

Πολλαπλασιασμός πίνακων

Ο πολλαπλασιασμός πίνακων αποτελεί μία από τις υλασιμότερες και πλέον συχνά χρησιμοποιούμενες αλγεβρικές πράξεις σε προβλήματα επιτεφμουισμού υπολογιστού. Μπορεί δε να παραλληλοποιθεί αρκετά εύκολα σε ότι αφορά τους απαιτούμενους υπολογιστού, ειδικότερα σε περιβάλλον κοινών μνήμης, καθώς οι απαιτούμενοι υπολογιστού για κάθε στοιχείο του πίνακα αποτελέσματος είναι ανεξαρτητοι μεταξύ τους.

$$\begin{pmatrix} m \times n & n \times k \\ A & B \end{pmatrix} \longrightarrow C = A * B$$

$$\text{όπου } C_{ij} = \sum_{s=1}^n a_{is} b_{sj}, \quad 1 \leq i \leq m, \quad 1 \leq j \leq k$$

Κόστος υλασιμού ακολουθιακού αλγορίθμου: $O(mnk)$

ή $O(n^3)$ αν $m \leq n$ και $k \leq n$.

Ακολουθιακός αλγόριθμος πολλαπλασιασμού πίνακων

procedure MATRIX_MULTIPLICATION(A, B, C)

for $i=1$ to m do

for $j=1$ to k do

(1). $C_{ij} \leftarrow 0$

(2). for $s=1$ to n do

$C_{ij} \leftarrow C_{ij} + a_{is} b_{sj}$

end for

end for

Αλγόριθμοι πολλαπλασιασμού πινάκων με καλύτερες ασυμπτωτικές αποδοτικότητες ποσοτικοποιούνται είναι διαθέσιμες; Για παράδειγμα, ο αλγόριθμος του Strassen με $O(n^{2.31})$

Γενικότερα, υπάρχουν κάποιοι αποδοτικοί αλγόριθμοι πολλαπλασιασμού πινάκων των οποίων ο χρόνος εκτέλεσης είναι $O(n^x)$, όπου $2 \leq x \leq 3$

Στη συνέχεια παρουσιάζονται αλγόριθμοι των οποίων το κόστος ταυριάζει με το χρόνο εκτέλεσης της MATRIX-MULTIPLICATION.

Ο παραπάνω αναδρομικός τρόπος εκφραγής του πολλαπλασιασμού πινάκων έχει αποδειχθεί ότι επιτυγχάνει καλύτερη εκμετάλλευση της αρυθμικής μνήμης του επεξεργαστή όπου λαμβάνει χώρα η εκτέλεση, και οδηγεί συνήθως σε καλύτερους χρόνους απόκρισης για μεγάλα μεγέθη του πίνακα B. Αποτέλεσε επίσης όπως θα δούμε και παρακάτω

τη βάση για την αποδοτική γενίκευση των παράλληλων αλγορίθμων πολλαπλασιασμού πινάκων σε τοπολογία διδιδάκτου πλεγματός. Σε αυτές τις περιπτώσεις μιλάμε για block πολλαπλασιασμό πινάκων. Στις block μεθόδους, κάθε πίνακας $A^{n \times n}$ θεωρείται ότι είναι ένας $q \times q$ πίνακας με εσωκείμενα blocks A_{ij} ($i, j = 0, 1, \dots, q-1$) όπου κάθε block είναι ένας $n/q \times n/q$ υποπίνακας

Παράδειγμα: $n=4$, $q=2$

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

Για τον παρακάτω υλοποιημένο του block αλγόριθμου συνήθως επιλέγουμε $q = \sqrt{p} \Rightarrow p = q^2$ επεξεργαστές.

Αλγόριθμος block πολλαπλασιασμού πινάκων

1. procedure BLOCK-MAT-MULT (A, B, C)

2. for $i=0$ to $q-1$ do

3. for $j=0$ to $q-1$ do

4. $C_{ij} = 0$

5. for $s=0$ to $q-1$ do

6. $C_{ij} = C_{ij} + A_{is} * B_{sj}$

7. end for

8. end for

9. end for.

Παράδειγμα block πολλαπλασιασμού πινάκων

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} * \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix}$$

$$\text{ή} \quad \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} * \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix} = \begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix}$$

$$\text{όπου} \quad C_{00} = A_{00} * B_{00} + A_{01} * B_{10}$$

$$= \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix} + \begin{pmatrix} a_{02} & a_{03} \\ a_{12} & a_{13} \end{pmatrix} \begin{pmatrix} b_{20} & b_{21} \\ b_{30} & b_{31} \end{pmatrix} = \begin{pmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{pmatrix}$$

Στους block αριθμούς u και v η A θεωρείται ότι είναι ένας $q \times q$ πίνακας με στοιχεία υποπίνακες (blocks) A_{ij} ($i, j = 0, 1, \dots, q-1$) όπου το κάθε block είναι ένας $\frac{q}{r} \times \frac{q}{r}$ υποπίνακας. Στην παράλληλη υλοποίηση ενός block αριθμού συνήθως επιλέγουμε $q = \sqrt{p}$ και $p = q^2$ επεξεργαστές. Έτσι λοιπόν, ο κάθε επεξεργαστής P_{ij} αναλαμβάνει του υπολογισμό του αντίστοιχου υποπίνακα C_{ij} του πίνακα γινομένου $C = A * B$.

1. Ένας απλός Παράλληλος αλγόριθμος

Θεωρούμε ότι οι δύο $n \times n$ πίνακες A και B είναι διαχωρισμένοι σε p τετραγωνικούς $(n/\sqrt{p}) \times (n/\sqrt{p})$

υποπίνακες A_{ij} και B_{ij} ($i, j = 0, 1, \dots, \sqrt{p}-1$).

Οι p αυτοί υποπίνακες αντιστοιχίζονται σε ένα $\sqrt{p} \times \sqrt{p}$ λογικό πλέγμα p επεξεργαστών $P_{00}, \dots, P_{\sqrt{p}-1, \sqrt{p}-1}$

Ο επεξεργαστής P_{ij} αρχικά αποθηκεύει τους υποπίνακες A_{ij} και B_{ij} και μετά υπολογίζει τον υποπίνακα C_{ij} του πίνακα γινομένου C .

Για τον υπολογισμό του υποπίνακα C_{ij} , ο επεξεργαστής χρειάζεται να αποκτήσει όλους τους υποπίνακες A_{ik} και B_{kj} ($k = 0, 1, \dots, \sqrt{p}-1$). Αυτό επιτυγχάνεται με μία αποστολή όλων προς όλους των υποπινάκων του A στην κάθε γραμμή επεξεργαστών και με μία αποστολή όλων προς όλους των υποπινάκων του B στην κάθε στήλη επεξεργαστών. Αφού ο κάθε επεξεργαστής P_{ij} αποκτήσει τους υποπίνακες $A_{i0}, A_{i1}, \dots, A_{i, \sqrt{p}-1}$ και $B_{0j}, B_{1j}, \dots, B_{\sqrt{p}-1, j}$ ανασυνθέτει τον πολλαπλασιασμό και την πρόσθεση των υποπινάκων (βλ. γραφείς 5, 6 στον αλγόριθμο BLOCK-MAT-MULT).

Υπερκύβος

Υποθέτουμε ότι το λογικό πλέγμα των επεξεργαστών είναι εμμετρεμένο
 δ' ένα υπερκύβο με p επεξεργαστές. Ο αλγόριθμος απαιτεί δύο
 βήματα αποστολής μηνυμάτων όλων προς όλους (όσο και αν βήμα
 εκτελούνται $\sqrt{p}$ ταυτόχρονα αποστολές μηνυμάτων σε όλες τις
 γραμμές και βύθες του πλέγματος επεξεργαστών) μεταξύ των ορόφων
 με $\sqrt{p}$ επεξεργαστές. Τα μηνύματα είναι υπονόμενα με n^2/p βίτσια.
 Ο συνολικός χρόνος επικοινωνίας είναι

$$T_p^{\text{comm}} = 2(t_s \log \sqrt{p} + t_w \underbrace{\frac{n^2}{p}}_{\substack{\text{ων} \\ \text{πληθος} \\ \text{αεζων} \\ \text{που εκω}}} \underbrace{(\sqrt{p} - 1)}_{\substack{\text{διαμετρος} \\ \text{στο πλεγμα}}})$$

Στη συνέχεια, ο κάθε επεξεργαστής υπολογίζει έναν υπονόμενα C_{ij}
 ο οποίος απαιτεί $\sqrt{p}$ πολλαπλασιασμούς $(n/\sqrt{p}) \times (n/\sqrt{p})$ υπονόμενων. Αυτό
 γίνεται σε υπολογιστικό χρόνο

$$T_p^{\text{comp}} = \sqrt{p} \times \underbrace{\left(\frac{n}{\sqrt{p}} \right)^2}_{\substack{\text{υποδος} \\ \text{υπονομα}}} = n^2/p$$

Επομένως, ο συνολικός χρόνος της παραλληλως διατρέξης είναι
 προεξοχικά

$$T_p^{\text{total}} = \frac{n^3}{p} + t_s \log p + 2t_w \frac{n^2}{\sqrt{p}}$$

Αρα, το υότος του απλού παράλληλου αλγόριθμου είναι:

$$p \cdot T_p^{\text{total}} = n^3 + t_s p \log p + 2t_w n^2 \sqrt{p}$$

το οποίο είναι βέλτιστο για $p = O(n^2)$.

Πλέγμα

Αν χρησιμοποιήσουμε ένα αναδιηλούμενο $\sqrt{p} \times \sqrt{p}$ πλέγμα επεξεργαστών αυτών του υπερύβου (μ. αποστολή μηνυμάτων γίνεται με αποδιμευση και προώθηση) τότε ο χρόνος της παράλληλης διαστροφής επιρρεάζεται μόνο στο συντελεστή του t_s και είναι

$$T_p^{mesh} = \frac{n^3}{p} + 2t_s \sqrt{p} + 2t_w \frac{n^2}{\sqrt{p}}$$

Αν τα μηνύματα είναι μεγάλα, τότε ο χρόνος της παράλληλης επεξεργασίας του αυτού αλγορίθμου είναι ο ίδιος και $\neq$ μη αναδιηλούμενο πλέγμα που υποστηρίζει αποστολή μηνυμάτων με διαχωρισμό και απόδοση και επικοινωνία δύο κατευθύνσεων.

Πολυπλοκισμός πίνακων σε τοπολογία Δαυινίου

Ένας απλός αλγόριθμος ο οποίος επιτυγχάνει σε καυσοποιητικό βαθμό τους βασικούς στόχους ενός παράλληλου αλγορίθμου πολυπλοκισμού πίνακων, μπορεί να διατυπωθεί ακολουθώντας: (i) όσον αφορά την απαιτούμενη βέλτιστη συμπεριφορά από άποψη χώρου μνήμης, την ευθεία κατανομή των πολυπλοκισμένων πίνακων στους πολυπλοκιστές επεξεργαστές κατά γραμμές και (ii) εν συνεχεία τη διατύπωση των απαιτούμενων επιπλέον δεδομένων για τους υπολογιστές κάθε επεξεργαστή, προσδεντικά μέσω του δικτύου σε αντίστοιχα συστήματα βιβλίου επιμορφωσίας. Μπορεί δε να εκτελεστεί αποδοτικά σε μια απλή τοπολογία Δαυινίου.

Αν υποθέτουμε αρχικά ότι έχουμε στη διάθεση μας αριθμό επεξεργαστών ίσο με τη διάσταση των πολυπλοκισμένων πίνακων ($n \times p$) οι οποίοι είναι οργανωμένοι σε τοπολογία δαυινίου. Έτσι όπως ότι κάθε επεξεργαστής P_i έχει αρχικά στην κατοχή του μια γραμμή του πίνακα A (τη γραμμή τάξης i , δηλ. τα στοιχεία $a_{ij} \mid j=0, 1, \dots, n-1$) και μια γραμμή του πίνακα B (τη γραμμή τάξης i , δηλ. τα στοιχεία $b_{ij} \mid j=0, \dots, n-1$), και ότι είναι υπεύθυνος για τον υπολογισμό των στοιχείων μιας γραμμής του πίνακα αποτελέσματος C (τη γραμμή τάξης i , δηλ. τα στοιχεία $c_{ij} \mid j=0, \dots, n-1$). Ο αλγόριθμος συντίθεται από τα ακόλουθα βήματα:

1. Κάθε επεξεργαστής P_i με βάση τις γραμμές του πίνακα A και B που κατέχει, εκτελεί ότι υπολογισμούς είναι χρήσιμοι για τον υπολογισμό των στοιχείων της γραμμής του πίνακα -

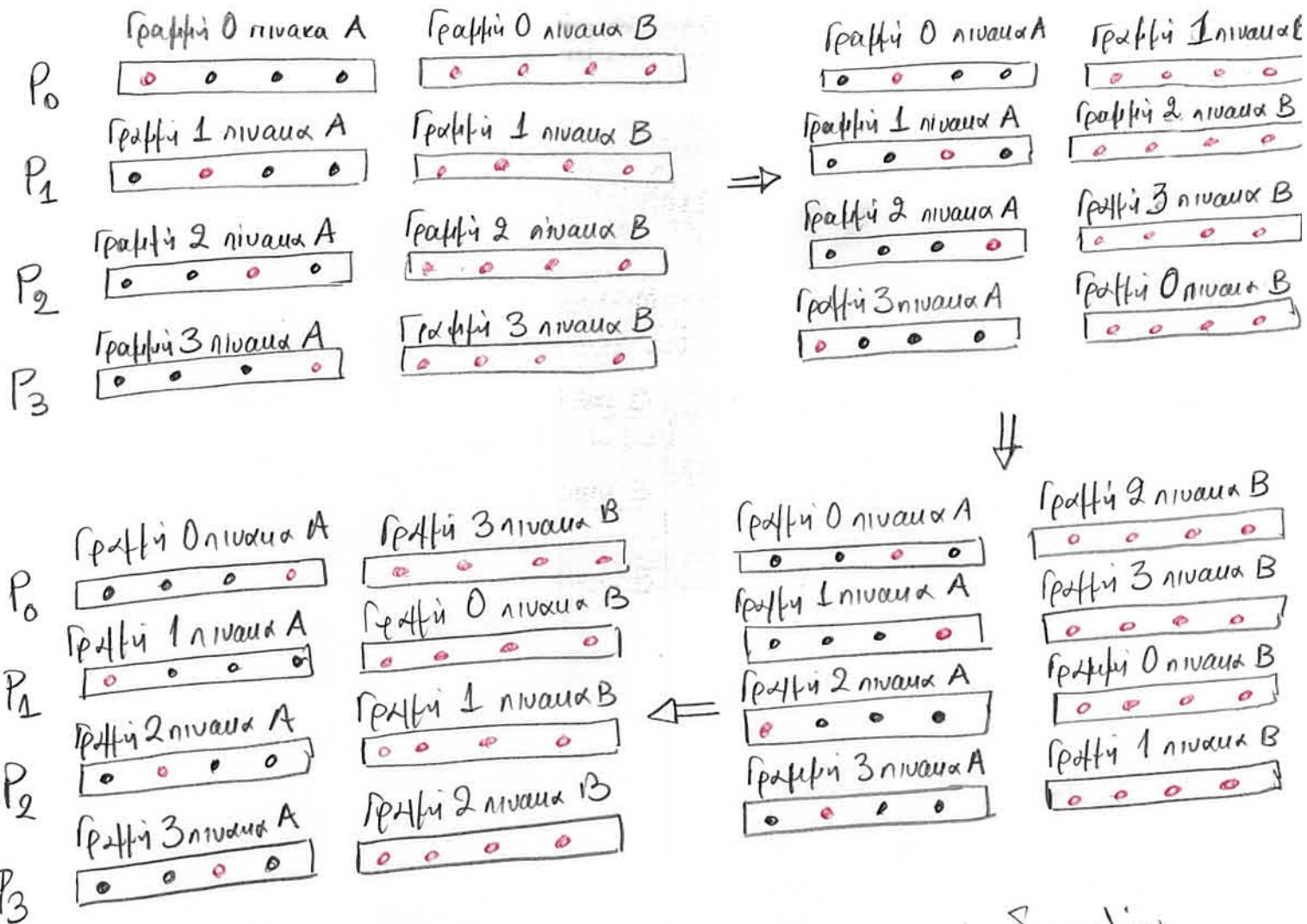
- αποτελέσματος C για την οποία είναι υπεύθυνος.

2. Κάθε επεξεργαστής P_i δέχεται τη γραφή του πίνακα B που κατέχει στον προηγούμενο του επεξεργαστή P_{i-1} (ήτοι P_{p-1} αν $i=0$) στην τοπολογία δακτυλίου. Έτσι κάθε επεξεργαστής φέροντας τον βύσματός αυτού, θα έχει λάβει μια νέα (την επόμενη στη σειρά) γραφή του πίνακα B με βάση την οποία θα συνεχίσει τους υπολογισμούς του.

3. Κάθε επεξεργαστής P_i επαναλαμβάνει τα βήματα 1 και 2 άλλες $p-2$ φορές (συνολικά $p-1$ επαναλήψεις), προκειμένου να παραλάβει σταδιακά και να πραγματοποιήσει υπολογισμούς με όλες τις γραφές του πίνακα B . Μετά από την τελευταία επανάληψη, επαναλαμβάνει άλλη μια φορά το βήμα 1 σφουγγίζοντας έτσι τους απαιτούμενους υπολογισμούς με βάση την τελευταία γραφή του πίνακα B που παρέλαβε από τον προηγούμενο επεξεργαστή.

Ειδικότερα, σε ότι αφορά το βήμα υπολογισμών (βήμα 1), κάθε επεξεργαστής P_i κατά την πρώτη επανάληψη εκτελεί για κάθε στοιχείο της γραμμής του πίνακα B που κατέχει ένα λογισμικό (και μία πρόθεση) με το στοιχείο τάξης i της γραμμής του πίνακα A που κατέχει, κατά τη 2η επανάληψη με το στοιχείο τάξης $i+1$ κλπ, και γενικά με το στοιχείο $(i+step)\%n$ όπου $step$ ο κύβων αριθμός της επανάληψης ($0 \dots p-1$)

Η παραπάνω ακολουθία λειτουργιών σε κάθε επανάληψη παρουσιάζεται στο ακόλουθο παράδειγμα



Σχ: Το Ψανδαδικός πίνακων σε τοπολογία δακτύλιου

Σημείωση: Αντίστοιχος αλγόριθμος μπορεί να διατυπωθεί (αφήνεται σαν άσκηση) και με την υπόθεση ότι αρχικά οι δύο πίνακες εισόδου A και B είναι κατανεμημένοι στους πολλαπλούς επεξεργαστές κατά τύπους και κάθε επεξεργαστής έχει ως έργο να υπολογίζει μια σειρά του πίνακα αποτελέσματος

Ο παραπάνω αλγόριθμος μπορεί εύκολα επίσης να μεταθεί και για την περίπτωση που ο αριθμός των επεξεργασιών είναι μικρότερος από το n . Σε αυτή την περίπτωση υποθέτουμε χωρίς βλάβη της γενικότητας ότι το n είναι ακέραιο πολλαπλάσιο του p , θα πρέπει να θεωρήσουμε ματαρχήν ότι κάθε επεξεργασία P_i έχει αρχικά στην κατοχή του περισσότερες από μία (n/p) γραμμές των πινάκων A και B (και πιο συγκεκριμένα τις γραμμές τάξης $i(n/p)$ έως $(i+1)(n/p)-1$ και ότι συνεπώς να διαμορφώσουμε τα βήματα του αλγορίθμου ως εξής:

1. Κάθε επεξεργασία P_i κατά το βήμα 1 θα κάνει υπολογισμούς για τα στοιχεία n/p γραμμών του πίνακα C , δηλ. των γραμμών τάξης $i(n/p)$ έως $(i+1)(n/p)-1$, με βάση τα στοιχεία των n/p γραμμών των πινάκων A και B που κατέχει.

2. Κάθε επεξεργασία κατά το βήμα 2 θα διακινεί προς το προηγούμενο του επεξεργαστή στην τοπολογία δακτυλίου, το σύνολο των στοιχείων των n/p γραμμών του πίνακα B που κατέχει.

3. Ο αριθμός των επαναλήψεων δε, παραμένει ο ίδιος (p βήματα υπολογισμών και $p-1$ βήματα επιμοιουσίας/αυτοαλλαγής μινουφάτων).

Αναφορικά με τον απαιτούμενο χρόνο ολοκλήρωσης του αλγορίθμου εξετάζοντας απευθείας τη γενική περίπτωση που κάθε επεξεργασία έχει αρχικά στην κατοχή του περισσότερες από μία (n/p) γραμμές των πινάκων A και B , μπορούμε να εξαγάγουμε τα ακόλουθα συμπεράσματα:

1. Σε ότι αφορά το χρόνο που απαιτείται για υπολογισμούς, κάθε επεξεργασία θα εκτελεί σε κάθε βήμα, για κάθε στοιχείο κάθε μιας από τις n/p γραμμές του πίνακα B που μόλις έχει λάβει (για κάθε ένα δηλαδή από τα συνολικά n^2/p στοιχεία του πίνακα αυτού που κατέχει), έναν πολλαπλασιασμό (και μία πρόσθεση για τη σταδιακή αθροίση στο αντίστοιχο κάθε φορά στοιχείο της αντίστοιχης γραμμής του πίνακα C), με ένα στοιχείο από κάθε μια από τις n/p γραμμών του πίνακα A που κατέχει εξ αρχής. Θα συνεπώς δυνάμει σε κάθε βήμα σε κάθε επεξεργαστική συνολικά $(n^2/p)(n/p) = n^3/p^2$ πολλαπλασιασμοί και προσθέσεις. Καθώς δε το σύνολο των βημάτων υπολογισμού είναι p , απαιτείται συνολικά σε κάθε επεξεργαστική χρόνος $O(p(n^3/p^2)) = O(n^3/p)$. Συνεπώς ο υπολογιστικός φόρτος που υπολογίζεται από το αντίστοιχο ασυμπτωτικό πρόβλημα $O(n^3)$ καταμετράται ιδεατά στο σύνολο των επεξεργασιών και επιτυγχάνεται ο βέλτιστος δυνατός παράλληλος χρόνος από n πλευρές υπολογιστών.

2. Σε ότι αφορά τις επιμοιωνίες (φόρτος ανταλλαγής μηνυμάτων) που απαιτούνται από κάθε επεξεργασία, αρκεί και εδώ να παρατηρήσουμε ότι σε κάθε βήμα του αλγορίθμου ειςτός από το τελευταίο, κάθε επεξεργασία στέλνει σε γειτονικό επεξεργαστή τα στοιχεία n/p γραμμών (σύνολο n^2/p στοιχεία). Καθώς δε το σύνολο των βημάτων κατά τα οποία

Λαμβάνει χώρα επιμοιωνία της παραπάνω μορφής είναι $p-1$,
ο συνολικός δαπανούμενος χρόνος επιμοιωνίας για κάθε επεξεργασία
είναι $O(p(n^2/p)) = O(n^2)$. Δηλ. ο χρόνος επιμοιωνίας που
απαιτείται, είναι ανάλογος του μεγέθους των δύο πολλαπλασιαζόμενων
πινάκων

Χρόνος υπολογισμού: $T_{comp} = O(n^3/p)$

Χρόνος επιμοιωνίας: $T_{comm} = O(n^2)$.

Mesh Multiplication

Ο αλγόριθμος αυτός χρησιμοποιεί $m \times k$ επεξεργαστές τοποθετημένους σε 2-D πλέγμα για τον πολλαπλασιασμό ενός πίνακα $A^{m \times n}$ και του $B^{n \times k}$. Οι γραμμές του πλέγματος αριθμούνται από το $1 \dots m$ και οι στήλες του πλέγματος από το $1 \dots k$. Οι πίνακες A και B τροφοδοτούνται στα συσχετισμένους επεξεργαστές στη στήλη 1 και τη γραμμή 1, αντίστοιχα (βλ. σχ. 4(a) για $m=4, n=5, k=3$). Να σημειωθεί ότι η γραμμή i του πίνακα A υστερεί κατά μία χρονική μονάδα πίσω από τη γραμμή $i-1$ για $2 \leq i \leq m$. Όμοια, η στήλη j του πίνακα B υστερεί κατά μία χρονική μονάδα πίσω από τη στήλη $j-1$ για $2 \leq j \leq k$. Αυτό εξασφαλίζει ότι το a_{is} συναντά το b_{sj} στον επεξεργαστή $P(i,j)$ τη σωστή στιγμή.

Στο τέλος του αλγορίθμου, το στοιχείο c_{ij} του παραγόμενου ^(του) πίνακα C βρίσκεται στον επεξεργαστή $P(i,j)$. Αρχικά το $c_{ij}=0$. Στη συνέχεια όταν ο $P(i,j)$ λαμβάνει δύο εισόδους a και b ,

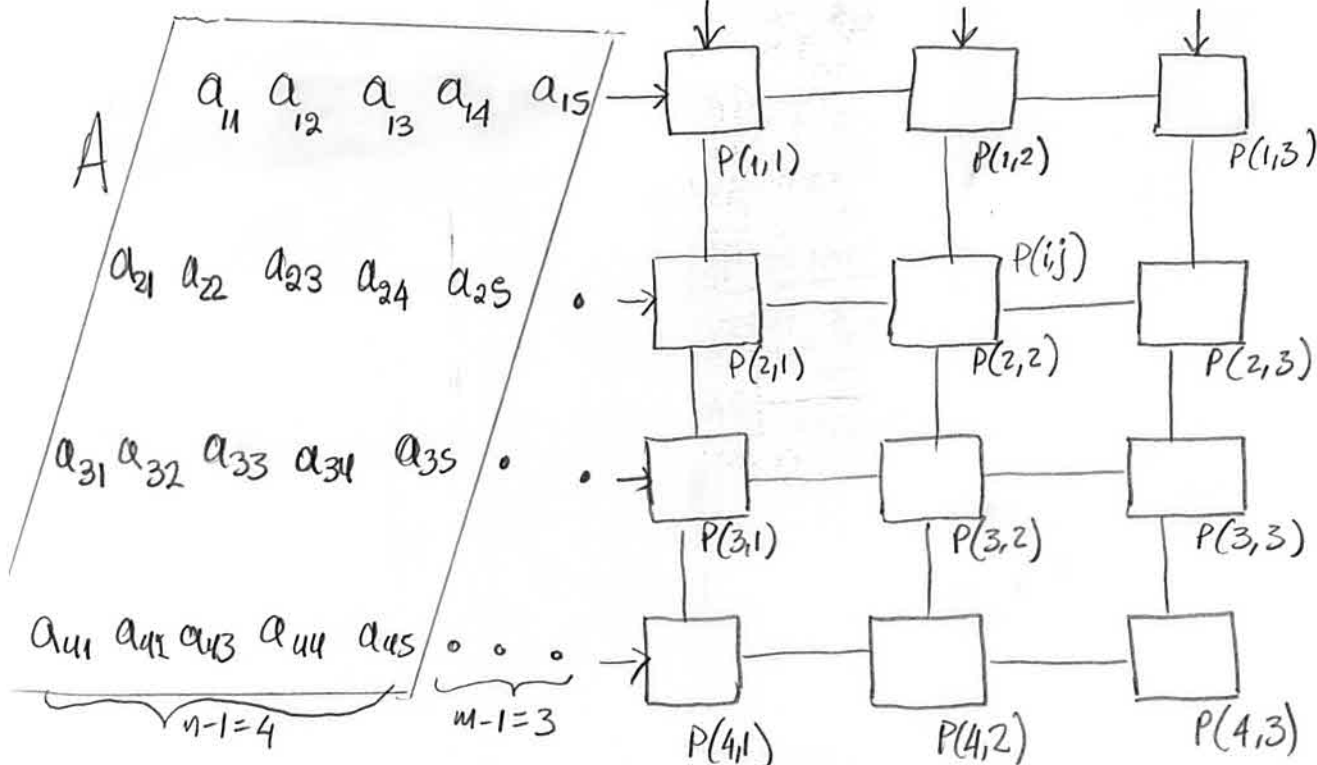
- τις πολλαπλασιάζει
- προσθέτει το αποτέλεσμα στο c_{ij}
- βγάλει το a στο $P(i,j+1)$ εφόσον $j=k$, και
- βγάλει το b στο $P(i+1,j)$ εφόσον του $i=m$.

-13b-

$m=4$ Η τελεία ο σημαίνει
 $n=5$ μαθηματική είσοδος
 $k=3$ κατά 1 χρονική βελτίωση

b_{11}	b_{12}	b_{13}
b_{21}	b_{22}	b_{23}
b_{31}	b_{32}	b_{33}
b_{41}	b_{42}	b_{43}
b_{51}	b_{52}	b_{53}
		$\vdots$

B



Ο αλγόριθμος δίνεται από την ακόλουθη συνάρτηση.

procedure MESH MATRIX MULTIPLICATION (A, B, C)

Για $i=1$ μέχρι m κάνε παράλληλα

Για $j=1$ μέχρι k κάνε παράλληλα

(1) $c_{ij} \leftarrow 0$

(2) Όσο ο $P(i,j)$ λαμβάνει δύο εισόδους a και b , κάνε

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

(ii) αν $i < m$ τότε στείλε το b στον $P(i+1, j)$
 τέλος-αν

(iii) αν $j < k$ τότε στείλε το a στον $P(i, j+1)$
 τέλος-αν

τέλος-όσο

τέλος-για

τέλος-για

Παράδειγμα: MESH MATRIX MULTIPLICATION

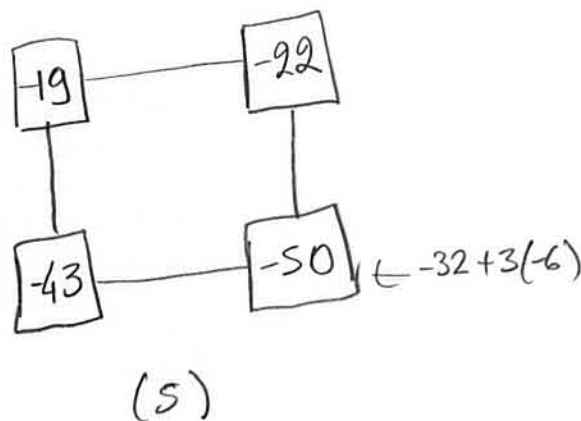
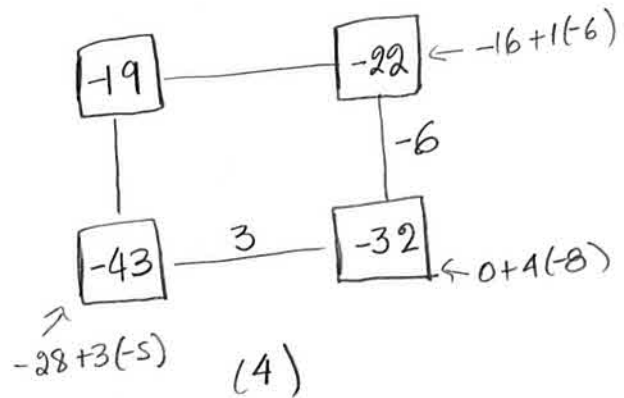
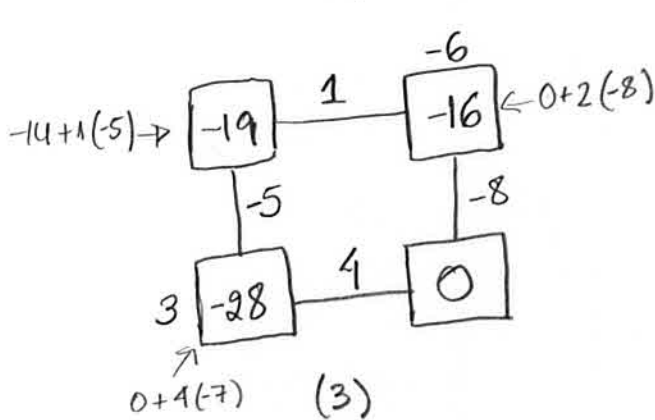
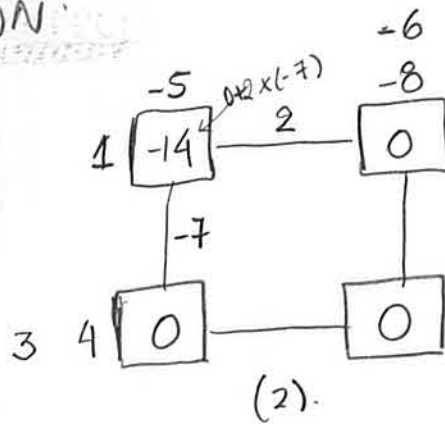
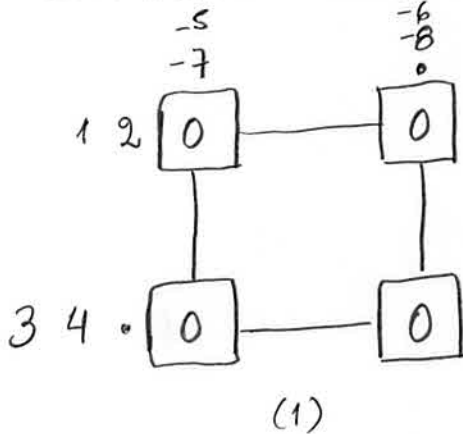
Δίνονται οι πίνακες $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ και $B = \begin{bmatrix} -5 & -6 \\ -7 & -8 \end{bmatrix}$

Η τιμή του a_{ij} μετά από κάθε βήμα φαίνεται μέσα στον $P(i,j)$.

- Αποδοτικότητα πολλαπλασιασμός πινάκων

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

• MESH MATRIX MULTIPLICATION



Ανάλυση: Τα στοιχεία a_{m1} και b_{1k} δέλουν $m+k+n-2$ βήματα από την αρχή των υπολογισμών για να φτάσουν το $P(m,k)$. Αφού ο $P(m,k)$ είναι ο τελευταίος επεξεργαστής που χρησιμοποιείται για τον τερματισμό του αλγορίθμου, απαιτούνται πολλά βήματα για τον υπολογισμό του χυομένου. Υποθέτοντας ότι $m \leq n$ και $k \leq n$ η συνάρτηση $MESH$ MATRIX MULTIPLICATION τρέχει σε χρόνο $t(n) = O(n)$. Αφού $p(n) = O(n^2)$, $c(n) = O(n^3)$, το οποίο ταιριάζει με το χρόνο εκτέλεσης του ακολουδιασμού αλγορίθμου πολλαπλασιασμού πινάκων. Θα πρέπει να σημειωθεί ότι ο χρόνος εκτέλεσης της συνάρτησης $MESH$ MATRIX MULTIPLICATION είναι ο γρηγορότερος δυνατός για τον πολλαπλασιασμό πινάκων σε ηλέγχα επεξεργαστών, θεωρώντας ότι μόνο οι συνοριακοί επεξεργαστές είναι σε θέση να χειρίζονται λειτουργίες εισόδου και εξόδου. Πράγματι, με αυτή την υπόθεση, χρειάζονται $O(n)$ βήματα για να διαβαστούν τα δεδομένα στην είσοδο (από τους επεξεργαστές στη γραμμή 1 και τη στήλη 1) και/ή να παραχθούν τα αποτελέσματα στην έξοδο (από τους επεξεργαστές στη γραμμή m και τη στήλη k)

Πολύς πίνακων σε τοπολογία Διδιάστατου Πλέγματος

• Αλγόριθμος Canny

Ο αλγόριθμος Canny αποτελεί τον πιο γνωστό από τους ειδικότερους αλγορίθμους που αναπτύχθηκαν με σκοπό να εγκαταστήσουν παράλληλα το μεγαλύτερο αριθμό συνδεδεμένων επισημειωσιμότητας που προσφέρει μια τοπολογία διδιάστατου πλέγματος, και να καταλήξει σε παράλληλο χρόνο οργάνωσης αμετά καλότερο από αυτόν που θα επιτυγχανόταν αν εφαρμόζονταν στην τοπολογία διδιάστατου πλέγματος απλά ένας αλγόριθμος δακτυλίου. Εκμεταλλεύεται δε, στη γενική περίπτωση ($p < n^2$) του αναδρομικού αριθμού του πολλαπλασιασμού πίνακων (διαβαση σε υποπίνακες), ο οποίος δίνει τη δυνατότητα για αποδοτική παράλληλη ανταλλαγή μινυφάτων και των δύο διαστάσεων του πλέγματος.

Ας υποθέσουμε αρχικά ότι έχουμε στη διάθεσή μας αριθμό επεξεργασίας ίσο με το πλήθος των στοιχείων του πίνακων που πρόκειται να πολλαπλασιαστούν (δηλ. ένα στοιχείο ανά επεξεργασία: $p = n^2$) οι οποίοι είναι οργανωμένοι σε τοπολογία διδιάστατου πλέγματος διαστάσεων $\sqrt{p} \times \sqrt{p}$ (ή αλλιώς $n \times n$). Για καλύτερη αντιστοίχιση και κατανομή αριθμού της επεξεργασίας με βάση τη θέση τους (επιτεταγμένες) στην τοπολογία του πλέγματος, αντί για την κλασική αρίθμηση $(P_0, \dots, P_{p-1})$ δηλ. ως $P_{i,j}$ ($i=0, \dots, n-1, j=0, \dots, n-1$).

Υποθέτουμε επίσης ότι αρχικά κάθε επεξεργαστής $P_{i,j}$ κατέχει τα στοιχεία a_{ij} και b_{ij} από τους αρχικούς πίνακες A και B και ότι είναι υπεύθυνος για τον υπολογισμό του στοιχείου c_{ij} του πίνακα C . Με βάση τα παραπάνω ο αλγόριθμος του Canny μπορεί να διατυπωθεί ως εξής:

1. Αρχικά τα στοιχεία των δύο πινάκων μετακινούνται χειροκίνητα πάνω στο πλέγμα προκειμένου να βρεθούν τελικά σε κατάλληλες θέσεις για την εφαρμογή της διαδικασίας της κατά σειρά σταδιακή διάχυσης τους που θα ακολουθήσει. Πιο συγκεκριμένα, κάθε στοιχείο γραμμής i του πίνακα A μετακινείται κυλιανά i θέσεις αριστερά, και κάθε ολόκληρη στήλη j του πίνακα B μετακινείται j θέσεις πάνω. Έτσι τελικά, κάθε επεξεργαστής P_{ij} θα καταλήξει να κατέχει τα στοιχεία $a_{i,j+i}$, $b_{i+j,j}$ τα οποία αποτελούν ένα από τα ζεύγη που συσχετίζονται στο άθροισμα υπολογισμού της τελικής τιμής ενός του στοιχείου αποτελέσματος c_{ij} για το οποίο είναι υπεύθυνος ο εν λόγω επεξεργαστής.
2. Κάθε επεξεργαστής πολλαπλασιάζει τα στοιχεία που κατέχει, ξεκινώντας έτσι τον υπολογισμό του αθροίσματος που απαιτείται για το στοιχείο αποτελέσματος c_{ij} για το οποίο είναι υπεύθυνος ($c_{ij} += a_{i,j+i} * b_{i+j,j}$).
3. Κάθε επεξεργαστής βρέλνει το στοιχείο που κατέχει από του πίνακα A μια θέση αριστερά, στον επεξεργαστή $P_{i,j-1}$ (και κυλιανά - δηλ. αν $j=0$ βρέλνει το στοιχείο στον επεξεργαστή $P_{i,n-1}$), και το στοιχείο που κατέχει από του πίνακα B μια θέση πάνω, στον επεξεργαστή $P_{i-1,j}$ (και κυλιανά - δηλ. αν $i=0$ βρέλνει τα στοιχεία στον επεξεργαστή $P_{n-1,j}$).
4. Κάθε επεξεργαστής πολλαπλασιάζει τα στοιχεία που μόλις παρέλαβε από το δεξιό του και τον κάτω του επεξεργαστή - $P_{i,j+1}$ και $P_{i+1,j}$ αντίστοιχα), συσχετίζοντας έτσι αθροιστικά έναν επιπλέον όρο για τον υπολογισμό του αθροίσματος που απαιτείται για το στοιχείο αποτελέσματος c_{ij} για το οποίο είναι υπεύθυνος ($c_{ij} += a_{i,j+i+1} * b_{i+j+1,j}$).

5. Τα ανωτέρω βήματα 3 και 4 επαναλαμβάνονται άλλες $n-2$ φορές (συμβατικά δηλ. $n-1$ επαναλήψεις των βημάτων αυτών - ή αλλιώς $nr-1$ καθώς $n^2=r$), έτσι ώστε να έρθουν στην κατοχή κάθε επεξεργαστή σταδιακά όλα τα ζεύγη στοιχείων των δύο πινάκων, που απαιτούνται για τον υπολογισμό του στοιχείου αποτελέσματος c_{ij} για το οποίο είναι υπεύθυνος.

Κομβικό σημείο του αλγορίθμου αποτελούν οι επιμοιωσίες που συντελούνται κατά το βήμα 3 παραπάνω. Ουσιαστικά σε κάθε επανάληψη, κατά το βήμα αυτό, κάθε ολόκληρη γραμμή τάξης i του πίνακα A μετακινείται κυκλικά μία θέση αριστερά, και κάθε ολόκληρη στήλη τάξης j του πίνακα B μετακινείται μία θέση πάνω. Έτσι ζευγαίει η επιμέου (σταδιακή) κυκλική μετακίνηση των στοιχείων κάθε γραμμής και στήλης έτσι ώστε κάθε επεξεργαστής να αποκτά προδευτικά στην κατοχή του ένα προς ένα και τα υπόλοιπα στοιχεία (πέραν αυτών που ήδη έχει λάβει από την αρχική μετακίνηση που έλαβε χώρα κατά το βήμα 1), τα οποία χρειάζεται για τον πλήρη υπολογισμό του στοιχείου αποτελέσματος c_{ij} για το οποίο είναι υπεύθυνος.

Παράδειγμα Cannon

Αρχικοποίηση του A

A(0,0)	A(0,1)	A(0,2)
A(1,0)	A(1,1)	A(1,2)
A(2,0)	A(2,1)	A(2,2)

Αρχική μετακίνηση γραμμών A

A(0,0)	A(0,1)	A(0,2)
A(1,1)	A(1,2)	A(1,0)
A(2,2)	A(2,0)	A(2,1)

Η 1η γραμμή σταθερή, η 2η γραμμή μετακινείται αυξάνοντας κατά 1 και η 3η γραμμή κατά 2

0 A μετά το 1^ο βήμα

A(0,1)	A(0,2)	A(0,0)
A(1,2)	A(1,0)	A(1,1)
A(2,0)	A(2,1)	A(2,2)

Μετακίνηση αυξάνει των γραμμών κατά 1

0 A μετά το 2^ο βήμα

A(0,2)	A(0,0)	A(0,1)
A(1,0)	A(1,1)	A(1,2)
A(2,1)	A(2,2)	A(2,0)

Μετακίνηση αυξάνει των γραμμών κατά 1

Αρχικοποίηση του B

B(0,0)	B(0,1)	B(0,2)
B(1,0)	B(1,1)	B(1,2)
B(2,0)	B(2,1)	B(2,2)

Αρχική μετακίνηση στηλών B

B(0,0)	B(1,1)	B(2,2)
B(1,0)	B(2,1)	B(0,2)
B(2,0)	B(0,1)	B(1,2)

Η 1η στήλη σταθερή, η 2η στήλη μετακινείται αυξάνοντας κατά 1 και η 3η κατά 2

0 B μετά το 1^ο βήμα

B(1,0)	B(2,1)	B(0,2)
B(2,0)	B(0,1)	B(1,2)
B(0,0)	B(1,1)	B(2,2)

Μετακίνηση αυξάνει των στηλών κατά 1

0 B μετά το 2^ο βήμα

B(2,0)	B(0,1)	B(1,2)
B(0,0)	B(1,1)	B(2,2)
B(1,0)	B(2,1)	B(0,2)

Μετακίνηση αυξάνει των στηλών κατά 1

Πολλαπλασιασμός

Αρχικοποίηση

C(0,0)=0	C(0,1)=0	C(0,2)=0
A(0,0)	A(0,1)	A(0,2)
B(0,0)	B(1,1)	B(2,2)
C(1,0)=0	C(1,1)=0	C(1,2)=0
A(1,1)	A(1,2)	A(1,0)
B(1,0)	B(2,1)	B(0,2)
C(2,0)=0	C(2,1)=0	C(2,2)=0
A(2,2)	A(2,0)	A(2,1)
B(2,0)	B(0,1)	B(1,2)

C(0,0)	C(0,1)	C(0,2)
+=	+=	+=
A(0,0)	A(0,1)	A(0,2)
*B(0,0)	*B(1,1)	*B(2,2)
C(1,0)	C(1,1)	C(1,2)
+=	+=	+=
A(1,1)	A(1,2)	A(1,0)
*B(1,0)	*B(2,1)	*B(0,2)
C(2,0)	C(2,1)	C(2,2)
+=	+=	+=
A(2,2)	A(2,0)	A(2,1)
*B(2,0)	*B(0,1)	*B(1,2)

C(0,0)	C(0,1)	C(0,2)
+=	+=	+=
A(0,1)	A(0,2)	A(0,0)
*B(1,0)	*B(2,1)	*B(0,2)
C(1,0)	C(1,1)	C(1,2)
+=	+=	+=
A(1,2)	A(1,0)	A(1,1)
*B(2,0)	*B(0,1)	*B(1,2)
C(2,0)	C(2,1)	C(2,2)
+=	+=	+=
A(2,0)	A(2,1)	A(2,2)
*B(0,0)	*B(1,1)	*B(2,2)

C(0,0)	C(0,1)	C(0,2)
+=	+=	+=
A(0,2)	A(0,0)	A(0,1)
*B(2,0)	*B(0,1)	*B(1,2)
C(1,0)	C(1,1)	C(1,2)
+=	+=	+=
A(2,0)	A(1,1)	A(1,2)
*B(0,0)	*B(1,1)	*B(2,2)
C(2,0)	C(2,1)	C(2,2)
+=	+=	+=
A(2,1)	A(2,2)	A(2,0)
*B(1,0)	*B(2,1)	*B(0,2)

Πολλαπλασιασμός τα στοιχεία κάθε γραμμής του A με τα αντίστοιχα κάθε στήλης του B και προσθέτω τα προηγούμενα

← Ομοίως

← Ομοίως

Ο παραπάνω αλγόριθμος μπορεί να επεκταθεί κατάλληλα και για την πιο γενική περίπτωση που ο αριθμός των επεξεργαζών είναι μικρότερος από το πλήθος των στοιχείων των πινάκων A και B ($p < n^2$). Αρχικά σε αυτή την περίπτωση να αναμεσάσουμε τον πίνακα του αναδρομικού τρόπου εκφραγής του πολλαπλασιασμού πινάκων με βάση το διαχωρισμό σε υποπίνακες (blocks), θέτοντας $q=p$.

Πιο συγκεκριμένα, αρχικά να χωρίσουμε κάθε έναν από τους πινάκες A και B σε p υποπίνακες διαστάσεων $(n/\sqrt{p}) \times (n/\sqrt{p})$, να αναθέσουμε ένα ζεύγος τέτοιων υποπίνακων σε κάθε επεξεργαστή (τους υποπίνακες A_{ij}, B_{ij} στον επεξεργαστή P_{ij} , $i, j = 0, 1, \dots, \sqrt{p}-1$ και στη συνέχεια να διαμορφώσουμε τα βήματα του αλγορίθμου ως εξής:

1. Κάθε επεξεργαστής P_i κατά το βήμα υπολογισμού κάθε επανάληψης (βήματα 2 και 4 της παραπάνω περιγραφής), θα εκτελεί πολλαπλασιασμό και πρόσθεση υποπίνακων (αυτή για πολλαπλασιασμό και πρόσθεση μεμονωμένων στοιχείων), με βάση τα στοιχεία των δύο blocks των πινάκων A και B που θα έχει στην κατοχή του εφόσον ή θα έχει χυθεί από τους γειτονικούς του (αριστερά και δεξιά) επεξεργαστές κατά την τρέχουσα επανάληψη, ώστε να υπολογίσει διαδοχικά τα στοιχεία του υποπίνακα C_{ij} του πίνακα αποτελέσματος.

2. Κάθε επεξεργαστής κατά το βήμα επιμοιουσιότητας κάθε επανάληψης (βήμα 3 της παραπάνω περιγραφής) θα διαμινεί προς τους γειτονικούς του επεξεργαστές (αριστερά και πάνω) στο πλήρες, ολόκληρο block στοιχείων διαστάσεων $(n/\sqrt{p}) \times (n/\sqrt{p})$ (και για μεμονωμένα στοιχεία), το block του πίνακα A που ανήκει προς τα αριστερά και το block του πίνακα B προς τα πάνω.

3. Ο αριθμός των επαναλήψεων δε, παράγει ο ίδιος ($\sqrt{p}$ βήματα υπολογισμού και $\sqrt{p}-1$ βήματα επιμοιουσιότητας/ανταλλαγής μινυφάτων).

(βλ. προηγ. παραδείγματα)

Αλγόριθμος Γάννηση

% Δίνονται $A = [a_{ij}]^{n \times n}$, $B = [b_{ij}]^{n \times n}$. Να υπολογιστεί το $C = A * B$.

% Χωρίζουμε τους A, B σε p τετραγωνικά blocks $A_{i,j}$ και $B_{i,j}$
($0 \leq i, j < \sqrt{p}$) μεγέθους $(n/\sqrt{p}) \times (n/\sqrt{p})$ το καθένα

% Χρησιμοποιούμε Καρτεσιανή τοπολογία για να φτιάξουμε το πλέγμα
Η διεργασία $P_{i,j}$ αρχικά αποθηκεύει τα $A_{i,j}$ και $B_{i,j}$ και υπολογίζει
το $C_{i,j}$

% Παρατήρηση: Ο υπολογισμός κάθε υποπίνακα $C_{i,j}$ απαιτεί όλους τους
υποπίνακες $A_{i,k}$ και $B_{k,j}$ για $0 \leq k < \sqrt{p}$

% Τύπος υπολογισμού:

$$C_{i,j} = \sum_{k=0}^{\sqrt{p}-1} A_{i,(i+j+k)\% \sqrt{p}} B_{(i+j+k)\% \sqrt{p},j}$$

Αλγόριθμος Γάννηση

1. Αρχική ευθυγράφιση

Για $i, j = 0$ μέχρι $\sqrt{p}-1$ κάνε

Στείλε το block $A_{i,j}$ στη διεργασία $(i, (j-i+\sqrt{p}) \bmod \sqrt{p})$ και το block
 $B_{i,j}$ στη διεργασία $((i-j+\sqrt{p}) \bmod \sqrt{p}, j)$;

Τέλος για

Η διεργασία $P_{i,j}$ πολλαπλασιάζει τους υποπίνακες που έλαβε και προδίδει
το αποτέλεσμα στον $C_{i,j}$

2. Υπολογισμός και μετατόπιση ($C_{i,j} = C_{i,j} + A_{i,k} B_{k,j}$)

Για $step = 1$ ως $\sqrt{p}-1$ κάνε

• Μετακίνησε το $A_{i,j}$ κυκλικά κατά 1 θέση αριστερά και το $B_{i,j}$
κυκλικά κατά 1 θέση πάνω

• Η διεργασία $P_{i,j}$ πολλαπλασιάζει τους υποπίνακες που έλαβε και
προδίδει το αποτέλεσμα στον $C_{i,j}$

Τέλος για

Πομπηλουργία

Αναφορικά με τον απαιτούμενο χρόνο ολολύρωσης του αλγορίθμου, εξετάζοντας απευθείας τη γενική περίπτωση που κάθε επεξεργαστής έχει αρχικά στην κατοχή του περισσότερα από ένα στοιχεία των πινάκων A και B (ένα block στοιχείων διαστάσεων $(n/\sqrt{p}) \times (n/\sqrt{p})$ μπορούμε να εξαγάγουμε τα ακόλουθα συμπεράσματα

1) Σε ότι αφορά το χρόνο που απαιτείται για υπολογισμούς, κάθε επεξεργαστής θα εκτελεί σε κάθε βήμα, ένα πολλαπλασιασμό (και μια πρόσθεση για τη σταδιακή αθροίση στο αντίστοιχο κάθε φορά block του πίνακα C μεταξύ δύο blocks διαστάσεων $(n/\sqrt{p}) \times (n/\sqrt{p})$ των πινάκων A και B). Θα συντελούνται δηλαδή σε κάθε βήμα σε κάθε επεξεργαστή συνολικά $(n/\sqrt{p})^3 = n^3/p^{3/2}$ πολλαπλασιασμοί και προσθέσεις. Καθώς δε το σύνολο των βημάτων υπολογισμού είναι $\sqrt{p}$, απαιτείται συνολικά σε κάθε επεξεργαστή χρόνος $O(p^{1/2}(n^3/p^{3/2})) = O(n^3/p)$. Συνεπώς ο υπολογιστικός φόρτος που υπολογίζεται από το ακολουθιακό πρόγραμμα ($O(n^3)$) κατανέμεται με εδώ ιδεατά στο σύνολο των επεξεργαστών και επιτυγχάνεται ο βέλτιστος δυνατός παράλληλος χρόνος από ημετέρας υπολογιστών

2) Σε ότι αφορά τις επιμοιουσίες (φόρτος και τάξης μηνυμάτων) που απαιτούνται από κάθε επεξεργαστή, αυτές διαχωρίζονται σε δύο κατηγορίες

- Τις επιμοιουσίες που απαιτούνται κατά την αρχική μετακίνηση των υποπινάκων πάνω στο πλέγμα: για τις επιμοιουσίες αυτές θα πρέπει να παρατηρήσουμε αρχικά ότι η μεγαλύτερη σε μήκος μετακίνηση (ώσο σε επίπεδο γραμμής όσο και σε επίπεδο στήλης) είναι κατά $\sqrt{p}-1$ θέσεις. Παρόλο επιπλέον

που περιλαμβάνει διακίνηση όλων των block διαστάσεων $(n/\sqrt{p}) \times (n/\sqrt{p})$ (συνολο n^2/p στοιχείων το καθένα) των επεξεργασιών της συγχρονισμένης γραμμής (ή στύλης) για τους θεσμούς, οι μετακινήσεις αυτές μπορούν να γίνουν παράλληλα σε κάθε γραμμή (ή στύλη) - π.χ. η διεγρή που μετακινείται ένα block από τον P_i στον P_{i+1} μπορεί να μετακινείται παράλληλα και ένα block από τον P_{i-1} στον P_i κ.ο.κ. οπότε συνολικά μπορούν να ολοκληρωθούν όλες οι μετακινήσεις σε $(\sqrt{p}-1)(n^2/p)$ βήματα, δηλ. χρόνο $O(\sqrt{p}(n^2/p)) = O(n^2/\sqrt{p})$

- Τις επιμοιωνίες που απαιτούνται σε κάθε βήμα (επανάληψη) κατά τη διάρκεια εκτέλεσης του αλγορίθμου: για τις επιμοιωνίες αυτές αρκεί να παρατηρήσουμε ότι σε κάθε βήμα του αλγορίθμου εκτός από το πρώτο, κάθε επεξεργασία βάζει παράλληλα σε γειτονικούς επεξεργαστές δύο block διαστάσεων $(n/\sqrt{p}) \times (n/\sqrt{p})$ ένα του πίνακα A και ένα του πίνακα B (συνολο n^2/p στοιχεία το κάθε block). Καθώς δε το συνολο των βημάτων κατά τα οποία λαμβάνει χώρα επιμοιωνία της παραπάνω μορφής είναι $\sqrt{p}-1$, ο συνολικός δαπανώμενος χρόνος επιμοιωνίας για κάθε επεξεργασία είναι $O(\sqrt{p}(n^2/p)) = O(n^2/\sqrt{p})$

Χρόνος υπολογισμού: $T_{comp} = O(n^3/p)$

Χρόνος επιμοιωνίας: $T_{comm} = O(n^2/\sqrt{p})$

Παρατηρούμε ότι $T_{comm}^{απλοχ} = O(n^2/\sqrt{p})$ με τοπολογία 2διάστατου πλέγματος είναι εμφανώς καλύτερος από το χρόνο αλγορίθμου ποσ/μού πίνακων ^{κατά τη} με τοπολογία δακτυλίου $O(n^2)$.

Γενιότερα

Πομπητικότητα σε υπερκύβο αλγορίθμου Caimon

- Η αρχική διευθέτηση των δύο πινακών περιλαμβάνει μια κατά γραμμή και μια κατά στήλη συνδυαστική ολιεθμική. Σε ολιεθμικές αυτές η βέλτιστη απόσταση ολιεθμικής ενός υποπίνακα είναι $\sqrt{p}-1$.
- Σε ένα υπερκύβο (με αποστολή μινιμάτων με διαχωρισμό και κλάδοση) οι δύο λειτουργίες ολιεθμικής απαιτούν συνολικό χρόνο

$$2 \left(t_s + t_w \frac{n^2}{p} + t_h \log \sqrt{p} \right)$$

- Καθε μια από τις $\sqrt{p}$ ολιεθμικές αυτού βηματός στη φάση υπολογισμός και ολιεθμική του αλγορίθμου είναι:

$$T_p^{\text{Caimon}} = 2 \left(t_s + t_w \frac{n^2}{p} \right) \sqrt{p}$$

- Αν το πηλός p είναι αρκετά μεγάλο, τότε ο χρόνος επιμοιωνίας για την αρχική διευθέτηση μπορεί να είναι αμελητέος σε σύγκριση με το χρόνο που απαιτείται για τη φάση υπολογισμού και ολιεθμικής.
- Στον αλγόριθμο αυτόν, ο κάθε επεξεργαστής ετερεί $\sqrt{p}$ πολλαπλασιασμούς $(n/\sqrt{p}) \times (n/\sqrt{p})$ υποπίνακων. Από γίνεται σε υπολογιστικό χρόνο

$$T_p^{\text{comp}} = \sqrt{p} \times (n/\sqrt{p})^2 = n^3/p$$

- Γνωμένως ο συνολικός χρόνος της παραλληλως διάτρεξης είναι

$$T_p^{\text{total}} = \frac{n^3}{p} + 2\sqrt{p} t_s + 2t_w \frac{n^2}{\sqrt{p}}$$

- Αρα, το κόστος του παραλληλου αλγορίθμου Caimon είναι το ίδιο με εκείνο του αυτού παραλληλου αλγορίθμου.

Πολυπλοκότητα αλγορίθμου Canny σε ηλέγχα

Για την υλοποίηση του αλγορίθμου Canny χρησιμοποιούμε ένα αναδιηλούμενο ηλέγχα $\sqrt{p}$, ώστε ο χρόνος που απαιτείται για την αρχική διεύθυνση των δύο πινακων είναι

$$2(t_s + t_w \cdot \frac{n^2}{p})\sqrt{p}$$

Ο ίδιος επιπλο χρόνος απαιτείται και στη φάση υπολογισμού και ολίσθησης του αλγορίθμου

Επομένως ο συνολικός χρόνος παράλληλης εκτέλεσης του αλγορίθμου Canny σε τετραγωνικό ηλέγχα είναι

$$T_p^{mesh} = \frac{n^3}{p} + 4\sqrt{p}t_s + 4t_w \frac{n^2}{\sqrt{p}}$$

άρα, αυτός είναι αποτελεσματικότερος αλγόριθμος σε σχέση με τον αλγόριθμο σε υπερκύβο.

MPI-Code Cannon -24-

```
int MPI_Sendrecv_replace(void *buf, int count, MPI_Datatype datatype,  
    int dest, int sendtag, int source, int recvtag, MPI_Comm comm,  
    MPI_Status *status);
```

- Execute a blocking send and receive. The same buffer is used both for the send and for the receive, so that the message sent is replaced by the message received.
- buf [in/out]: initial address of send and receive buffer

```
#include "mpi.h"
```

```
#include <stdio.h>
```

```
int main (int argc, char*argv[])
```

```
{  
    int myid, numprocs, left, right;
```

```
    int buffer[10];
```

```
    MPI_Request request;
```

```
    MPI_Status status;
```

```
    MPI_Init (&argc, &argv);
```

```
    MPI_Comm_size (MPI_COMM_WORLD, &numprocs);
```

```
    MPI_Comm_rank (MPI_COMM_WORLD, &myid);
```

```
    right = (myid + 1) % numprocs;
```

```
    left = myid - 1;
```

```
    if (left < 0)
```

```
        left = numprocs - 1;
```

```
    MPI_Sendrecv_replace (buffer, 10, MPI_INT, left, 123, MPI_COMM_WORLD, &status,
```

```
    MPI_Finalize ( );
```

```
    return 0;
```

Αλγόριθμος Fox.

Ο αλγόριθμος του Fox αποτελεί μια εναλλακτική μέθοδο για του πολλαπλασιασμό πυαίων σε τοπολογία διεδράδατος ηλέγματος, αρμετά παρόμοια με αυτή του Cannon που παρουσιάστηκε στην προηγούμενη παράγραφο. Η κύρια διαφοροποίηση του σε σχέση με του αλγόριθμο του Cannon είναι ότι δεν επιτελεί αρχικά κάποια μαζική προετοιμασία (μετακίνηση) των στοιχείων των δύο πυαίων στο ηλέγμα, αλλά αντίθετα προβαθεί να κατανείμει το σύνολο των απαιτούμενων επικοινωνιών κατά τη διάρκεια των βημάτων εκτέλεσης του αλγόριθμου.

Για καλύτερη αντιμετώπιση και κατανόηση, αριθμούμε τους επεξεργαστές και εδώ με βάση τη θέση τους (συμμεταγμένες) στην τοπολογία του ηλέγματος, δηλ. ως $P_{i,j}$ ($i=0, \dots, n-1$, $j=0, \dots, n-1$). Υποθέτουμε επίσης και εδώ ότι αρχικά κάθε επεξεργαστής $P_{i,j}$ κατέχει τα στοιχεία $a_{i,j}$ και $b_{i,j}$ από τους αρχικούς πυαίους A και B και ότι είναι υπεύθυνος για του υπολογισμό του στοιχείου $c_{i,j}$. Με βάση τα παραπάνω ο αλγόριθμος του Fox μπορεί να διατυπωθεί ως εξής:

1. Επιλέγεται ένα στοιχείο $a_{i,j}$ του πυαίου A από κάθε γραμμή του ηλέγματος. Πιο συγκεκριμένα, αρχικά επιλέγεται το στοιχείο $a_{0,0}$ από την 1η γραμμή, το στοιχείο $a_{1,0}$ από τη 2η γραμμή κ.ο.κ., δηλ. τα στοιχεία $a_{i,0}$ ($i=0, \dots, n-1$).

2. Ο επεξεργαστής υάδε γραμμής που κατέχει το στοιχείο του πίνακα A που επεξεργάστηκε στο βήμα 1 (αρχικά δηλ. οι επεξεργαστές $P_{i,i}$ ($i=0, \dots, n-1$), στέλνει το στοιχείο αυτό ($a_{i,i}$) σε όλους τους επεξεργαστές της συγκεκριμένης γραμμής.
3. Κάθε επεξεργαστής πολλαπλασιάζει το στοιχείο του πίνακα A που μόλις παρέλαβε με το στοιχείο του πίνακα B που κατέχει, ξεκινώντας έτσι τον υπολογισμό του αθροίσματος που απαιτείται για το στοιχείο αποτελέσματος c_{ij} για το οποίο είναι υπεύθυνος ($c_{ij} += a_{i,j} * b_{i+j,j}$).
4. Κάθε επεξεργαστής στέλνει το στοιχείο που κατέχει από τον πίνακα B μια θέση πάνω, δηλ. στον επεξεργαστή $P_{i-1,j}$ (και φυσικά - δηλ. αν $i=0$ στέλνει το στοιχείο στον επεξεργαστή $P_{n-1,j}$).
5. Τα ανωτέρω βήματα 1 έως 4 επαναλαμβάνονται άλλες $n-1$ φορές (βασισμένα δηλ. n επαναλήψεις, εκ των οποίων στην τελευταία δεν είναι απαραίτητο να εκτελεστεί το βήμα 4), επιλέγοντας σε υάδε επανάληψη του βήματος 1 από υάδε γραμμή για αποστολή, το αμέσως επόμενο σε σειρά στοιχείο του πίνακα A . Πιο συγκεκριμένα, σε υάδε βήμα k ($k=0, \dots, n-1$) του αλγορίθμου πρέπει να επιλέγεται από υάδε γραμμή του πίνακα A το στοιχείο $a_{i,(i+k)\%n}$.

Κομβικό σημείο του αλγορίθμου αποτελούν και εδώ οι επικοινωνίες που συντελούνται κατά το βήμα 4 της περιγραφής. Ουσιαστικά, σε υάδε επανάληψη, κατά το βήμα αυτό, υάδε ολόκληρη στήλη τζης j του πίνακα B μετακινείται μια θέση πάνω. Έτσι ξεκινάει η (βαθιαύη) κυλιόμενη μετακίνηση των στοιχείων υάδε στήλης του πίνακα B , ώστε υάδε επεξεργαστής

να αποκτήσει προτεραιότητα στην κατοχή του ένα-προς-ένα και τα υπολοίπα στοιχεία του πίνακα B που χρειάζεται (μαζί με τα υπόλοιπα στοιχεία του πίνακα A τα οποία θα λαμβάνει επίσης σταδιακά μέσω των επικοινωνιών που συντελούνται κατά το βήμα 1) για τον πλήρη υπολογισμό του στοιχείου αποτελέσματος C_{ij} για το οποίο είναι υπεύθυνος.

Αλγόριθμος Fox.

/* my process row = i, my process column = j */

$q = \text{sqrt}(p);$

$\text{dest} = ((i-1) \bmod q, j);$

for (stage = 0; stage < q; stage++)

{
 $k\text{-bar} = (i + \text{stage}) \bmod q;$

 (a) Broadcast $A[i, k\text{-bar}]$ across process row i;

 (b) $C[i, j] = C[i, j] + A[i, k\text{-bar}] * B[k\text{-bar}, j];$

 (c) Send $B[(k\text{-bar} + 1) \bmod q, j]$ to dest;

 Receive $B[(k\text{-bar} + 1) \bmod q, j]$ from source;

}

Παράδειγμα Fox

Αρχικοποίηση του A

A(0,0)	A(0,1)	A(0,2)
A(1,0)	A(1,1)	A(1,2)
A(2,0)	A(2,1)	A(2,2)

A(0,0)	A(0,1)	A(0,2)
A(1,0)	A(1,1)	A(1,2)
A(2,0)	A(2,1)	A(2,2)

Επιλογή των διαγώνιων στοιχείων του A

A(0,1)	A(0,2)	A(0,0)
A(1,1)	A(1,2)	A(1,0)
A(2,1)	A(2,2)	A(2,0)

Μετακίνηση υπολοίπων γραμμών κατά 1 και επιλογή των διαγώνιων στοιχείων

A(0,2)	A(0,0)	A(0,1)
A(1,2)	A(1,0)	A(1,1)
A(2,2)	A(2,0)	A(2,1)

Μετακίνηση υπολοίπων γραμμών κατά 1 και επιλογή των διαγώνιων στοιχείων

Αρχικοποίηση του B

B(0,0)	B(0,1)	B(0,2)
B(1,0)	B(1,1)	B(1,2)
B(2,0)	B(2,1)	B(2,2)

B(0,0)	B(0,1)	B(0,2)
B(1,0)	B(1,1)	B(1,2)
B(2,0)	B(2,1)	B(2,2)

Καμία μεταβολή στο B.

B(1,0)	B(1,1)	B(1,2)
B(2,0)	B(2,1)	B(2,2)
B(0,0)	B(0,1)	B(0,2)

Μετακίνηση υπολοίπων γραμμών κατά 1

B(2,0)	B(2,1)	B(2,2)
B(0,0)	B(0,1)	B(0,2)
B(1,0)	B(1,1)	B(1,2)

Μετακίνηση υπολοίπων γραμμών κατά 1

Πολλαπλασιασμός

Αρχικοποίηση

C(0,0)=0	C(0,1)=0	C(0,2)=0
A(0,0)	A(0,1)	A(0,2)
B(0,0)	B(0,1)	B(0,2)
C(1,0)=0	C(1,1)=0	C(1,2)=0
A(1,0)	A(1,1)	A(1,2)
B(1,0)	B(1,1)	B(1,2)
C(2,0)=0	C(2,1)=0	C(2,2)=0
A(2,0)	A(2,1)	A(2,2)
B(2,0)	B(2,1)	B(2,2)

C(0,0)	C(0,1)	C(0,2)
+ =	+ =	+ =
A(0,0)	A(0,1)	A(0,2)
* B(0,0)	* B(0,1)	* B(0,2)
C(1,0)	C(1,1)	C(1,2)
+ =	+ =	+ =
A(1,0)	A(1,1)	A(1,2)
* B(1,0)	* B(1,1)	* B(1,2)
C(2,0)	C(2,1)	C(2,2)
+ =	+ =	+ =
A(2,0)	A(2,1)	A(2,2)
* B(2,0)	* B(2,1)	* B(2,2)

Πολλαπλασιάζω τα διαγώνια στοιχεία κάθε γραμμής του A με την αντίστοιχη γραμμή του B και προσθέτω τα προκύψοντα

C(0,0)	C(0,1)	C(0,2)
+ =	+ =	+ =
A(0,1)	A(0,1)	A(0,1)
* B(1,0)	* B(1,1)	* B(1,2)
C(1,0)	C(1,1)	C(1,2)
+ =	+ =	+ =
A(1,2)	A(1,2)	A(1,2)
* B(2,0)	* B(2,1)	* B(2,2)
C(2,0)	C(2,1)	C(2,2)
+ =	+ =	+ =
A(2,0)	A(2,0)	A(2,0)
* B(0,0)	* B(0,1)	* B(0,2)

Όμοια με

C(0,0)	C(0,1)	C(0,2)
+ =	+ =	+ =
A(0,2)	A(0,2)	A(0,2)
* B(2,0)	* B(2,1)	* B(2,2)
C(1,0)	C(1,1)	C(1,2)
+ =	+ =	+ =
A(1,0)	A(1,0)	A(1,0)
* B(0,0)	* B(0,1)	* B(0,2)
C(2,0)	C(2,1)	C(2,2)
+ =	+ =	+ =
A(2,1)	A(2,1)	A(2,1)
* B(1,0)	* B(1,1)	* B(1,2)

Όμοια με

Αριθμητικό Παράδειγμα

Έστω ότι έχουμε να πολλαπλασιάσουμε τον 3x3 block matrix

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 2 \\ 2 & 0 & 3 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 6 & 2 & 9 \\ 4 & 4 & 5 \\ 4 & 2 & 6 \end{bmatrix}$$

Stage 0:

Process (i, i mod 3)	Broadcast along row 1
(0,0)	a_{00}
(1,1)	a_{11}
(2,2)	a_{22}

a_{00}, b_{00}	a_{00}, b_{01}	a_{00}, b_{02}
a_{11}, b_{10}	a_{11}, b_{11}	a_{11}, b_{12}
a_{22}, b_{20}	a_{22}, b_{21}	a_{22}, b_{22}

Process (i,j) computes

$C_{00} = 1 \times 1 = 1$	$C_{01} = 1 \times 0 = 0$	$C_{02} = 1 \times 2 = 2$
$C_{10} = 1 \times 2 = 2$	$C_{11} = 1 \times 0 = 0$	$C_{12} = 1 \times 3 = 3$
$C_{20} = 1 \times 1 = 1$	$C_{21} = 1 \times 2 = 2$	$C_{22} = 1 \times 1 = 1$

Shift-rotate on the columns of B

Stage 1:

Process ($i, (i+1) \bmod 3$)	Broadcast along row i
(0, 1)	a_{01}
(1, 2)	a_{12}
(2, 0)	a_{20}

$$\begin{array}{ccc}
 a_{01}, b_{10} & a_{01}, b_{11} & a_{01}, b_{12} \\
 a_{12}, b_{20} & a_{12}, b_{21} & a_{12}, b_{22} \\
 a_{20}, b_{00} & a_{20}, b_{01} & a_{20}, b_{02}
 \end{array}$$

Process (i, j) computes:

$c_{00} = 1 + (2 \times 2) = 5$	$c_{01} = 0 + (2 \times 0) = 0$	$c_{02} = 2 + (2 \times 3) = 8$
$c_{10} = 2 + (2 \times 1) = 4$	$c_{11} = 0 + (2 \times 2) = 4$	$c_{12} = 3 + (2 \times 1) = 5$
$c_{20} = 1 + (1 \times 1) = 2$	$c_{21} = 2 + (1 \times 0) = 2$	$c_{22} = 1 + (1 \times 2) = 3$

Shift-rotate on the columns of B.

Stage 2:

Process $(i, (i+2) \bmod 3)$	Broadcast along row i
$(0, 2)$	a_{02}
$(1, 0)$	a_{10}
$(2, 1)$	a_{21}

a_{02}, b_{20} a_{02}, b_{21} a_{02}, b_{22}
 a_{10}, b_{00} a_{10}, b_{01} a_{10}, b_{02}
 a_{21}, b_{10} a_{21}, b_{11} a_{21}, b_{12}

Process (i, j) computes:

$c_{00} = 5 + (1 \times 1) = 6$	$c_{01} = 0 + (1 \times 2) = 2$	$c_{02} = 8 + (1 \times 1) = 9$
$c_{10} = 4 + (0 \times 1) = 4$	$c_{11} = 4 + (0 \times 0) = 4$	$c_{12} = 5 + (0 \times 2) = 5$
$c_{20} = 2 + (1 \times 2) = 4$	$c_{21} = 2 + (1 \times 0) = 2$	$c_{22} = 3 + (1 \times 3) = 6$

Ο παραπάνω αλγόριθμος μπορεί να επεκταθεί μακρύτερα και για την πιο ρεαλιστική περίπτωση που ο αριθμός των επεξεργασιών είναι μικρότερος από το πλήθος των στοιχείων των πινάκων A και B ($p < n^2$), κατά παρόμοιο τρόπο όπως και ο αλγόριθμος του Canny. Αρχικά δηλαδή και εδώ να χωρίσουμε κάθε έναν από τους πίνακες A και B σε p υποπίνακες διαστάσεων $(n/\sqrt{p}) \times (n/\sqrt{p})$, να αναθέσουμε ένα ζεύγος τέτοιων υποπίνακων σε κάθε επεξεργασία, και να διαφορφώσουμε τα βήματα του αλγορίθμου έτσι ώστε κάθε επεξεργασία:

- σε κάθε βήμα υπολογισμού (βήμα 3 της παραπάνω περιγραφής) να εκτελέσει πολλαπλασιασμό και πρόσθεση υποπίνακων, αντί για πολλαπλασιασμό και πρόσθεση μεμονωμένων στοιχείων, και

- σε κάθε βήμα επιμοιωνυίας (βήματα 2 και 4 της παραπάνω περιγραφής) να διακινεί προς τους υπόλοιπους επεξεργαστές, ολόκληρα blocks στοιχείων διαστάσεων $(n/\sqrt{p}) \times (n/\sqrt{p})$, αντί για μεμονωμένα στοιχεία.

Αναφορικά με τον απαιτούμενο χρόνο ολοκλήρωσης του αλγορίθμου, εξετάζοντας και εδώ απευθείας τη γενική περίπτωση που κάθε επεξεργασία έχει αρχικά στην κατοχή του περισσότερα από ένα στοιχεία των πινάκων A και B (ένα block διαστάσεων $(n/\sqrt{p}) \times (n/\sqrt{p})$), παρατηρούμε τα ακόλουθα:

• Σε ότι αφορά το χρόνο που απαιτείται για υπολογισμούς, εκτελούνται όπως και στον αλγόριθμο του Camion συνολικά $\sqrt{p}$ αντίστοιχα βήματα, σε κάθε ένα από τα οποία πραγματοποιείται σε κάθε επεξεργαστή πολλαπλασιασμός και πρόσθεση μεταξύ δύο blocks διαστάσεων $(n/\sqrt{p}) \times (n/\sqrt{p})$. Έτσι μπορούμε να καταλήξουμε κατά παρόμοιο τρόπο στο συμπέρασμα ότι απαιτείται συνολικά σε κάθε επεξεργαστή χρόνος $O(n^3/p)$. Συνεπώς και εδώ ο υπολογιστικός φόρτος που υπολογίζεται από το ακολουθιακό πρόβλημα ($O(n^3)$) καταμετράται πώς αναμενόταν ιδεατά στο σύνολο των επεξεργαστών και πετυγχάνεται ο βέλτιστος δυνατός παράλληλος χρόνος από λευράς υπολογιστών.

Σε ότι αφορά τις επιμοιωμένες (φόρτος ανταλλαγής μηνυμάτων) που απαιτούνται διακρίνουμε μεταξύ των αναγκών που υπολογίζονται από τα βήματα 2 και 4 της περιγραφής του αλγόριθμου ως εξής:

- Για τις επιμοιωμένες οι οποίες υπολογίζονται από το βήμα 2 (αποστολή σε όλους του επεξεργαστές κάθε γραμμής ενός block του πίνακα A διαστάσεων $(n/\sqrt{p}) \times (n/\sqrt{p})$, από έναν επεξεργαστή της γραμμής) απαιτείται σε κάθε επανάληψη χρόνος $O((\sqrt{p})(n/\sqrt{p})(n/\sqrt{p})) = O((\sqrt{p})(n^2/p))$, καθώς το αντίστοιχο μήνυμα μεγέθους n^2/p στοιχείων θα πρέπει να διανύσει απόσταση της τάξεως της μισής διάστασης του πλέγματος ($\sqrt{p}/2$) προκειμένου

να φτάει σε όλους τους επεξεργαστές της γραμμής. Καθώς δε το εύρος των βυθιάτων κατά τα οποία λαμβάνει χώρα επιμοίωση της παραπάνω μορφής είναι $\sqrt{p}$, ο συνολικός δαπανώμενος χρόνος επιμοίωσης για κάθε επεξεργαστή είναι $O(\sqrt{p} \sqrt{p} (n^3/p)) = O(n^2)$

- Οι επιμοιώσεις αντίθετα οι οποίες υπαγορεύονται από το βήμα 4 (αποστολή από κάθε επεξεργαστή του block του πίνακα B που κατέχει μία θέση πύου), είναι ίδιας μορφής και φόρτου με τις επιμοιώσεις που υπαγορεύει και ο αλγόριθμος του Cannon κατά τη διάρκεια των επαναλήψεων του (μετακίνηση των blocks των πίνακων A και B μία θέση αριστερά και πάνω αντίστοιχα). Καθώς δε, ο συνολικός αριθμός των ^{επαναλήψεων} βυθιάτων (επαναλήψεων) επιμοιωσιών είναι ο ίδιος $(\sqrt{p}-1)$, μπορούμε διεκδικούμενοι κατά παρόμοιο τρόπο να καταλήξουμε στο συμπέρασμα ότι για τις n λόγω επιμοιωσιών απαιτείται συνολικά χρόνος $O(n^2/\sqrt{p})$

$$\text{Χρόνος υπολογισμού: } T_{comp} = q \left(\frac{n}{q} \times \frac{n}{q} \times \frac{n}{q} \right) = \frac{n^3}{q^2} = \frac{n^3}{p}$$

(όπου $p = q^2$ πλήθος επεξεργαστών σε ένα $q \times q$ πλέγμα)

$$\text{Χρόνος επιμοιωσιών: } T_{comm} = O(n^2/\sqrt{p}) + O(n^2) = O(n^2)$$

Συνεπώς, η επιτάχυνση και η αποδοτικότητα είναι:

$$S_p = \frac{n^3/p}{n^3/p} = p \quad \text{και} \quad E_p = \frac{n^3}{p(n^3/p)} = 1.$$

Όπως προκύπτει από τα παραπάνω, ο συνολικός χρόνος εκτελέσεως ο οποίος επιτυγχάνεται από τον αλγόριθμο του Fox είναι θεωρητικά χειρότερος από τον αντίστοιχο του αλγόριθμου του Camion ($O(n^2)$ έναντι $O(n^2/\sqrt{p})$) και ισοδύναμος σε τάξη πολυπλοκότητας με αυτόν του αλγόριθμου πολλαπλασιασμού κατά γραμμές σε τοπολογία δακτυλίου. Η παραπάνω διαφορά προκύπτει ουσιαστικά από την ανάγκη αποστολής σε όλους σε ιάδε βύβα, ενός μηνύματος σε ιάδε γραμμή του πλέγματος.

Έχει επίσηςδειχθεί πειραματικά ότι αν τα βύβα εκτέλεσης του αλγόριθμου του Fox δεν εκτελεστούν συγχρονισμένα (επιτρέπει δηλαδή στους επεξεργαστές να αρχίσουν τους υπολογισμούς τους αμέσως μόλις παραλάβουν τα δεδομένα που απαιτούνται σε ιάδε βύβα, χωρίς να περιφέρουν την οργάνωση ολογωγού του κύκλου επικοινωνίας) επιτυγχάνει χρόνους πρακτικά ισοδύναμους με αυτούς του αλγόριθμου του Camion.

Ένα σχετικό πλεονέκτημα του αλγόριθμου του Fox έναντι αυτού του Camion είναι ότι με το πέρας του αλγόριθμου η κατανομή των στοιχείων των δύο πινάκων (Α και Β) είναι επεξεργαστές παραμένει η ίδια όπως ήταν στην αρχή, κάτι το οποίο είναι αρκετά σημαντικό όταν η όλη πράξη του πολλαπλασιασμού αποτελεί ενδιαφέρον πράξη ως μέρος συνολικότερων υπολογισμών. Θυμάμαι ένα σχετικό

αυτά το ίδιο μέγεθος του αλγορίθμου του Fox είναι ότι εξ' ορισμού απαιτεί μεγαλύτερο χώρο μνήμης σε κάθε επεξεργασία από αυτόν που απαιτεί ο αλγόριθμος του Cannon (το χώρο που απαιτείται για την αποθήκευση του block του πινάκα Α που αποστέλλεται προς όλους τους επεξεργαστές είναι γραμμικό σε κάθε βήμα - δηλ. περίπου κατά $\frac{1}{3}$ παραπάνω πρωτεύοντα χώρο αποθήκευσης, αν θεωρήσουμε ότι οι υπόλοιποι απαιτήσεις σε πρωτεύοντα χώρο αποθήκευσης σε κάθε επεξεργασία αφορούν στο αρχικό block του πινάκα Α, στο χρησιμοποιώμενο σε κάθε βήμα block του πινάκα Β και στο block του πινάκα αποτελέσματος C για το οποίο είναι υπεύθυνοι).

Υπερκύβος

Μετά από $\sqrt{p}$ βήματα, ο επεξεργαστής P_{ij} έχει εκτελέσει τον πολλαπλασιασμό $A_{ik} \times B_{kj}$, για κάθε $k = 0, \dots, \sqrt{p} - 1$. Κάθε επεξεργαστής ζοδεύει συνολικά $\frac{n^3}{p}$ χρόνο εκτέλεσης των υπολογισμών. Ο χρόνος για το ένα-προς-όλους broadcast των blocks του Α κυριαρχεί στο χρόνο για μετατοπίσεις ενός βήματος των blocks του Β. Αν το λογικό πλήθος επεξεργαστών είναι εμφωλευμένο στον υπερκύβο, τότε κάθε λειτουργία broadcast μπορεί να εκτελεστεί σε χρόνο $(t_s + t_w \frac{n^2}{p}) \log \sqrt{p}$. Αφού αυτή η λειτουργία επαναλαμβάνεται $\sqrt{p}$ φορές, ο συνολικός

- 37 -

χρόνος επιμοιουσιότητας είναι $\frac{1}{2}(t_s \sqrt{p} + t_w \frac{n^2}{\sqrt{p}}) \log p$

και ο παράλληλος χρόνος είναι:

$$T_p = \frac{n^3}{p} + \frac{1}{2} t_s \sqrt{p} \log p + t_w \frac{n^2}{\sqrt{p}} \log p.$$

Η συνάρτηση ισοαποδοτικότητας του αλγορίθμου Fox ε'εαν υπερκύβο είναι $\Theta(p^{3/2} \log p)$ λόγω του t_s και $\Theta(p^{3/2} (\log p)^3)$ λόγω του t_w . Αν και ο παράλληλος χρόνος και η ιλιμακωγή του αλγορίθμου είναι καλύτερα και του απλού αλγορίθμου, αλλά και του Cannon αλγορίθμου, ο χρόνος εκτέλεσης του αλγορίθμου μπορεί να βελτιωθεί με διαστρωμένους υπολογιστές και την τεχνική της σωλήνωσης.

Αλγόριθμος DNS (Deke, Nassimi, Sahni [1981]) για τον πολλαπλασιασμό δύο $n \times n$ πινάκων

Οι αλγόριθμοι πολλαπλασιασμού πινάκων που παρουσιάστηκαν μέχρι τώρα χρησιμοποιούν το διαχωρισμό των πινάκων σε τετραγωνα block (βασίλειος) και χρησιμοποιούν ένα μεγάλο αριθμό επεξεργασιών n^2 για $n \times n$ πίνακες. Σαν αποτέλεσμα, αυτοί οι αλγόριθμοι έχουν παράλληλο χρόνο εκτέλεσης $\Omega(n)$ καθώς υπάρχουν $\Theta(n^3)$ πράξεις στον απολογιστικό αλγόριθμο.

Θα παρουσιάσουμε τώρα ένα παράλληλο αλγόριθμο που μπορεί να χρησιμοποιήσει n^2 επεξεργαστές και εκτελεί τον πολλαπλασιασμό πινάκων σε $\Theta(\log n)$ χρόνο χρησιμοποιώντας $\Omega(n^3 / \log n)$ επεξεργαστές.

Ο αλγόριθμος αυτός λέγεται DNS.

Θα περιγράψουμε αρχικά τον αλγόριθμο για ένα CREW PRAM, χωρίς να λαμβάνει υπόψη κάποιο συγκεκριμένο δίκτυο διασύνδεσης. Υποθέτουμε ότι n^3 επεξεργαστές είναι διαθέσιμοι για τον πολλαπλασιασμό δύο $n \times n$ πινάκων. Οι επεξεργαστές αυτοί ταυτοποιούνται με ένα τριδιάστατο $n \times n \times n$ λογικό πίνακα. ^{ο οποίος απεικονίζεται με έναν 3-D υπερκύβο.} Αφού ο αλγόριθμος πολλαπλασιασμού πινάκων εκτελεί n^3 βαθμωτούς πολλαπλασιασμούς, με κάθε έναν από τους n^3 επεξεργαστές ανατίθεται μόνο ένας βαθμωτός πολλαπλασιασμός. Οι επεξεργαστές ονομάζονται σύμφωνα με τη θέση τους στον πίνακα, και ο πολλαπλασιασμός $A[i, k] \times B[k, j]$ εκχωρείται στον επεξεργαστή $P_{i,j,k}$ ($0 \leq i, j, k < n$). Αφού κάθε επεξεργαστής εκτελεί έναν απλό πολλαπλασιασμό, τα περιεχόμενα του $P_{i,j,0}, P_{i,j,1}, \dots, P_{i,j,n-1}$

προβιβάζονται για να πάρουμε το $C[i,j]$. Οι προαδέσεις για όλα τα $C[i,j]$ μπορούν να γίνουν ταυτόχρονα σε $\log n$ βήματα. Επομένως, χρειάζεται ένα βήμα για τον πολλαπλασιασμό και $\log n$ βήματα για την πρόσθεση, δηλ. θέλει $\Theta(\log n)$ χρόνο για τον πολλαπλασιασμό δύο $n \times n$ πινάκων σε ένα CREW PRAM μοντέλο.

Στη συνέχεια θα περιγράψουμε την παράλληλη υλοποίηση του πολλαπλασιασμού πινάκων κάνοντας χρήση ενός υπερκύβου συνδεδεμένου παράλληλου υπολογιστή. Υποθέτουμε ότι ο λογισμός τριών διαστάσεων πίνακα απεικονίζεται σε ένα υπερκύβο με $n^3 = 2^{3d}$ επεξεργαστές

Όπως φαίνεται στο Σχήμα 1, η διάταξη του επεξεργαστή μπορεί να απεικονιστεί ως η επίπεδα $n \times n$ επεξεργαστών το καθένα. Κάθε επίπεδο αντιστοιχεί σε μια διαφορετική τιμή του k . Αρχικά, όπως φαίνεται στο 2x1α, οι πίνακες είναι κατανεμημένοι μεταξύ των n^2 επεξεργαστών του επιπέδου που αντιστοιχεί στο $k=0$ στη βάση του τριών διαστάσεων πίνακα επεξεργαστών. Ο επεξεργαστής $P_{i,j,0}$ αρχικά αποθηκεύει τα $A[i,j]$ και $B[i,j]$.

Στον CREW PRAM αλγόριθμο, εννοιολογικά, η κιάτση βέλγη των επεξεργαστών $P_{i,j,*}$ υπολογίζει το εσωτερικό γινόμενο της γραμμής $A[i,*]$ και της στήλης $B[* ,j]$. Δεν είναι απαραίτητη κανένα βήμα μετακίνησης δεδομένων γιατί σε ένα CREW PRAM μοντέλο, κάθε επεξεργαστής μπορεί να προσηλάσει την οποιαδήποτε θέση σε σταθερό χρόνο. Όμως, σε ένα υπερκύβο, οι γραμμές του A

και οι στύλοι του B χρειάζεται να μετακινούνται κατάλληλα ώστε κάθε κάθετη στύλη επεξεργαστών $P_{i,j,*}$ έχει τη γραμμή $A[i,*]$ και τη στύλη $B[* ,j]$. Πιο συγκεκριμένα, ο επεξεργαστής $P_{i,j,k}$ θα πρέπει να έχει τα $A[i,k]$ και $B[k,j]$

Το πρότυπο επιμοιωνίας για την κατανομή των στοιχείων του πίνακα A μεταξύ των επεξεργαστών φαίνεται στα Σχ 2-3. Αρχικά, κάθε στύλη του A μετακινείται σε διαφορετικό επίπεδο ώστε η j -οστή στύλη να καταλάβει την ίδια θέση στο αεροπλάνο που αντιστοιχεί στο $k=j$, όπως αρχικά έγινε στο επίπεδο που αντιστοιχεί στο $k=0$.

Η κατανομή του A μετά τη μετακίνηση του $A[i,j]$ από τον $P_{i,j,0}$ και $P_{i,j,j}$ φαίνεται στο Σχ 1(β). Τώρα, όλες οι στύλες του A αναπαράγονται n φορές στα αντίστοιχα επίπεδα τους με μια παράλληλη ενα-πρός-όλους επιμοιωνία broadcast κατά μήκος του άξονα j . Το αποτέλεσμα αυτού του βήματος φαίνεται στο Σχ 1(ε), όπου οι n επεξεργαστές $P_{i,0,j}, P_{i,1,j}, \dots, P_{i,n-1,j}$ λαμβάνουν ένα αντίγραφο του $A[i,j]$ από τον $P_{i,j,j}$. Σε αυτό το βήμα κάθε κάθετη στύλη επεξεργαστών $P_{i,j,*}$ έχει τη γραμμή $A[i,*]$. Για την αριβία, ο επεξεργαστής $P_{i,j,k}$ έχει την $A[i,k]$.

Για τον πίνακα B , τα βήματα επιμοιωνίας είναι παρόμοια, αλλά οι ρόλοι των i και j τους δίνεται του επεξεργαστή, εναλλάσσονται. Στο πρώτο 1-1 βήμα επιμοιωνίας, το $B[i,j]$ μετακινείται από το $P_{i,j,0}$ στο $P_{i,j,i}$. Έπειτα έχουμε broadcast επιμοιωνία από τον

$P_{i,j,i}$ μεταξύ των $P_{0,j,i}, P_{1,j,i}, \dots, P_{n-1,j,i}$. Η κατανομή του B μετά από αυτή την ενα-προς-όλους broadcast επιμοιωνυία μεταξύ του i άξονα φαίνεται στο Σχ(1d). Σε αυτή τη βήλη, κάθε επεξεργαστής $P_{i,j,*}$ έχει τη βήλη $B[* , j]$. Τώρα, ο επεξεργαστής $P_{i,j,k}$ έχει το $B[k, j]$, επιηλέου του $A[i, k]$. Μετά από αυτά τα βήματα επιμοιωνυίας, τα $A[i, k]$ και $B[k, j]$ πολλαπλασιάζονται στον $P_{i,j,k}$. Τώρα, κάθε στοιχείο $C[i, j]$ του παραγόμενου πίνακα λαμβάνεται με τη βοήθεια ενός μόνου κόμβου κατά μήκος του άξονα k . Κατά τη διάρκεια αυτού του βήματος, ο επεξεργαστής $P_{i,j,o}$ συββώρενι τα αποτελέσματα των πολλαπλασιασμών από τους επεξεργαστές $P_{i,j,1}, \dots, P_{i,j,n-1}$. Στο Σχ. 1 φαίνεται αυτό το βήμα για το $C[0,0]$.

Ο DNS αλγόριθμος έχει τρία βασικά βήματα επιμοιωνυίας:

(1) Μετακίνηση των βήλών του A και των γραμμών του B για αντίστοιχα επίπεδα

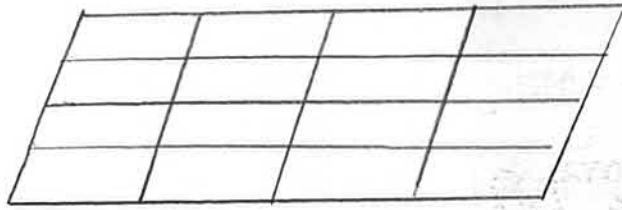
(2) Ευρέλεση μιας ενα-προς-όλους broadcast επιμοιωνυίας κατά μήκος του άξονα j για τον A και κατά μήκος του άξονα i για το B

(3) Συββώρεση αυτού κόμβου κατά μήκος του άξονα k .

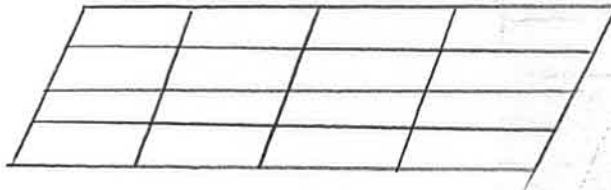
Όλες οι ανωτέρω λειτουργίες ευτελούνται εντός των υποκόβων ενός υπερκόβου με $N=n^3$ επεξεργαστές και απαιτείται χρόνος $\Theta(\log n)$. Επομένως, ο παράλληλος χρόνος για τον πολλαπλασιασμό δύο $n \times n$ πίνακων χρησιμοποιώντας τον αλγόριθμο DNS ε' ένα υπερκόβο με n^3 επεξεργαστές είναι $\Theta(\log n)$. Άρα το κόστος του αλγορίθμου DNS είναι

$\rho \cdot T_p = n^3 \Theta(\log n) = \Theta(n^3 \log n)$, δηλ. όχι βέλτιστο.

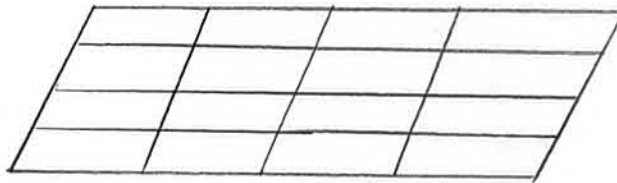
$k=3$



$k=2$



$k=1$



$k=0$ $\uparrow$ δ

$[0,3]$	$[1,3]$	$[2,3]$	$[3,3]$
$[0,2]$	$[1,2]$	$[2,2]$	$[3,2]$
$[0,1]$	$[1,1]$	$[2,1]$	$[3,1]$
$[0,0]$	$[1,0]$	$[2,0]$	$[3,0]$

$\rightarrow i$

$\Sigma x(1\alpha)$: Αρχική ικανοφύ των A και B.

$A[0,3]$	$A[1,3]$	$A[2,3]$	$A[3,3]$

$k=3$

			$B[3,3]$
			$B[3,2]$
			$B[3,1]$
			$B[3,0]$

$A[0,2]$	$A[1,2]$	$A[2,2]$	$A[3,2]$

$k=2$

		$B[2,3]$	
		$B[2,2]$	
		$B[2,1]$	
		$B[2,0]$	

$A[0,1]$	$A[1,1]$	$A[2,1]$	$A[3,1]$

$k=1$

	$B[1,3]$		
	$B[1,2]$		
	$B[1,1]$		
	$B[1,0]$		

$A[0,0]$	$A[1,0]$	$A[2,0]$	$A[3,0]$

$k=0$

$B[0,3]$			
$B[0,2]$			
$B[0,1]$			
$B[0,0]$			

Μετά τη μετακίνηση του $A[i,j]$
από τον $P[i,j,0]$ στον $P[i,j,j]$

Μετά τη μετακίνηση του $B[i,j]$
από τον $P[i,j,0]$ στον $P[i,j,i]$

$\Sigma_x(1p)$

$$\begin{array}{c} k=3 \\ \begin{array}{|c|c|c|c|} \hline A[0,3] & A[1,3] & A[2,3] & A[3,3] \\ \hline A[0,3] & A[1,3] & A[2,3] & A[3,3] \\ \hline A[0,3] & A[1,3] & A[2,3] & A[3,3] \\ \hline A[0,3] & A[1,3] & A[2,3] & A[3,3] \\ \hline \end{array} \\ \end{array}
 \begin{array}{c} C[0,0] \\ A[0,3] \times B[3,0] \\ \begin{array}{|c|c|c|c|} \hline B[3,3] & B[3,3] & B[3,3] & B[3,3] \\ \hline B[3,2] & B[3,2] & B[3,2] & B[3,2] \\ \hline B[3,1] & B[3,1] & B[3,1] & B[3,1] \\ \hline B[3,0] & B[3,0] & B[3,0] & B[3,0] \\ \hline \end{array} \end{array}$$

$$\begin{array}{c} k=2 \\ \begin{array}{|c|c|c|c|} \hline A[0,2] & A[1,2] & A[2,2] & A[3,2] \\ \hline A[0,2] & A[1,2] & A[2,2] & A[3,2] \\ \hline A[0,2] & A[1,2] & A[2,2] & A[3,2] \\ \hline A[0,2] & A[1,2] & A[2,2] & A[3,2] \\ \hline \end{array} \\ \end{array}
 +
 \begin{array}{c} A[0,2] \times B[2,0] \\ \begin{array}{|c|c|c|c|} \hline B[2,3] & B[2,3] & B[2,3] & B[2,3] \\ \hline B[2,2] & B[2,2] & B[2,2] & B[2,2] \\ \hline B[2,1] & B[2,1] & B[2,1] & B[2,1] \\ \hline B[2,0] & B[2,0] & B[2,0] & B[2,0] \\ \hline \end{array} \end{array}$$

$$\begin{array}{c} k=1 \\ \begin{array}{|c|c|c|c|} \hline A[0,1] & A[1,1] & A[2,1] & A[3,1] \\ \hline A[0,1] & A[1,1] & A[2,1] & A[3,1] \\ \hline A[0,1] & A[1,1] & A[2,1] & A[3,1] \\ \hline A[0,1] & A[1,1] & A[2,1] & A[3,1] \\ \hline \end{array} \\ \end{array}
 +
 \begin{array}{c} A[0,1] \times B[1,0] \\ \begin{array}{|c|c|c|c|} \hline B[1,3] & B[1,3] & B[1,3] & B[1,3] \\ \hline B[1,2] & B[1,2] & B[1,2] & B[1,2] \\ \hline B[1,1] & B[1,1] & B[1,1] & B[1,1] \\ \hline B[1,0] & B[1,0] & B[1,0] & B[1,0] \\ \hline \end{array} \end{array}$$

$$\begin{array}{c} k=0 \\ \begin{array}{|c|c|c|c|} \hline A[0,0] & A[1,0] & A[2,0] & A[3,0] \\ \hline A[0,0] & A[1,0] & A[2,0] & A[3,0] \\ \hline A[0,0] & A[1,0] & A[2,0] & A[3,0] \\ \hline A[0,0] & A[1,0] & A[2,0] & A[3,0] \\ \hline \end{array} \\ \end{array}
 +
 \begin{array}{c} A[0,0] \times B[0,0] \\ \begin{array}{|c|c|c|c|} \hline B[0,3] & B[0,3] & B[0,3] & B[0,3] \\ \hline B[0,2] & B[0,2] & B[0,2] & B[0,2] \\ \hline B[0,1] & B[0,1] & B[0,1] & B[0,1] \\ \hline B[0,0] & B[0,0] & B[0,0] & B[0,0] \\ \hline \end{array} \end{array}$$

$\Sigma x(1c)$: Μετά την broadcast μετάδοση του $A[i,j]$ κατά μήκος του j άξονα.

$\Sigma x(1d)$: Αντίστοιχη παρανοή του B

- Η κλάση βύθου του επεξεργαστή $P(i,j,*)$ υπολογίζει το εσωτερικό γινόμενο των γραμμών $A_{i,*}$ και της στήλης $B_{*,j}$.

Αριθμητικό Παράδειγμα

Έστω δύο 3×3 πίνακες A, B και το γινόμενο τους στον πίνακα C .

$$A^{3 \times 3} \times B^{3 \times 3} = C^{3 \times 3}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 0 & 4 \\ 1 & 4 & 2 \end{pmatrix} \times \begin{pmatrix} 2 & 4 & 0 \\ 2 & 1 & 3 \\ 1 & 2 & 0 \end{pmatrix} = \begin{pmatrix} 9 & 12 & 6 \\ 8 & 16 & 0 \\ 12 & 12 & 12 \end{pmatrix}$$

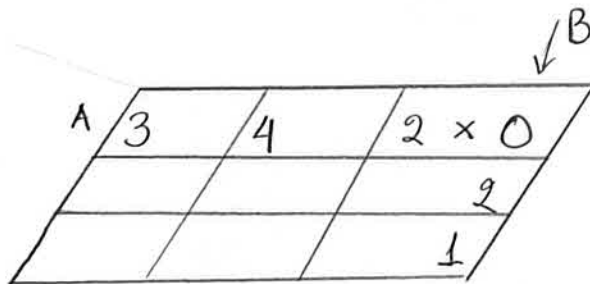
αποδοτικότερος αλγόριθμος $\rightarrow O(n^3)$

DNS αλγόριθμος

Έχουμε n^3 επεξεργαστές δηλ $3^3 = 27$ επεξεργαστές και 27 κόμβοι.

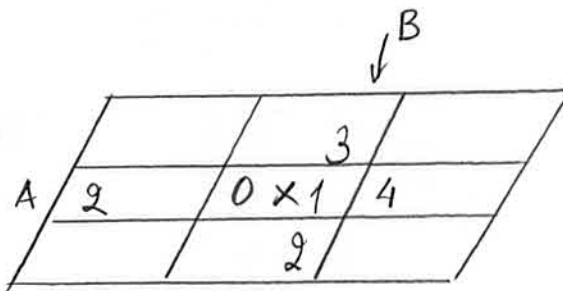
Βήμα 1:

cluster 2



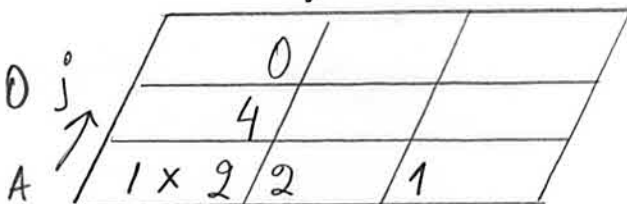
B γραμμές, A στήλες

cluster 1



$k \uparrow$

cluster 0



$\rightarrow$ η 1η γραμμή του B από πάνω προς τα πάνω

$\rightarrow$ η 1η στήλη του A από δεξιά προς τα αριστερά

Βήμα 2: Θα κάνουμε broadcast ως επιτέ σε όλο το cluster οπότε ενδιαφέρων είναι η σειρά με τον αριθμό που έχουμε και πολλαπλασιάζουμε με το στοιχείο του B.

3x0	4x0	2x0
3x2	4x2	2x2
3x1	4x1	2x1

cluster 2

+

2x3	0x3	4x3
2x1	0x1	4x1
2x2	0x2	4x2

cluster 1

+

$\begin{matrix} \uparrow k \\ j \nearrow \end{matrix}$

1x0	2x0	1x0
1x4	2x4	1x4
1x2	2x2	1x2

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

cluster 0



$$\begin{matrix} j \nearrow \\ \begin{pmatrix} 6 & 0 & 12 \\ 12 & 16 & 12 \\ 9 & 8 & 12 \end{pmatrix} \\ \rightarrow i \end{matrix} \Rightarrow \begin{matrix} i \nearrow \\ \begin{pmatrix} 9 & 12 & 6 \\ 8 & 16 & 0 \\ 12 & 12 & 12 \end{pmatrix} \\ \rightarrow j \end{matrix} = C$$

Σχηματική Λειτουργία του αλγορίθμου DNS

Παράδειγμα: $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$, $B = \begin{bmatrix} -5 & -6 \\ 7 & 8 \end{bmatrix}$ $\eta = 2$
 $N = 2^3$
 $q = 1$

• Αρχική Καταχώριση των A και B

Επεξεργαστές

P_0
 P_1
 P_2
 P_3

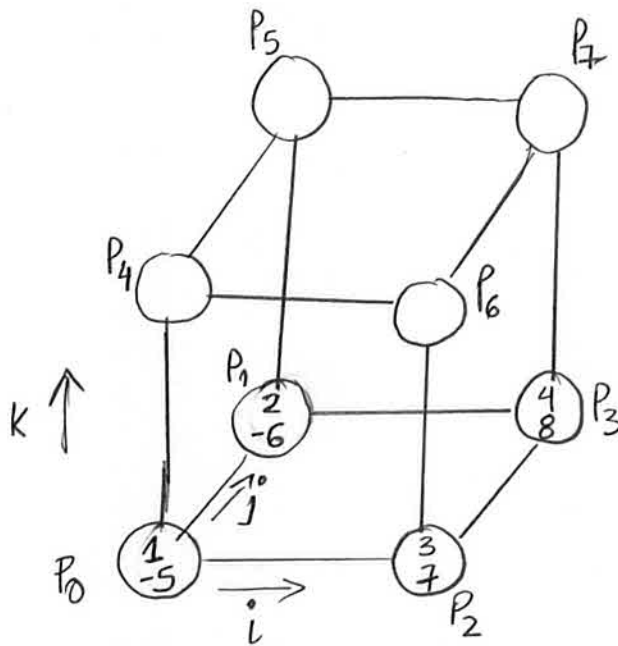
στοιχείο του A

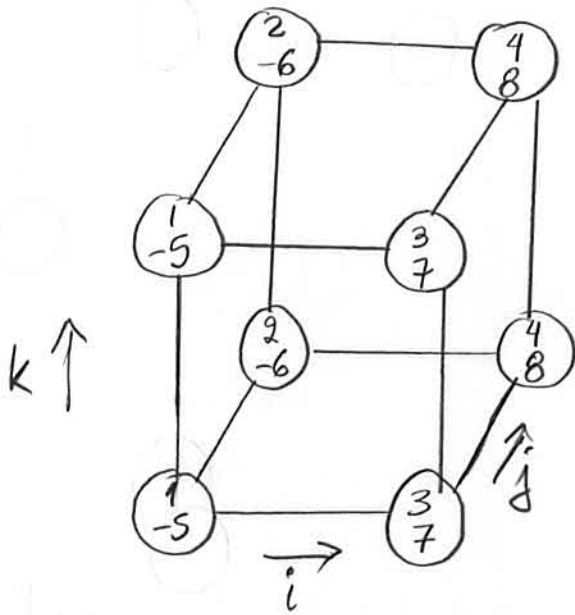
$A(0) = a_{11} = 1$
 $A(1) = a_{12} = 2$
 $A(2) = a_{21} = 3$
 $A(3) = a_{22} = 4$

στοιχείο του B

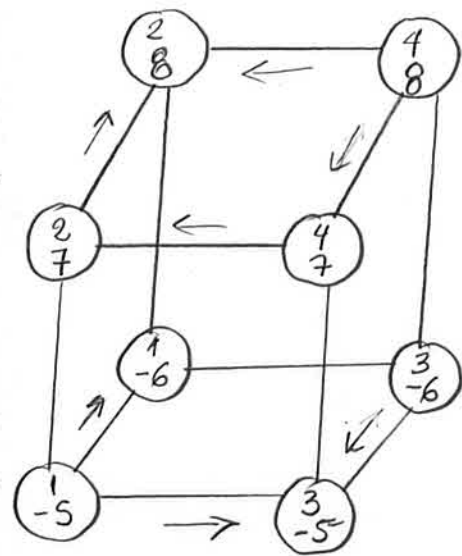
$B(0) = b_{11} = -5$
 $B(1) = b_{12} = -6$
 $B(2) = b_{21} = 7$
 $B(3) = b_{22} = 8$

P_4
 P_5
 P_6
 P_7



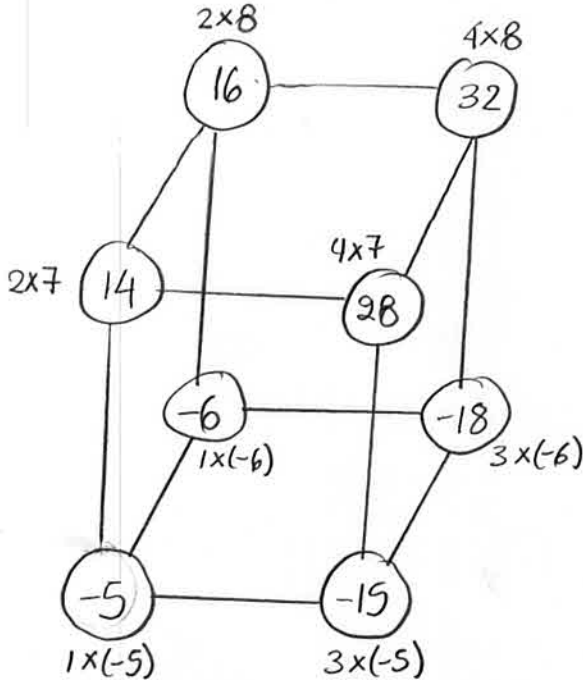


(α) Αρχική κατανομή των A και B.

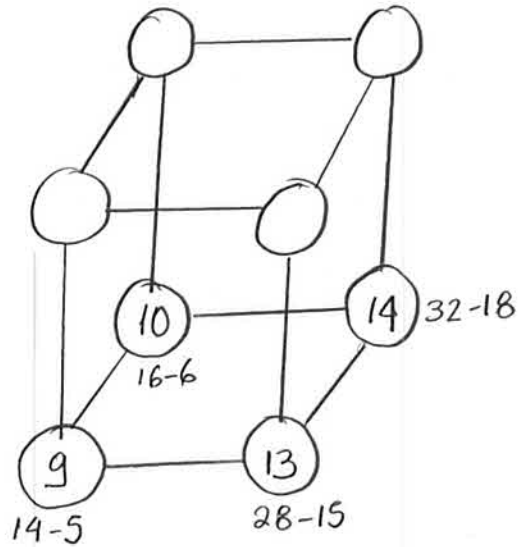


Τα 1,2 μετακινούνται προς τα πάνω, τα -5, -6 προς τα δεξιά, τα 7,8 προς τα αριστερά, τα 3,4 προς τα κάτω

(β) Μετακίνηση των $A[i,j]$ από $P(i,j,0) \rightarrow P(i,j,j)$
Μετακίνηση των $B[i,j]$ από $P(i,j,0) \rightarrow P(i,j,i)$



(γ) Υπολογισμός των γινομένων
 $C(i,j,k) = A(i,j,k) \times B(i,j,k)$
($c_{ij} = c_{ij} + a_{ik} * b_{kj}$)



(δ) Υπολογισμός των αθροισμάτων
 $c(i,j,0) = \sum_k c(i,j,k)$

DNS Αλγόριθμος με λιγότερους από n^3 επεξεργαστές

Ο DNS αλγόριθμος δεν είναι βέλτετου κόστους για n^3 επεξεργαστές, αφού ο χρόνος επεξεργασίας $\Theta(n^3 \log n)$ υπερβαίνει το $\Theta(n^3)$ που είναι η πολυπλοκότητα για τον ακολουθιακό πολλαπλασιασμό πινάκων.

Σε αυτή την υποενότητα, παρουσιάζουμε μία βέλτετου κόστους εικόνη του αλγορίθμου που χρησιμοποιεί λιγότερους από n^3 επεξεργαστές.

Υποθέτουμε ότι ο αριθμός των επεξεργαστών p ισούται με q^3 για $q < n$. Για την υλοποίηση του DNS αλγορίθμου, οι δύο πίνακες χωρίζονται σε blocks μεγέθους $(n/q) \times (n/q)$. Μπορούμε έτσι να θεωρήσουμε κάθε πίνακα σαν ένα $q \times q$ δύο διαστάσεων τετραγωνικό πίνακα blocks.

Η υλοποίηση του αλγορίθμου με q^3 επεξεργαστές είναι παρόμοια με αυτή με n^3 επεξεργαστές. Η μόνη διαφορά είναι ότι τώρα δουλεύουμε με blocks αντί με απλά στοιχεία. Αφού $1 \leq q \leq n$, το πλήθος των επεξεργαστών μπορεί να ποικίλλει μεταξύ του 1 και του n^3 .

Χρόνος Εκτέλεσης σε υπερκύβο με cut-through δρομολόγηση

Το πρώτο ένα-προς-ένα βήμα επικοινωνίας εκτελέστηκε και για τον Α και για τον Β και χρειάστηκε $t_s + t_w (n/q)^2 + t_h \log q$ χρόνο για κάθε πίνακα στον υπερκύβο με cut-through δρομολόγηση. Το δεύτερο βήμα ως ένα-προς-όλους broadcast επικοινωνίας επίσης εκτελέστηκε και για τους δύο πίνακες και χρειάστηκε $t_s \log q + t_w (n/q)^2 \log q$ χρόνο για κάθε πίνακα. Η τελική συσσώρευση ενός κόμβου εκτελέστηκε μία

μόνο φορά (για τον πίνακα C) και χρειάζεται $t_s \log q + t_w (\frac{q}{q})^2 \log q$ χρόνο. Θεωρώντας ότι κάθε ζευγάρι πολλαπλασιασμού και πρόσθεσης χρειάζεται μια χρονική μονάδα, ο πολλαπλασιασμός των $(\frac{q}{q}) \times (\frac{q}{q})$ υποπινάκων σε κάθε επεξεργαστή δέλει $(\frac{q}{q})^3$ χρόνο. Μπορούμε να αγνοήσουμε το χρόνο επιμοιωνυίας για το πρώτο βήμα της ενα-προς-ένα επιμοιωνυίας (επειδή είναι πολύ μικρότερο από το χρόνο επιμοιωνυίας της ενα-προς-όλους broadcast μετάδοσης και της συββώρευσι του κηλού κόμβου) και ο χρόνος υπολογισμού για την πρόσθεση και την τελική φάση συββώρευσις (επειδή είναι μικρότερη τάξης μεγέθους από το χρόνο υπολογισμού για τον πολλαπλασιασμό δύο υποπινάκων). Με αυτές τις υποθέσεις, έχουμε την ακόλουθη προσεγγιστική έκφραση για τον παράλληλο χρόνο εκτέλεσης του DNS αλγορίθμου σε υπερύβω με την cut-through φορολόγηση.

$$T_p \approx \left(\frac{q}{q}\right)^3 + 3t_s \log q + 3t_w \left(\frac{q}{q}\right)^2 \log q$$

Αφού $q = \sqrt[3]{p}$, έχουμε

$$T_p = \frac{q^3}{p} + t_s \log p + t_w \frac{q^2}{p^{2/3}} \log p.$$

Το κόστος αυτού του παράλληλου συστήματος είναι:

$$p T_p = q^3 + t_s p \log p + t_w q^2 p^{1/3} \log p$$

Η συνάρτηση ισοποδομότητας είναι $\Theta(p \log p)$. Το παράλληλο σύστημα είναι βέλτετου κόστους για $q^3 = \underline{O}(p \log p)$ ή $p = \underline{O}(q^3 / \log q)$

Συνοψικά :

Χρόνοι επικοινωνίας :

$$1. T_1^{\text{comm}} = t_s + t_w \left(\frac{n}{q}\right)^2 + t_h \log q$$

$$2. T_2^{\text{comm}} = t_s \log q + t_w \left(\frac{n}{q}\right)^2 \log q$$

$$3. T_3^{\text{comm}} = t_s \log q + t_w \left(\frac{n}{q}\right)^2 \log q$$

Χρόνος υπολογισμού :

$$T^{\text{comp}} = (n/q)^3$$

Επειδή $T_1^{\text{comm}} \ll T_2^{\text{comm}}, T_3^{\text{comm}}$ (*) έχουμε

$$T_P = T^{\text{comp}} + T^{\text{comm}}, \text{ όπου } T^{\text{comm}} = 3T_2^{\text{comm}}$$

(γιατί $T_2^{\text{comm}} = T_3^{\text{comm}}$ και λόγω (*) θεωρώ $T_1^{\text{comm}} = T_2^{\text{comm}} = T_3^{\text{comm}}$)

$$\text{άρα } T_P \approx \left(\frac{n}{q}\right)^3 + 3t_s \log q + 3t_w \left(\frac{n}{q}\right)^2 \log q$$

και αφού $p = q^3$ έχουμε

$$T_P \approx \frac{n^3}{p} + t_s \log p + t_w \frac{n^2}{p^{2/3}} \log p$$

$$\text{Κόστος : } p T_P = n^3 + t_s p \log p + t_w n^2 p^{1/3} \log p$$

το οποίο είναι βέλτιστο για $n^3 = O(p \log p)$ ή $p = O(n^3 / \log n)$

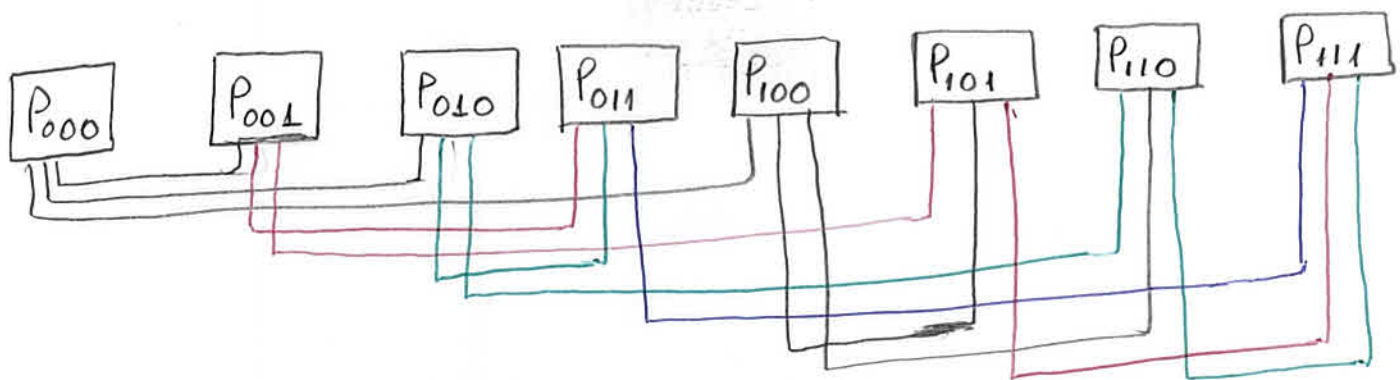
DNS Αλγόριθμος 6ε Τριδιάστατο Πλέγμα με cut-through δροφολογία

Χρησιμοποίηση χωρίς τροποποιήσεις

$$T_P = \frac{n^3}{p} + t_s \log p + t_w \frac{n^2}{p^{2/3}} \log p + 3t_h p^{1/3}$$

Σύνδεση σε υπερκύβο

Έστω $N=2^q$, για $q \geq 1$ και έστω N επεξεργαστές διαθέσιμοι $P_0, P_1, \dots, P_{N-1}$.
 Ένας κύβος q -διάστατος (ή υπερκύβος) λαμβάνεται συνδέοντας κάθε επεξεργαστή με 4 γείτονες. Οι q γείτονες P_j και P_i ορίζονται ως εξής:
 Η δυαδική αναπαράσταση του j λαμβάνεται από το i συμπληρώνοντας ένα από bit. Να σημειωθεί ότι κάθε επεξεργαστής έχει 3 γείτονες.



Αναζητώντας λοιπόν έναν ταχύτερο αλγόριθμο σε σχέση με τη δυνατότητα $MESH$ MATRIX MULTIPLICATION που κάνει χρήση n^2 επεξεργαζών, στρεφόμαστε σε μία άλλη αρχιτεκτονική. Το μοντέλο που επιλέγουμε είναι ένας SIMD υπολογιστής του οποίου οι επεξεργαστές συνδέονται με ένα δίτυο μορφής κύβου.

Έστω $N=2^g$ επεξεργαστές $P_0, P_1, \dots, P_{2^g-1}$, διαδέσμοι για $g \geq 1$.

Επίσης, έστω i και $i^{(b)}$ δύο αμέραιοι τέτοιοι ώστε $0 \leq i, i^{(b)} \leq 2^g-1$, των οποίων οι δυαδικές αναπαράξεις διαφέρουν μόνο στη θέση b , $0 \leq b < g$. Δηλαδή, αν $i_{g-1} \dots i_{b+1} \underline{i_b} i_{b-1} \dots i_1 i_0$ είναι

η δυαδική αναπαράσταση του i , τότε η

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

είναι η δυαδική αναπαράσταση του $i^{(b)}$, όπου το i_b είναι το δυαδικό ψηφίο του i_b bit.

Διακριτικό βρόχο:

1. Το ψηφίο ως προς ένα ενός αριθμού αποτελεί ο αριθμός ο οποίος προκύπτει αν αντιστρέψουμε κάθε bit του αρχικού αριθμού.

Πχ: Το ψηφίο ως προς ένα του αριθμού 10011001 είναι ο αριθμός 01100110

2. Το ψηφίο ως προς δύο ενός αριθμού αποτελεί ο αριθμός ο οποίος προκύπτει αν στο ψηφίο ως προς ένα του αριθμού αυτού προσθέσουμε το ένα.

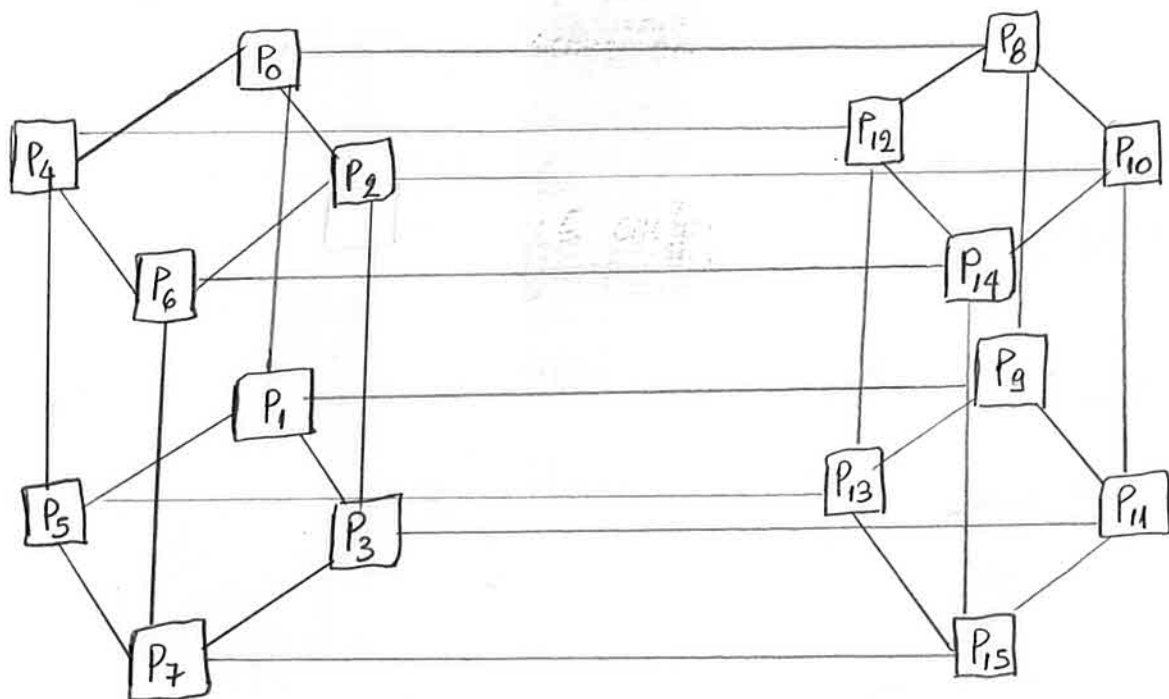
Πχ: Το ψηφίο ως προς δύο του αριθμού 10011001 είναι ο αριθμός 01100111. Προκύπτει ως εξής:

$$\begin{array}{r} \text{Αρχικός αριθμός:} \quad 10011001 \\ \text{Ψηφίο ως προς ένα:} \quad 01100110 \\ \quad \quad \quad \quad \quad \quad + 1 \\ \hline \text{Ψηφίο ως προς δύο:} \quad 01100111 \end{array}$$

Η βύθιση των επεξεργασιών σε κύβο, καθορίζει να κάθε επεξεργασία

P_i συνδέεται με τον επεξεργαστή $P_i(b)$ με δύο συνδέσμους για κάθε $0 \leq b < g$. Οι g επεξεργαστές στους οποίους συνδέεται ο P_i λέγονται γειτονισμοί του P_i .

Παράδειγμα: $g=4$, $N=2^g=2^4=16$ επεξεργαστές. Διαδύνδεση τύπου κύβου.



Έστω τώρα $n=2^q$. Θα χρησιμοποιήσουμε έναν SIMD υλοποιητή με διαδύνδεση τύπου κύβου με $N=n^3=2^{3q}$ επεξεργαστές, για τον πολλαπλασιασμό δύο $n \times n$ πινάκων A και B . (Υποθέτουμε για λόγους απλότητας της παρουσίασης ότι οι δύο πίνακες έχουν τον ίδιο αριθμό γραμμών και στηλών). Βοηθάει να απεικονίσουμε τους επεξεργαστές β' ένα τριδιάστατο $n \times n \times n$ πινάκω. Σε αυτόν τον πίνακα, ο επεξεργαστής P_r κατέχει τη θέση (i, j, k) , όπου $r=in^2+jn+k$ και $0 \leq i, j, k \leq n-1$ (αυτό αναφέρεται σαν row-major order). Έτσι λοιπόν, αν η δυαδική αναπαράσταση του r είναι γ

$v_{3q-1}, v_{3q-2}, \dots, v_0$, τότε οι δυαδικές αναπαράστασεις των i, j και k είναι: $v_{3q-1} \dots v_{2q}$, $v_{2q-1} \dots v_q$, και $v_{q-1} \dots v_0$, αντίστοιχα.

Κάθε επεξεργαστής P_r έχει τρεις 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 $(0, j, k)$, όπου $0 \leq j < n$, $0 \leq k < n$, περιέχει τα στοιχεία a_{jk} και b_{jk} στον καταχωρητή 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

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

Ο αλγόριθμος σχεδιάζεται για να εκτελέσει ταυτόχρονα τον n^3 πολλαπλασιασμό που εμπεριέχονται στον υπολογισμό των n^2 στοιχείων του C . Αυτό γίνεται σε τρεις φάσεις:

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

Φάση 2: Υπολογίζονται τα γινόμενα

$$C(i, j, k) = A(i, j, k) \times B(i, j, k)$$

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

Ο αλγόριθμος δίνεται ως συνάρτηση CUBE MATRIX MULTIPLICATION.
 Συμβολίζουμε με $\{N, r_m = d\}$ το σύνολο των ακεραίων r , $0 \leq r < N-1$
 των οποίων η δυαδική αναπαράσταση είναι η

$$r_{3q-1} \dots r_{m+1} d r_{m-1} \dots r_0$$

procedure CUBE MATRIX MULTIPLICATION(A, B, C)

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

Στη Φάση 1 του αλγορίθμου υλοποιούνται τα βήματα 1-3. Κατά τη διάρκεια του βήματος 1, τα δεδομένα που βρίσκονται αρχικά στους $A(0, j, k)$ και $B(0, j, k)$ αναγράφονται στους επεξεργαστές στις θέσεις (i, j, k) , όπου $1 \leq i \leq n$, ώστε στο τέλος αυτού του βήματος να έχουμε ότι $A(i, j, k) = a_{jk}$ και $B(i, j, k) = b_{jk}$ για $0 \leq i < n$. Στο βήμα 2 αναγράφονται τα περιεχόμενα του $A(i, j, i)$ στους επεξεργαστές στις θέσεις (i, j, k) , ώστε στο τέλος αυτού του βήματος $A(i, j, k) = a_{ji}$, όπου $0 \leq k < n$. Όμοια, το βήμα 3 αναγράφει τα περιεχόμενα του $B(i, i, k)$ στους επεξεργαστές στις θέσεις (i, j, k) , ώστε στο τέλος αυτού του βήματος $B(i, j, k) = b_{ik}$, $0 \leq j < n$. Στο βήμα 4, το γινόμενο $C(i, j, k) = A(i, j, k) \times B(i, j, k)$ υπολογίζεται ταυτόχρονα από τους επεξεργαστές στη θέση (i, j, k) για κάθε $0 \leq i, j, k < n$. Τέλος, στο

Βήμα 5, υπολογίζονται ταυτόχρονα τα n^2 αθροίσματα

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

Ανάλυση: Τα βήματα 1, 2, 3 και 5 αποτελούνται από q επαναλήψεις σταθερού χρόνου, ενώ το βήμα 4 εκτελείται σε σταθερό χρόνο. Επομένως, η συνάρτηση CUBE MATRIX MULTIPLICATION εκτελείται σε χρόνο $O(q)$ δηλαδή: $T(n) = O(\log n)$.

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

Παρατηρούμε ότι κάθε c_{ij} είναι άθροισμα n στοιχείων. Απαιτούνται $O(\log n)$ βήματα για να υπολογιστεί αυτό το άθροισμα σε ένα δίκτυο διαβάσεων με n (ή περισσότερους) επεξεργαστές. Για να το λαμβάνουμε αυτό, έστω s ο μικρότερος αριθμός βημάτων που απαιτούνται από ένα δίκτυο για τον υπολογισμό του αθροίσματος n αριθμών. Κατά τη διάρκεια του τελικού βήματος, χρειάζεται ένας πολύ επεξεργαστής για να εκτελέσει την τελευταία πρόσθεση και να παράγει το αποτέλεσμα.

Κατά τη διάρκεια του βήματος $s-1$ χρειάζονται το πολύ δύο επεξεργαστές, κατά τη διάρκεια του βήματος $s-2$ χρειάζονται το πολύ τρεις επεξεργαστές, κ.ο.κ. Επομένως, μετά από s βήματα, ο μέγιστος αριθμός των συνήθων προσθέσεων που μπορεί να εκτελεσθούν

είναι:

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

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

$$n-1 \leq 2^s - 1 \quad (\Rightarrow) \quad 2^s \geq n \quad (\Rightarrow) \quad s \geq \log n.$$

Επειδή $p(n) = n^3$, η συνάρτηση ΟΥΒΕ MATRIX MULTIPLICATION έχει κόστος $c(n) = O(n^3 \log n)$, το οποίο είναι υψηλότερο από το χρόνο εκτέλεσης του ακολουθιακού αλγορίθμου MATRIX MULTIPLICATION. Επομένως, αν και ο πολλαπλασιασμός πινάκων είναι κυβικός είναι φαινομενικά χειρότερος απ' ό,τι ένα πλέγμα, το κόστος του είναι υψηλότερο λόγω του μεγάλου αριθμού επεξεργασιών που χρησιμοποιεί.

Παράδειγμα:

Έστω $n=2^2$ και υποθέτουμε ότι έχουμε να πολλαπλασιάσουμε δύο 4×4 πίνακες, τους A και B , όπου

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

Υπάρχουν $N=2^6$ επεξεργασίες $P_0, P_1, \dots, P_{63}$ συνδεδεμένοι δ' έναν SIMD υπολογιστή με συνδεση κύβου.

Οι επεξεργασίες διατάσσονται δ' ένα τριεδιαστατικό πλέγμα $(2 \times 2 \times 2)$ που στην πράξη είναι ένας 6-διαστατικός κύβος με συνδεση που παραδείχεται με λόγους ασφαλείας.

Κάθε ένας από τους δείκτες i, j, k μοιράζεται 2 bits από τη δυαδική παράσταση $r_5 r_4 r_3 r_2 r_1 r_0$ του δείκτη r , του επεξεργαστή

P_r , δηλ. είναι: $i = r_5 r_4$, $j = r_3 r_2$ και $k = r_1 r_0$. Αρχικά, οι πίνακες

A και B φορτώνονται στους καταχωρητές $P_0, \dots, P_{15}$ όπως φαίνεται

στο 6x(2b).

Αφού $q=2$, το βήμα 1 επαναλαμβάνεται 2 φορές, μια φορά για $m=5$ και μια φορά για $m=4$.

Στην πρώτη επανάληψη (για $m=5$), όλοι οι επεξεργαστές P_r των οποίων η δυαδική παράσταση δείκτη $r = r_5 r_4 r_3 r_2 r_1 r_0$ είναι τέτοια ώστε $r_5 = 0$ 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 $P_0, \dots, P_{15}$ αντιγράφουν τα αρχικά τους περιεχόμενα στους $P_{32}, \dots, P_{47}$, αντίστοιχα, και ταυτόχρονα οι επεξεργαστές $P_{16}, \dots, P_{31}$ αντιγράφουν τα αρχικά τους περιεχόμενα (όλα μηδέν) στους επεξεργαστές $P_{48}, \dots, P_{63}$, αντίστοιχα.

Στη δεύτερη επανάληψη (για $m=4$) όλοι οι επεξεργαστές P_r των οποίων η δυαδική παράσταση δείκτη $r = r_5 r_4 r_3 r_2 r_1 r_0$ είναι τέτοια ώστε $r_4 = 0$ 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 $P_0, \dots, P_{15}$ αντιγράφουν τα περιεχόμενά τους, στους $P_{16}, \dots, P_{31}$, αντίστοιχα και ταυτόχρονα οι επεξεργαστές $P_{32}, \dots, P_{47}$ αντιγράφουν τα νέα τους περιεχόμενα (αυτά που απέκτησαν

6ου προηγούμενου επαναλήψης) 6του $P_{48}, \dots, P_{63}$, αντίστοιχα. Στο τέλος του βήματος 1, τα περιεχόμενα των 64 επεξεργαστών φαίνονται στο σχ (2c).

Στο βήμα 2 γίνονται δύο επαναλήψεις: μία για $m=1$ και μία για $m=0$. Κατά τη διάρκεια της 1ης επανάληψης, όπου $m=1$, όλοι οι επεξεργαστές των οποίων η δυαδική παράσταση δείκτη $r = r_5 r_4 r_3 r_2 r_1 r_0$ είναι τέτοια ώστε $r_1 = r_5$, αντιγράφουν τα περιεχόμενα των καταχωρητών A 6του καταχωρητές A των επεξεργαστών με δυαδική παράσταση δείκτη $r_5 r_4 r_3 r_2 r_1' r_0$. Για παράδειγμα, οι επεξεργαστές P_0 και P_1 αντιγράφουν τα περιεχόμενα των καταχωρητών A 6του καταχωρητές A των P_2 και P_3 , αντίστοιχα. Κατά τη διάρκεια της 2ης επανάληψης, όπου $m=0$, όλοι οι επεξεργαστές με δυαδική παράσταση δείκτη $r = r_5 r_4 r_3 r_2 r_1 r_0$ τέτοια ώστε $r_0 = r_4$ αντιγράφουν τα περιεχόμενα των καταχωρητών A 6του 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 και P_2 αντιγράφουν τα περιεχόμενα των καταχωρητών A 6του καταχωρητές A των P_1 και P_3 , αντίστοιχα. Στο τέλος του βήματος αυτού κάθε στοιχείο του A έχει γίνει κώδικας διαμέσου κάθε "γραφής" του $6 \times (2a)$.

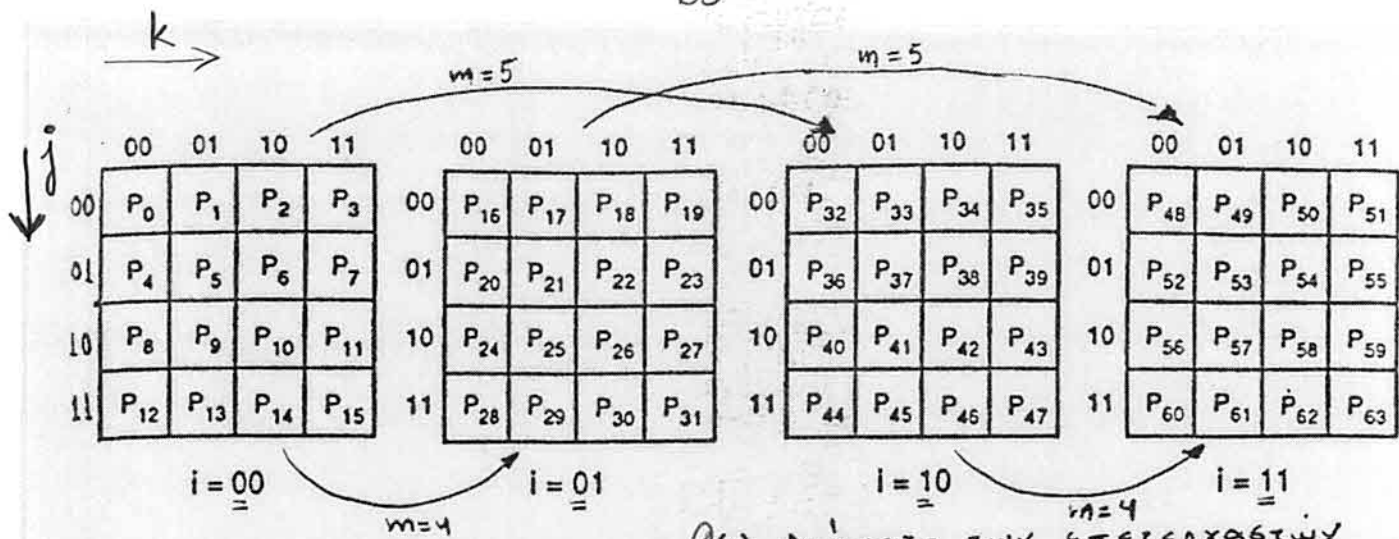
Το βήμα 3 είναι ισοδύναμο με το βήμα 2 εκτός του ότι ε'αυτό αντιγράφεται ένα στοιχείο του B διαμέσου κάθε "στήλης". Τα περιεχόμενα των 64 επεξεργαστών μετά το τέλος των βημάτων 2 και 3 φαίνονται στο $6 \times (2d)$.

Στο βήμα 4, με όλοι τους επεξεργαστές να λειτουργούν ταυτόχρονα, κάθε επεξεργαστής υπολογίζει το γινόμενο των A και B καταχωριτών που και καταχωρεί το αποτέλεσμα στον καταχωριτή του C .

Το βήμα 5 αποτελείται από 2 επαναλήψεις: μια για $m=4$ και μια για $m=5$.

Στην 1η επανάληψη (για $m=4$) προστίθενται τα περιεχόμενα των καταχωριτών C των ζευγών επεξεργαστών των οποίων οι δυαδικές παραστάσεις των δευτέρων τους διαφέρουν στο bit r_4 . Και οι δύο επεξεργαστές διατηρούν το αποτέλεσμα. Το ίδιο γίνεται και στη 2η επανάληψη (για $m=5$) για τους επεξεργαστές με δεύτες που διαφέρουν στο bit r_5 . Το τελικό αποτέλεσμα αποθηκεύεται στους επεξεργαστές

$P_0, P_1, \dots, P_{15}$ όπως φαίνεται στο 6x(2c).



2(a) Διάταξη των επεξεργασιών σε ένα 3-διάστατο πίνακα

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

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

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

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

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

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

2(c) Τα περιεχόμενα των 64 επεξεργασιών μετά το τέλος του Βημάτος 1

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

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

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

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

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

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

$2(e)$

$$-944 = 17(-7) + 23(-18) + 27(-13) + 3(-20)$$

$$-1688 = 17(-25) + 23(-30) + 27(-21) + 3(-2)$$

Πολλαπλασιασμός πινάκων σε ένα CRCW SM SIMD υπολογιστή

Σε αυτή την ενότητα παρουσιάσουμε ένα παράλληλο αλγόριθμο για τον πολλαπλασιασμό πινάκων ο οποίος είναι πιο γρήγορος και έχει χαμηλότερο κόστος από τη συνάρτηση CUBE MATRIX MULTIPLICATION.

Υποθέτουμε ότι οι αλληλοσυγχορούσες για γράψιμο αυξετώνονται ως εξής:

Όταν μερικοί επεξεργαστές επιδιώκουν να γράψουν στην ίδια θέση μνήμης τότε το άθροισμα των αριθμών που είναι για γράψιμο αποθηκεύεται δ' αυτή τη θέση. Ο αλγόριθμος είναι μία άμεση παραλληλοποίηση του αναγνωστικού αλγορίθμου MATRIX MULTIPLICATION.

Χρησιμοποιεί μηχανική επεξεργαστή για τον πολλαπλασιασμό ενός μηχανή πίνακα A με έναν $n \times k$ πίνακα B . Συνεπώς, οι επεξεργαστές διατάσσονται δ' ένα μηχανή $n \times k$ πίνακα του οποίου ο κάθε επεξεργαστής έχει τρεις δείκτες (i, j, s) , όπου $1 \leq i \leq n$, $1 \leq j \leq n$, και $1 \leq s \leq k$.

Αρχικά, οι πίνακες A και B βρίσκονται στην κοινή μνήμη. Όταν ο αλγόριθμος τελειώσει το γινόμενο τους $C = A \cdot B$ βρίσκεται επίσης στην κοινή μνήμη.

Ο αλγόριθμος αυτός είναι γνωστός σαν CRCW MATRIX MULTIPLICATION.

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) $c_{ij} \leftarrow 0$

(2) $c_{ij} \leftarrow a_{is} \times b_{sj}$

end for

end for

end for

Ανάλυση: Είναι προφανές ότι η συνάρτηση CRCW MATRIX MULTIPLICATION τρέχει σε σταθερό χρόνο.

Αφού το πλήθος των επεξεργασιών είναι $p(n) = n^3$ και ο χρόνος εκτέλεσης $t(n) = O(1)$, το κόστος θα είναι:

$$c(n) = p(n) \times t(n) = n^3 \times O(1) = O(n^3)$$

Παρατηρούμε ότι το κόστος $c(n)$ είναι το ίδιο με εκείνο του ακολουδιασίου αλγορίθμου της συνάρτησης MATRIX MULTIPLICATION

Παράδειγμα: Ένας CRCW SM SIMD υπολογιστής με 64 επεξεργαστές μπορεί να πολλαπλασιάσει τους δύο πίνακες A και B του παραδείγματος της προηγούμενης ενότητας σε σταθερό χρόνο. Και τα 16 γινόμενα να φανωθούν στο $6 \times (2d)$ υπολογίζονται ταυτόχρονα και αποθηκεύονται (προστίθενται) σε ομάδες των 4, στην κατάλληλη θέση του C. Για παράδειγμα οι P_1, P_2, P_3 και 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 $C_{11} = -944$.

Πολλαπλασιασμός πίνακα-με-διάνυσμα.

Το πρόβλημα που μελετάται ε'αυτή των ενόχτων είναι ο πολλαπλασιασμός ενός $m \times n$ πίνακα A με το $n \times 1$ διάνυσμα U για την παραγωγή του $m \times 1$ διανύσματος V , όπως φαίνεται για $m=3$ και $n=4$:

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

Τα στοιχεία του V λαμβάνονται από

$$v_i = \sum_{j=1}^n a_{ij} \times u_j, \quad 1 \leq i \leq m.$$

Φυσικά και αυτό το πρόβλημα αποτελεί ειδική περίπτωση του προβλήματος πολλαπλασιασμού πίνακα $\times$ πίνακα. Το μελετάμε ξεχωριστά για να δείξουμε τη χρήση δύο διατύπων διασύνδεσης στην επίλυση πράξεων πίνακων, δηλ.

1. των πολλαπλασιασμών σε γραμμικό (μονοδιάστατο) πίνακα
2. του πολλαπλασιασμού σε δέντρο
3. τη συνέλιξη (convolution).

1. Πολλαπλασιασμός πίνακα με διάνυσμα σε μονοδιάστατο πίνακα (linear array)

Ο αλγόριθμος αυτός σχεδιάστηκε για να τρέξει σε μονοδιάστατο πίνακα με m επεξεργαστές $P_1, P_2, \dots, P_m$.

Ο επεξεργαστής P_i χρησιμοποιείται για να υπολογίσει το στοιχείο v_i του V . Αρχικά, το v_i είναι μηδέν. Ο πίνακας A και το διάνυσμα U τροφοδοτούνται στον πίνακα, όπως φαίνεται στο 6x.3, για $n=4$ και $m=3$.

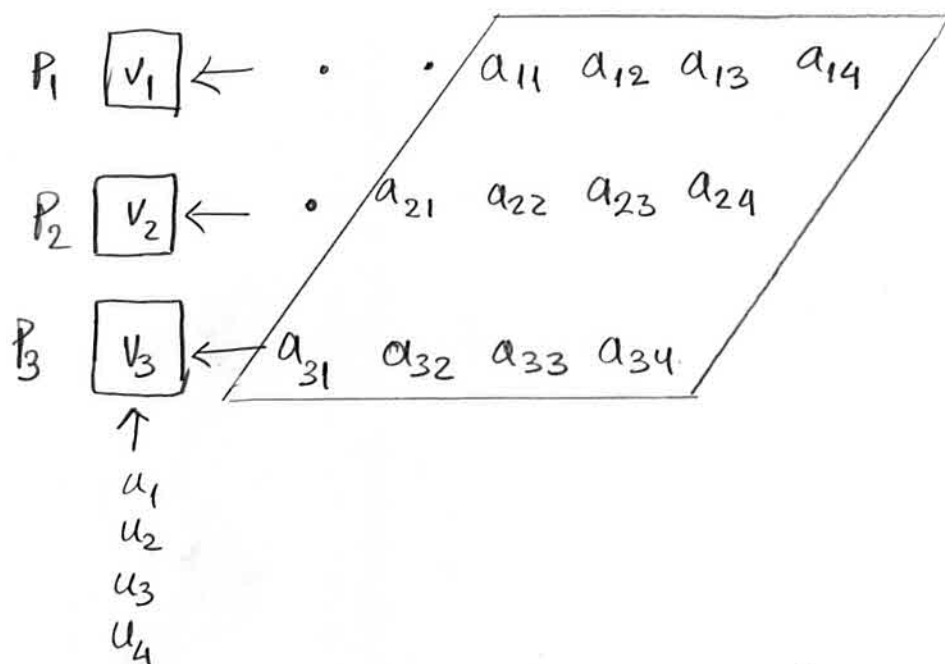
Κάθε επεξεργαστής P_i έχει 3 παραχωρητές a, u και v . Όταν ο P_i δέχεται δύο εισόδους a_{ij} και u_j , τότε

(i) λαμβάνει το a_{ij} στον a και το u_j στον u

(ii) πολλαπλασιάζει το a με το u

(iii) προσδίδει το αποτέλεσμα στο v_i και

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



Σχ 3 : Πολλαπλασιασμός πίνακα με διάνυσμα τροφοδοτώντας τον είσοδο του γραμμικού πίνακα

Να σημειωθεί ότι η γραφή i του A έχει ακριβώς μια χρονική μονάδα πίσω από τη γραφή $i+1$, για $1 \leq i \leq m-1$.

Αυτό εξασφαλίζει ότι η αλγ συντακτική των u_j των ακριβώς χρονική σειρά. Ο αλγόριθμος δίνεται από τη συνάρτηση LINEAR MV MULTIPLICATION

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

Ανάλυση:

Το στοιχείο a_{im} χρειάζεται $m+n-1$ βήματα για να φτάσει 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.

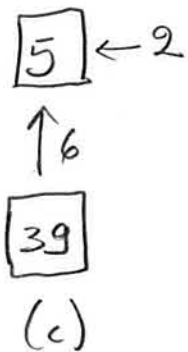
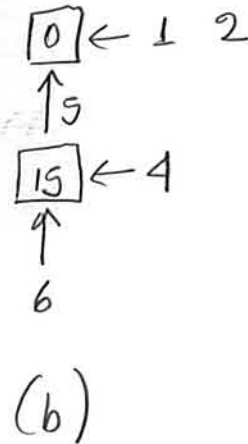
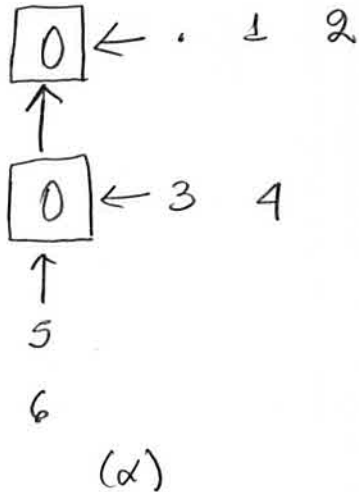
Υποθέτουμε ότι $m \leq n$, ο χρόνος εκτέλεσης της συνάρτησης LINEAR MV MULTIPLICATION είναι $t(n) = O(n)$. Αφού χρησιμοποιούνται m επεξεργαστές, ο αλγόριθμος έχει κόστος $c(n) = O(n^2)$ που είναι βέλτερο αφού για να διαβάσει η είσοδος ακολουθιακά απαιτούνται $O(n^2)$ βήματα.

Παράδειγμα:

Έστω $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ και $U = \begin{pmatrix} 5 \\ 6 \end{pmatrix}$ όπου $m=n=2$

Η συμπεριφορά του συστήματος LINEAR MV MULTIPLICATION φαίνεται στο Σχ 4.

Λύση



Σχ 4: Πολλαπλασιασμός πίνακα x διανύσας χρησιμοποιώντας τη συσκευή LINEAR MV MULTIPLICATION

2. Πολλαπλασιασμός πίνακα με διάνυσμα σε δέντρο

Όπως παρατηρήθηκε στην προηγούμενη ενότητα, ο πολλαπλασιασμός πίνακα $\times$ διάνυσμα απαιτεί $m+n-1$ βήματα σε γραμμικό πίνακα. Είναι δυνατό να μειωθεί αυτός ο χρόνος σε $m-1+\log n$ βήματα αν ο πολλαπλασιασμός γίνει σε ένα SIMD υπολογιστή με συνδεση δένδρου.

Η διάταξη φαίνεται στο 2×5 για $m=3$ και $n=4$.

Έστω ότι το δένδρο έχει n τερματικούς επεξεργαστές (φύλλα) $P_1, P_2, \dots, P_n$ $n-2$ ενδιαμέσους επεξεργαστές $P_{n+1}, P_{n+2}, \dots, P_{2n-2}$ και 1 επεξεργαστή ρίζα P_{2n-1} .

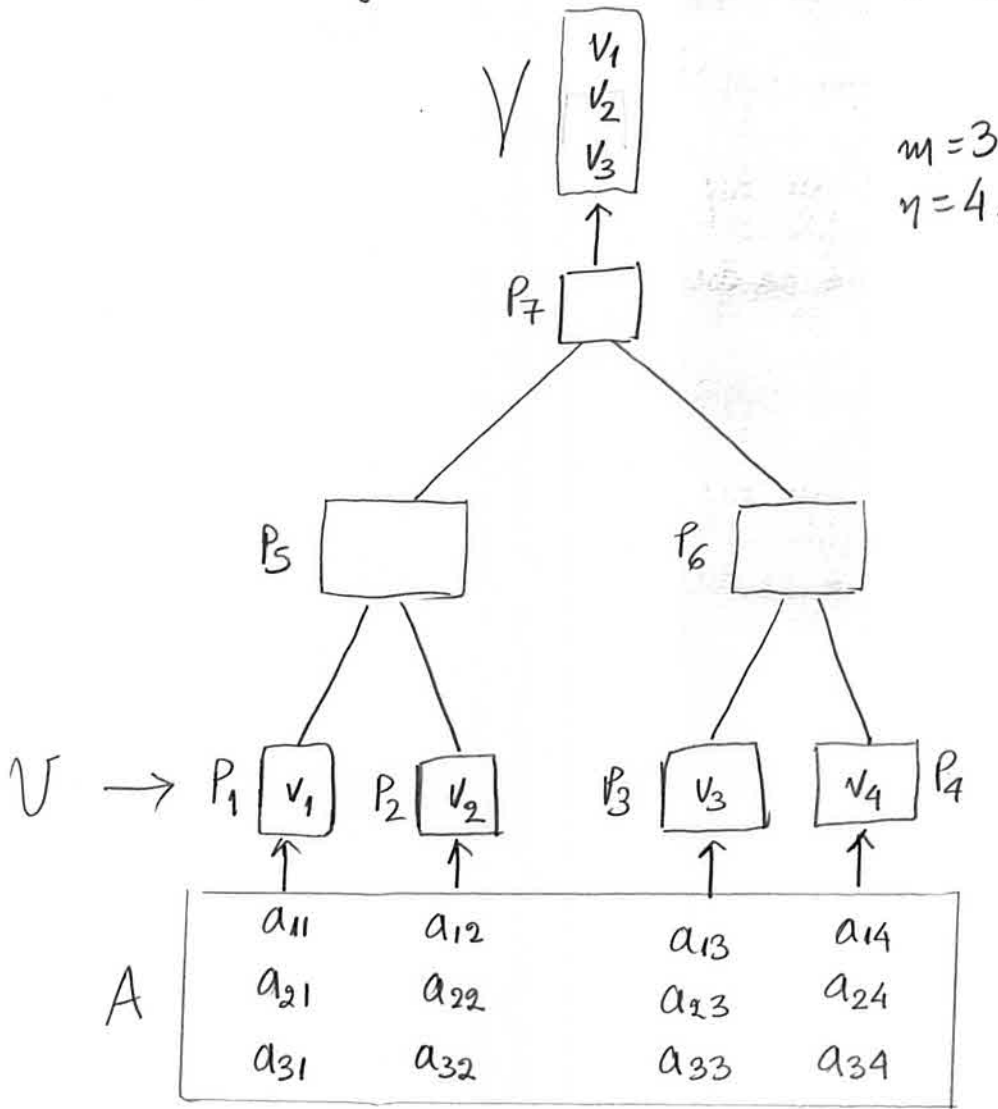
Οι αυραίοι (τερματικοί-φύλλα) επεξεργαστές P_i , $1 \leq i \leq n$, αποδιδούν το u_i κατά την εκτέλεση του αλγορίθμου.

Ο πίνακας A προωθείται στο δένδρο γραμμή γραμμή, ένα στοιχείο ανά αυραίο επεξεργαστή (φύλλο). Όταν ο αυραίος επεξεργαστής P_i λαβεί το a_{ji} , υπολογίζει το $u_i \times a_{ji}$ και στέλνει το γινόμενο στους γονείς του. Όταν ένας ενδιαμέσος επεξεργαστής P_k ή η ρίζα δέχεται δύο εισόδους από τα παιδιά του, τις προωδύει και στέλνει το αποτέλεσμα στο γονέα του. Τελικά, το v_j παράγεται από τη ρίζα.

Αν οι γραμμές του A εισέρχονται στους αυραίους επεξεργαστές P_i σε συνεχείς χρονικές στιγμές, τότε τα στοιχεία του γινομένου V επίσης παράγονται ως έξοδος από τη ρίζα σε συνεχείς χρονικές στιγμές.

-72-

Σχ 5 : Υπολογισμός συνδεδεμένος σε δένδρο για πολλαπλό πίνακα x διάνυσμα



Ο αλγόριθμος δίνεται από τη συνάρτηση TREE MV
MULTIPLICATION

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 n 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

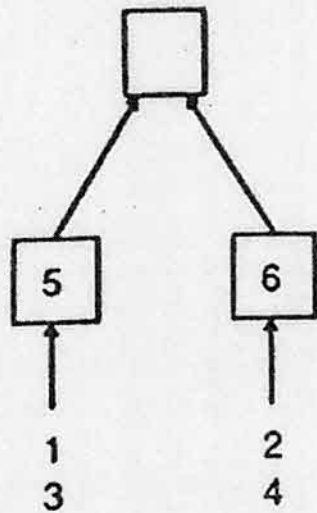
end while

end for.

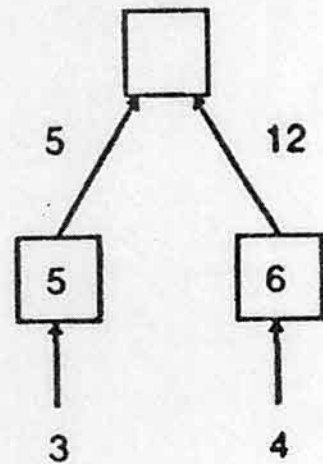
Παράδειγμα
 $m = n = 2$

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

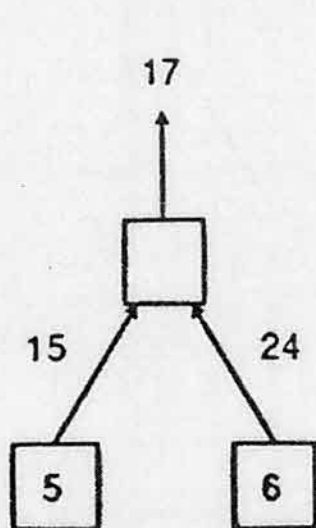
$$U = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$



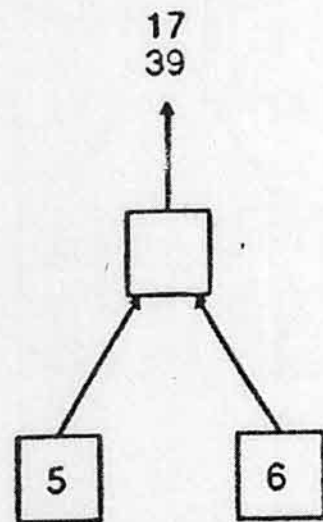
(a)



(b)



(c)



(d)

Η συνηρηφορά της συνάρτησης TREE MV MULTIPLICATION

Ανάλυση

Από τη στιγμή που η πρώτη γραφή του A καταχθεί στον αμείωτο επεξεργαστή έως ότου να καταχθεί το v_1 , στη ρίζα απαιτούνται $\log n$ βήματα. Συνολικά $m-1$ βήματα απαιτούνται για να καταχθεί 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 η συνάρτηση TREE MV MULTIPLICATION είναι γρηγορότερη από τη συνάρτηση LINEAR MV MULTIPLICATION αφού χρησιμοποιείται περίπου διττός αριθμός επεξεργασιών.

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

3. Συνέλιξη

Ολοκληρώνουμε αυτή την ενότητα παρουσιάζοντας μια εφαρμογή αλγορίθμου για τον πολλαπλασιασμό πίνακα με διάνυσμα.

Έστω μια ακολουθία σταθερών αριθμών $\{w_1, w_2, \dots, w_n\}$ και μια ακολουθία εισόδου $\{x_1, x_2, \dots, x_n\}$. Ζητείται να υπολογιστεί η ακολουθία εξόδου $\{y_1, y_2, \dots, y_{2n-1}\}$ που ορίζεται από τον τύπο

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

Ο υπολογισμός αυτός είναι γνωστός ως συνέλιξη και έχει μεγάλο ενδιαφέρον στην ψηφιακή επεξεργασία σήματος

Παράδειγμα: $n=3$

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

Το συναπάνε σε πολλαπλασιασμό πολωνύμων ηx

$$p(x) = x^2 - 5x + 6, \quad q(x) = x^2 + x + 1 \quad \text{Τότε}$$

$$p(x) \cdot q(x) = x^4 - 4x^3 + 2x^2 + x + 6.$$

Αν $p = [1 \ -5 \ 6]$, $q = [1 \ 1 \ 1]$ Τότε

$$\begin{bmatrix} 1 & 0 & 0 \\ -5 & 1 & 0 \\ 6 & -5 & 1 \\ 0 & 6 & -5 \\ 0 & 0 & 6 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 \\ -5 \cdot 1 + 1 \cdot 1 + 0 \cdot 1 \\ 6 \cdot 1 - 5 \cdot 1 + 1 \cdot 1 \\ 0 \cdot 1 + 6 \cdot 1 - 5 \cdot 1 \\ 0 \cdot 1 + 0 \cdot 1 + 6 \cdot 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -4 \\ 2 \\ 1 \\ 6 \end{bmatrix} = p(x)q(x)$$

Πολλαπλασιασμός πίνακα με διάνυσμα με διαχωρισμό

Η ενότητα αυτή ασχολείται με το πρόβλημα του πολλαπλασιασμού ενός πυκνού $n \times n$ πίνακα A με ένα $n \times 1$ διάνυσμα u με αποτέλεσμα το $n \times 1$ διάνυσμα v .

Ο ακολουθιακός αλγόριθμος γι' αυτό το πρόβλημα δίνεται από τη συνάρτηση MAT-VECT

```
procedure MAT-VECT (A, u, v)
```

```
  for  $i := 0$  to  $n-1$  do
```

```
     $v_i := 0$ 
```

```
    for  $j := 0$  to  $n-1$  do
```

```
       $v_i := v_i + a_{ij} u_j$ 
```

```
    end for
```

```
  end for
```

Υποθέτουμε ότι χρειαζόμαστε μια χρονική μονάδα για κάθε βήμα πολλαπλασιασμού - πρόσθεσης, ο ακολουθιακός χρόνος εκτέλεσης είναι: $W = n^2$.

Υπάρχουν τουλάχιστον 3 μορφές παράλληλων αλγορίθμων για τον πολλαπλασιασμό πίνακα $\times$ διάνυσμα

- Οριζόντιες λωρίδες (Rowwise stripping)
- Κατακόρυφες λωρίδες (Columnwise stripping)
- Τετράγωνα τεμάχια επιφανείας (blocks)-checkerboarding)

Διαχωρισμός κατά γραμμές - Ορίζοντες Λωρίδες.

Αρχικά θεωρούμε την περίπτωση όπου ο $n \times n$ πίνακας κατανέμεται μεταξύ των n επεξεργαστών με τέτοιο τρόπο ώστε ο κάθε επεξεργαστής να αποδίδει μια πλήρη γραμμή του πίνακα. Το $n \times 1$ διάνυσμα u κατανέμεται με τέτοιο τρόπο ώστε κάθε επεξεργαστής να αποδίδει ένα από τα στοιχεία του. Η αρχική κατανομή του πίνακα και του διανύσματος για διαχωρισμό κατά γραμμές φαίνεται στο Σχ (6a).

Ο επεξεργαστής P_i αρχικά αποδίδει το u_i και τα στοιχεία $a_{i0}, a_{i1}, \dots, a_{i,n-1}$ του A και αναλαμβάνει τον υπολογισμό του v_i .

Το διάνυσμα u πολλαπλασιάζεται με κάθε γραμμή του πίνακα (procedure MAT-VECT), έτσι κάθε επεξεργαστής χρειάζεται ολόκληρο το διάνυσμα. Αφού κάθε επεξεργαστής ζευγώνει με μόνο ένα στοιχείο του u , απαιτείται να έχουμε all-to-all broadcast για την κατανομή όλων των στοιχείων σε όλους τους επεξεργαστές. Στο Σχ (6b) φαίνεται αυτό ο βύθια επιμοιουσιότητας. Αφού το διάνυσμα u κατανομισθεί μεταξύ των επεξεργαστών (Σχ (6c)), ο επεξεργαστής P_i υπολογίζει τη συνιστώσα

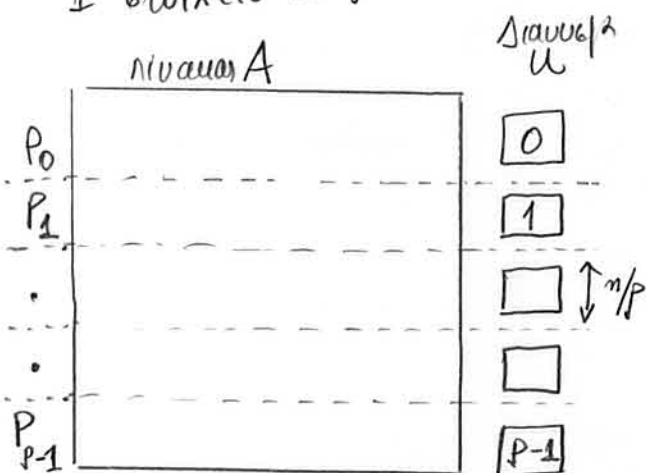
$$v_i = \sum_{j=0}^{n-1} a_{ij} u_j$$

όπως φαίνεται στο Σχ (6d), το διάνυσμα-αποτέλεσμα v αποδίδεται πάλι με τον ίδιο τρόπο όπως το αρχικό διάνυσμα u .

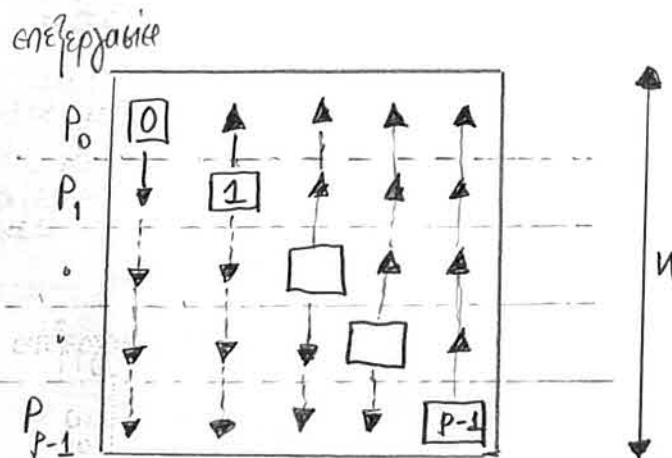
-79-

Αριθμός επεξεργασιών $P=n$

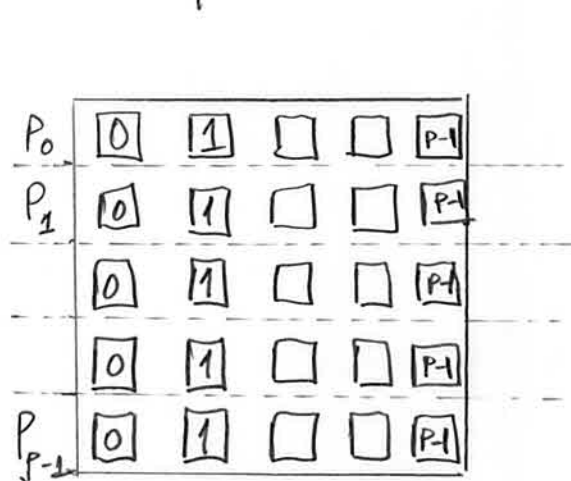
↓ γραμμή A } → ↓ επεξεργασία
↓ στοιχείο u



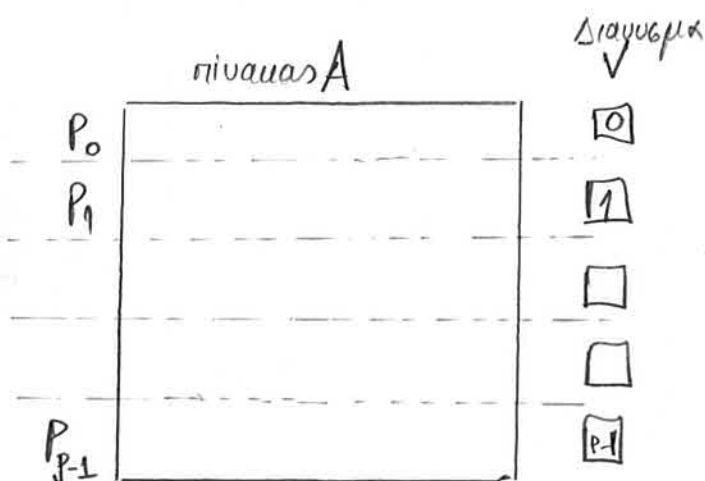
(α) Αρχικός διαχωρισμός του A και του αρχικού διανύσματος u



(β) Κατανομή του διανύσματος u σε όλους τους επεξεργαστές με μετάδοση all-to-all broadcast.



(γ) Μετά την κατανομή ολόκληρο το διάνυσμα u κατανέμεται στον κάθε επεξεργαστή



(δ) Τελική κατανομή του πίνακα και του διανύσματος αποτελέσματος y .

Σχ 6: Πολλαπλασιασμός ενός $n \times n$ πίνακα με το $n \times 1$ διάνυσμα χρησιμοποιώντας διαχωρισμό κατά γραμμές.

Χρόνος Παράλληλης εκτέλεσης με $p=4$ επεξεργαστές

Ξεκινώντας με ένα στοιχείο διανύσματος ανά επεξεργαστή, η all-to-all broadcast των στοιχείων του διανύσματος κατά μήκος η επεξεργαστών απαιτεί $\Theta(n)$ χρόνο σε οποιαδήποτε αρχιτεκτονική.

Ο πολλαπλασιασμός μιας γραμμής του A με το u , σε κάθε επεξεργαστή, απαιτεί $\Theta(n)$ χρόνο.

Επομένως, η όλη διαδικασία ευημερώνεται από η επεξεργαστές σε $\Theta(n)$ χρόνο με αποτέλεσμα το κόστος να είναι

$$p(n) \cdot t(n) = \Theta(n^2)$$

Ο παράλληλος αλγόριθμος είναι βέλτερου κόστους γιατί η πολυπλοκότητα του βερσιονού αλγορίθμου είναι $\Theta(n^2)$.

Χρήση λιγότερων επεξεργαστών ($p < n$)

Θαρούμε την περίπτωση όπου οι p επεξεργαστές που χρησιμοποιούμε είναι τέτοιοι ώστε $p < n$. Κάθε επεξεργαστής αρχικά αποθηκεύει n/p πλήρεις γραμμές του πίνακα και ένα τμήμα του διανύσματος μεγέθους n/p . Αφού το διάνυσμα u πολλαπλασιάζεται με κάθε γραμμή του πίνακα, κάθε επεξεργαστής χρειάζεται ολόκληρο το διάνυσμα (δηλ. όλα τα τμήματα αποθηκεύονται σε ξεχωριστές επεξεργαστές).

Από αυτή απορροφά all-to-all broadcast (Σχ (6b), (6c)). Η all-to-all broadcast λαμβάνει χώρα κατά μήκος p επεξεργαστών και αποστέλλει μηνύματα μεγέθους n/p . Έχουμε δηλ.

$$T_{\text{comm}} = t_s \log p + t_w \frac{n}{p} (p-1)$$

Μετά από αυτό το βήμα επιμερισμού, κάθε επεξεργαστής πολλαπλασιάζει τις n/p γραμμές του A με τα n στοιχεία του x στον ίδιο επεξεργαστή για να παράγει n/p στοιχεία στο τελικό διάνυσμα. (6x(6d))

Θεωρώντας:

$$T_{comp} = n^2/p.$$

Χρόνος παράλληλος εκτέλεσης σε υπερκώβο

Η all-to-all broadcast μεταδοει των μηνυμάτων μεγέθους n/p κατά μήκος p επεξεργαστών δίνει χρόνο

$$T_{comm} = t_s \log p + t_w \left(\frac{n}{p}\right) (p-1)$$

Για μεγάλο p , αυτός ο χρόνος μπορεί να προσεγγιστεί ως

$$T_{comm} = t_s \log p + t_w n$$

Μετά την επιμερισμό, κάθε επεξεργαστής ζοδεύει $n^2/p = T_{comp}$ χρόνο στον πολλαπλασιασμό των n/p γραμμών με το διάνυσμα. Θεωρώντας, ο παράλληλος χρόνος εκτέλεσης αυτής της διαδικασίας είναι:

$$T_p = \frac{n^2}{p} + t_s \log p + t_w n$$

και το κόστος είναι $C = p T_p = n^2 + t_s p \log p + t_w n p$.

Ο αλγόριθμος είναι βέλτιστου κόστους για $p = O(n)$.

Χρόνος παράλληλης εκτέλεσης σε διδιάστατο δίκτυο με αναδιπλώση (wrap-around mesh)

Σε ένα διδιάστατο πλέγμα με αναδιπλώση, ο όρος που σχετίζεται με τον t_s για all-to-all broadcast επικοινωνίας στον πολλαπλασιασμό πίνακα-διανύσματος με διαμέριση κατά γραμμές είναι $2(\sqrt{p}-1)$

Οπότε, όμοια με την περίπτωση του υπερκύβου είναι

$$T_p = \frac{n^2}{p} + 2t_s(\sqrt{p}-1) + t_w \cdot n$$

Διαχωρισμός σε τετράγωνα τεμάχια (blocks-checkerboard)

Στην ενότητα αυτή εστιάζουμε τον παράλληλο πολλαπλασιασμό πίνακα για την περίπτωση που ο πίνακας καταμερίζεται μεταξύ των επεξεργαστών χρησιμοποιώντας διαχωρισμό σε τετράγωνα.

Ένα βλοκείο ανά επεξεργαστή

Ξεκινάμε με την απλή περίπτωση κατά την οποία ο $n \times n$ πίνακας μοιράζεται σε n^2 επεξεργαστές έτσι ώστε κάθε επεξεργαστής να αποθηκεύει ένα μόνο βλοκείο. Το $n \times 1$ διάνυσμα u καταμερίζεται μόνο στην τελευταία στήλη των n επεξεργαστών, καθενας από τους οποίους αποθηκεύει ένα μόνο βλοκείο του διανύσματος. Αφού ο αλγόριθμος πολλαπλασιάζει τα βλοκεία του διανύσματος u με τα αντίστοιχα βλοκεία σε κάθε γραμμή του πίνακα, το διάνυσμα πρέπει να καταμεριστεί έτσι ώστε το $i^{\text{οστό}}$ βλοκείο του διανύσματος να είναι διαθέσιμο στο $i^{\text{οστό}}$ βλοκείο κάθε γραμμής του πίνακα.

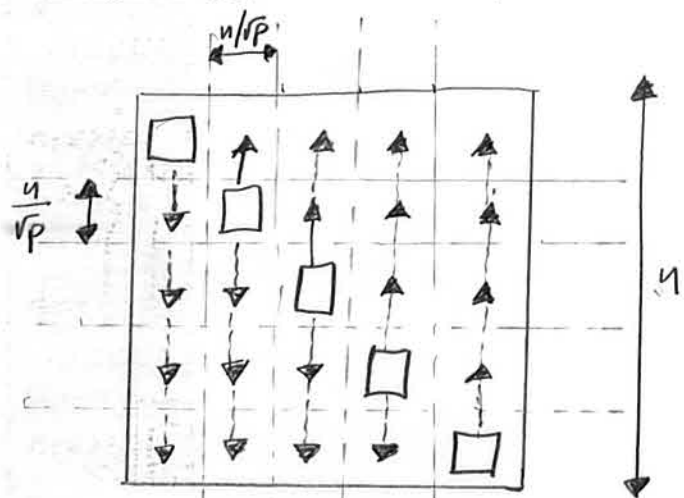
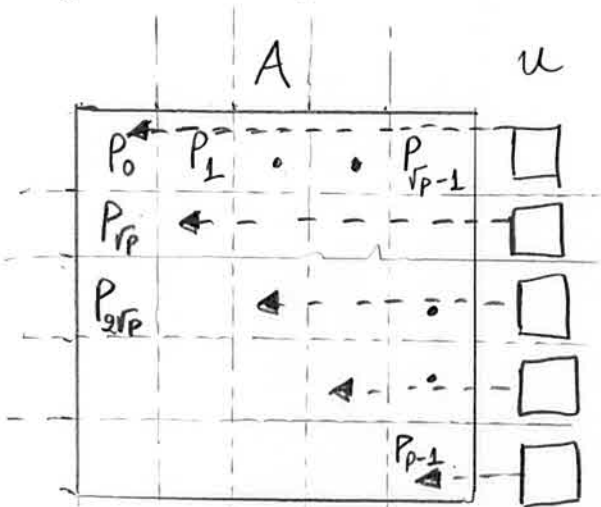
Τα βήματα επιμοιωνυίας φαίνονται στα σχ.(7a) και (7b).

Πριν του πολλαπλασιασμού, τα στοιχεία του πίνακα και του διανύσματος πρέπει να είναι στις ίδιες σχετιζόμενες θέσεις όπως στο σχ.(7c)

Στο σχ (7a) φαίνεται, το 1ο βήμα επιμοιωνυίας για την block διαίρεση που ενδογραφείζει το διάνυσμα u κατά μήκος της διαγώνιου. Το 2ο βήμα αναγράφει τα στοιχεία του διανύσματος από κάθε διαγώνιο επεξεργαστή σε όλους τους επεξεργαστές της αντίστοιχης στύλης. Στο σχ (7b), το βήμα αυτό αποτελείται από η ταυτόχρονα all-to-all broadcast λειτουργίες, μία για κάθε στύλη επεξεργαστών. Μετά από αυτά τα δύο βήματα επιμοιωνυίας, ο κάθε επεξεργαστής πολλαπλασιάζει το στοιχείο του πίνακα του με το αντίστοιχο στοιχείο του u . Για να λάβουμε το διάνυσμα-αποτέλεσμα v , τα γινόμενα που υπολογίζονται για κάθε γραμμή πρέπει να προστεθούν, αφήνοντας τα αθροίσματα στην τελευταία στύλη των επεξεργαστών. Στο σχ (7c) δειχνει το βήμα που απαιτεί μία συβωρρευση όλου κόμβου σε κάθε γραμμή με τον τελευταίο επεξεργαστή της γραμμής για προορισμό. Ο παράλληλος πολλαπλασιασμός πίνακα-διανύσματος συζητείται μετά το βήμα της συβωρρευσης.

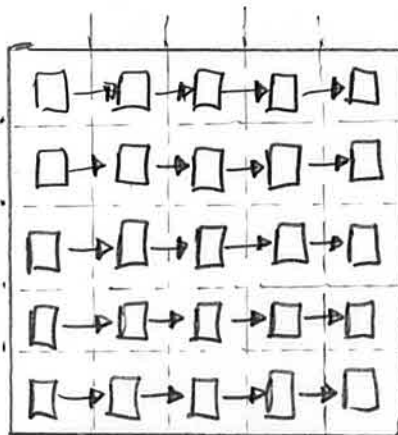
Διαχωρισμός σε τετραγωνα
αριθμός επεξεργασιών $p = n^2$

$\left\{ \begin{array}{l} 1 \text{ κομμάτι του } A \\ 1 \text{ κομμάτι του } u \end{array} \right\} \rightarrow 1 \text{ επεξεργασία}$

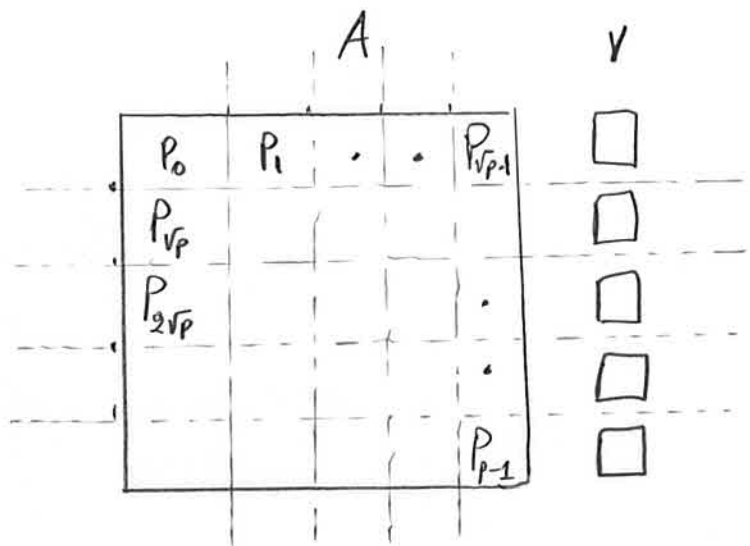


(α) Κατανομή αρχικών
δεδομένων και βήματα
επισημάνει για την ευθυγράφιση
του u κατά μήκος της
διαγωνίου

(β) Εκπομπή ενός προς όλους
των τριγώνων του u κατά
μήκος των βελών



(γ) Συνένωση αλυσίδας
κόμβου (ενός προς όλους)
των μερικών αποτελεσμάτων



(δ) Τελική κατανομή του
διαυδμένου αποτελέσματος v.

Σχ 7: Πολυδιασποράς πυαμα-διαυδμένος σε blocks.

Ο αλγόριθμος αυτός εκτελεί 3 βασικές λειτουργίες επιμορφωσίας που είναι:

1. Ένας προς ένα επιμορφωσία για τη διεύθυνση του διανύσματος u στην κόρη διαγώνιο
2. Ένας προς όλους εκπομπή του κώδε εισόδου του u στον u επεξεργαστή της κώδε βέλους
3. Συμπίεση κώδε κόμβου σε κώδε γραμμή

Οι παραπάνω λειτουργίες απαιτούν χρόνο

$O(n)$ για πλέγμα

$O(\log n)$ στον υπερκώδο.

Χρόνος παράλληλης εκτέλεσης με $p=n^2$ επεξεργαστές

	T_p	Κόστος (pT_p)
Mesh :	$O(n)$	$O(n^3)$
Υπερκώδος:	$O(\log n)$	$O(n^2 \log n)$

Αρα, ο αλγόριθμος δεν είναι βέλτιστος.

Αριθμός επεξεργασιών $p < n^2$.

$$\left. \begin{array}{l} \frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}} \text{ υποπινάκας του } A \\ \frac{n}{\sqrt{p}} \text{ στοιχεία του } u \end{array} \right\} \rightarrow 1 \text{ επεξεργασία}$$

Χρόνος παράλληλης εκτέλεσης

A) Διδιάστατο διάνυσμα με cut-trough δρομολόγηση.

$$T_p = T_{\text{comp}} + T_1 + T_2 + T_3$$

όπου

$$T_{\text{comp}} = n^2/p \quad \leftarrow \text{υπολογιστικός χρόνος}$$

$$T_1 = t_s + t_w \cdot \frac{n}{\sqrt{p}} + t_h \cdot \sqrt{p} \quad \leftarrow \text{χρόνος διεύθυνσης του } u \text{ στην} \\ \text{κώρια διαχώνιο}$$

$$T_2 = \left(t_s + t_w \cdot \frac{n}{\sqrt{p}}\right) \log \sqrt{p} + t_h \sqrt{p} \quad \leftarrow \text{χρόνος αποστολής του } u \text{ κατά} \\ \text{βήματα}$$

$$T_3 = \left(t_s + t_w \cdot \frac{n}{\sqrt{p}}\right) \log \sqrt{p} + t_h \sqrt{p} \quad \leftarrow \text{χρόνος συλλογής}$$

Άρα,
$$T_p \cong \frac{n^2}{p} + t_s \log p + t_w \frac{n}{\sqrt{p}} \log p + 3 t_h \sqrt{p}$$

B) Διδιακταρο δίστω με store and forward δροβολόγηση

$$T_p \approx \frac{u^2}{p} + 2t_s \sqrt{p} + 3t_w \cdot u$$

Σύγκριση των διαχωρισμών

1) κατά οριζόντιες λωρίδες

2) κατά τετράγωνα εμάχια

$$1) T_p = \frac{u^2}{p} + 2t_s (\sqrt{p} - 1) + t_w \cdot u$$

$$2) T_p = \frac{u^2}{p} + t_s \log p + t_w \frac{u}{\sqrt{p}} \log p + 3t_n \sqrt{p}$$

Ο πολλαπλασιασμός πίνακα με διάνυσμα είναι ταχύτερος στο διαχωρισμό σε block τετραγωνικές περιοχές

Το αποτέλεσμα είναι το ίδιο και στον υπερκύβο.