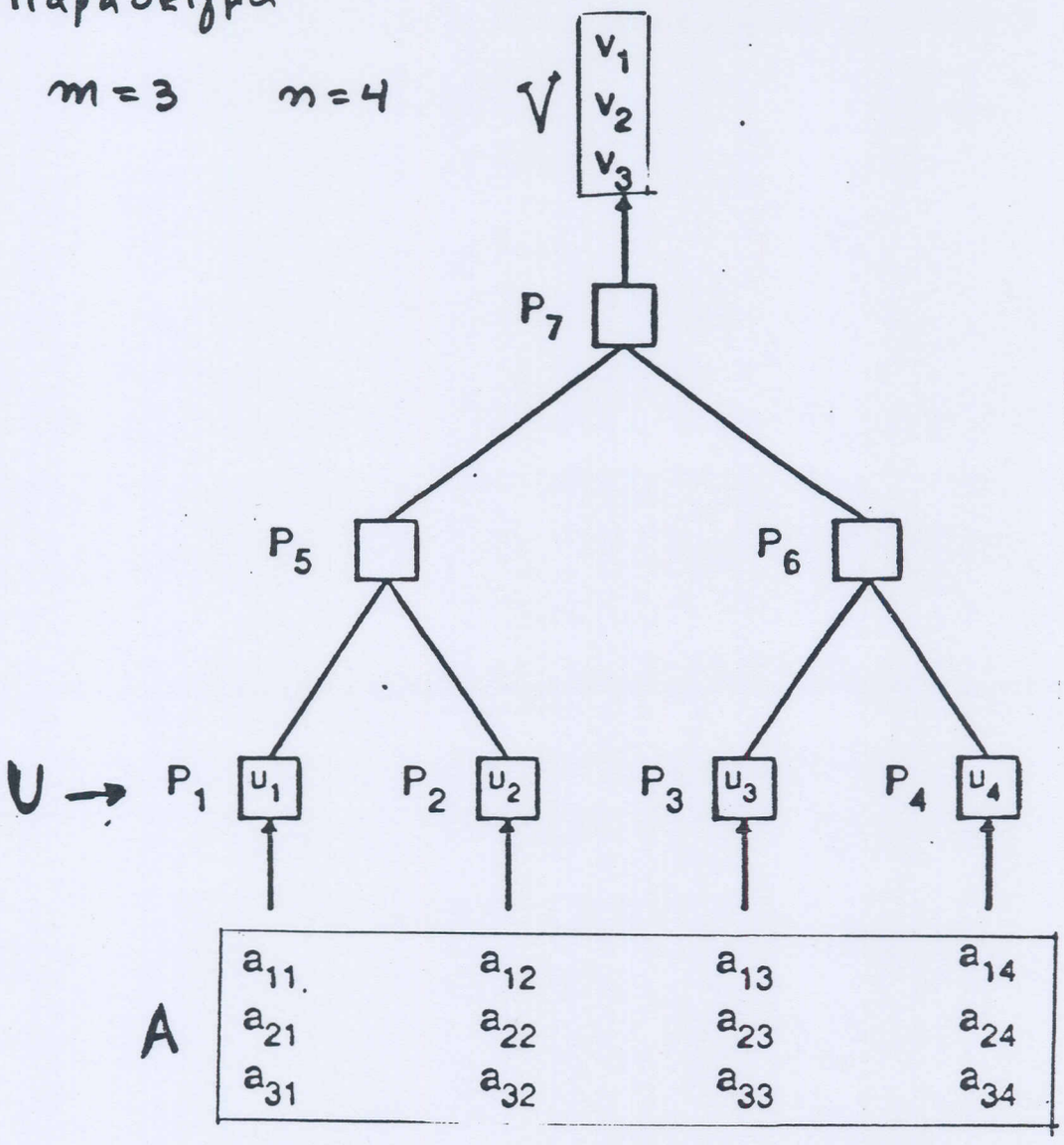
- Όταν ένας ενδιάμεσος επεξεργαστής P_k ή η ρίζα δέχεται δύο εισόδους από τα παιδιά του, τις προσθέτει και στέλνει το αποτέλεσμα στους γονείς του.

Το στοιχείο v_j παράγεται από την ρίζα.

Αν οι γραμμές του A εισέρχονται στους ακραίους P σε επαρχείς χρονικές στιγμές, τότε τα στοιχεία του γινομένου V επίσης παράγονται ως έξοδος από τη ρίζα σε διαδοχικές χρονικές στιγμές.

Παράδειγμα

$m = 3$ $n = 4$



procedure TREE MV MULTIPLICATION (A, U, V)

do steps 1 and 2 in parallel

(1) for $i = 1$ to n do in parallel

for $j = 1$ to m do

(1.1) compute $u_i \times a_{ji}$

(1.2) send result to parent

end for

end for

(2) for $i = n + 1$ to $2n - 1$ do in parallel

while P_i receives two inputs do

(2.1) compute the sum of the two inputs

(2.2) if $i < 2n - 1$ then send the result to parent

else produce the result as output

end if

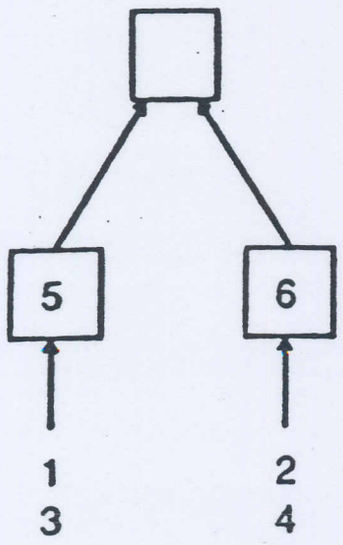
end while

end for.

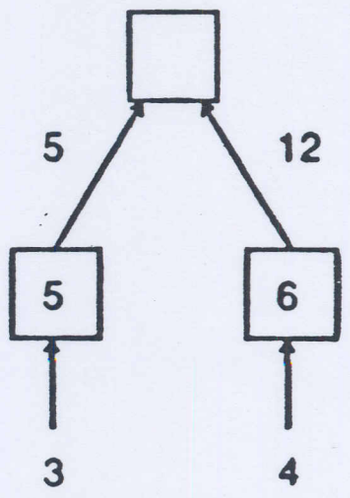
Παράδειγμα

$m = n = 2$

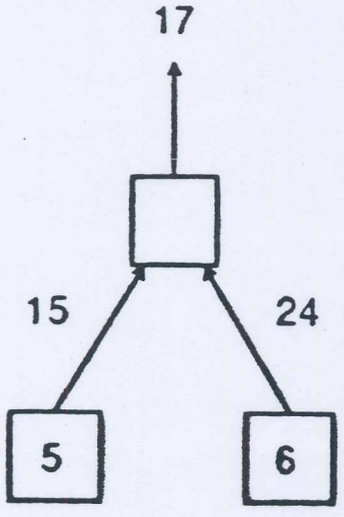
$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad U = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$



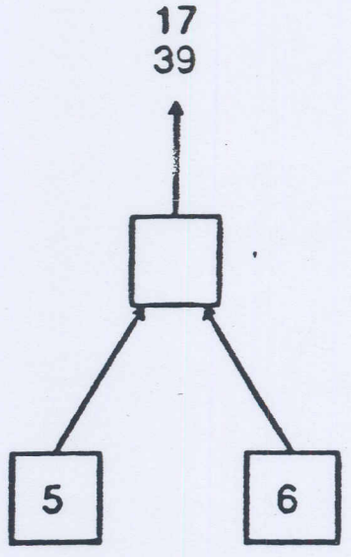
(a)



(b)



(c)



(d)

Ανάλυση

Από τη στιγμή που η πρώτη γραμμή του A εισαχθεί στους αρχαίους επεξεργαστές έως ότου να παραχθεί το V_1 στη ρίζα απαιτούνται $\log n$ βήματα.

Ακριβώς $m-1$ βήματα απαιτούνται για να παραχθεί το V_m από τη ρίζα.

Άρα ο αλγόριθμος απαιτεί $m-1+\log n$ βήματα με ένα κόστος $C(n) = O(n^2)$ αν $m \leq n$

Επομένως το procedure TREE MV MULT είναι ταχύτερο από το procedure LINEAR MV MULT εφόσον χρησιμοποιείται περίπου διπλάσιος αριθμός επεξεργαστών.

Το κόστος είναι βέλτιστο εφόσον απαιτούνται $\Omega(n^2)$ βήματα για να διαβαστεί η είσοδος ακολουθιακά.

3.3 Συνέλιξη (convolution)

Έστω μια ακολουθία σταθερών

$$\{w_1, w_2, \ldots, w_n\}$$

και μια ακολουθία εισόδου

$$\{x_1, x_2, \ldots x_n\}$$

Ζητείται να υπολογιστεί η ακολουθία εξόδου

$$\{y_1, y_2, \ldots, y_{2n-1}\}$$

που ορίζεται από τον τύπο:

$$y_i = \sum_{j=1}^{n} x_{i-j+1} \times w_j \qquad 1 \le i \le 2n-1$$

Ο υπολογισμός αυτός είναι γνωστός ως **συνέλιξη** και είναι πολύ ενδιαφέρων στην **ψηφιακή επεξεργασία σήματος**.

Παράδειγμα $n = 3$

$$\begin{bmatrix} x_1 & 0 & 0 \\ x_2 & x_1 & 0 \\ x_3 & x_2 & x_1 \\ 0 & x_3 & x_2 \\ 0 & 0 & x_3 \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}$$