MSc – CNT

ΠΑΡΑΛΛΗΛΑ και ΚΑΤΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ

Lecture #8

ΑΛΓΟΡΙΘΜΟΙ ΚΑΤΑΝΕΜΗΜΕΝΗΣ ΜΝΗΜΗΣ
(Πολλαπλασιασμός Πινάκων)

Πολλαπλασιασμός Πινάκων

Έστω ότι έχουμε δύο πίνακες *A* και *B* διαστάσεων *nxn* και ότι το αποτέλεσμα πρέπει να αποθηκευτεί στον πίνακα *C* (*nxn*)

```
for (i=0; i<n; i++)
    for (j=0; j<n; j++)
        begin
            c[i][j] = 0;
            for (k=0; k<n; k++)
                c[i][j]=c[i][j]+a[i][k]*b[k][j];
        end
```

$$A \quad x \quad B \quad = \quad C$$

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

$$c_{12} = a_{10}xb_{02} + a_{11}xb_{12} + a_{12}xb_{22} + a_{13}xb_{32}$$

Σχήμα 5.15 Πολλαπλασιασμός δύο δυσδιάστατων πινάκων *A* και *B*

Πολλαπλασιασμός Πινάκων
(παραλληλοποίηση)

Ο παραπάνω υπολογισμός απαιτεί $O(n^3)$ πράξεις, οι οποίες είναι οργανωμένες σε n^2 επαναλήψεις. Οι υπολογισμοί μέσα σε κάθε μία από τις επαναλήψεις αυτές είναι ανεξάρτητοι μεταξύ τους. Μπορεί να υλοποιηθεί παράλληλος αλγόριθμος ο οποίος να επιτυγχάνει

- παράλληλο χρόνο $O(n^2)$ χρησιμοποιώντας $p=n$ επεξεργαστές
- ή/και και $O(n)$ χρησιμοποιώντας $p=n^2$ επεξεργαστές
- *ή/και* $O(\log n)$ χρησιμοποιώντας $O(n^3)$ επεξεργαστές

Οι πρώτες δύο από τις παραπάνω μορφές παραλληλοποίησης είναι *βέλτιστες* από άποψη συνολικού κόστους καθώς το γινόμενο του αριθμού των επεξεργαστών επί τον παράλληλο χρόνο που επιτυγχάνεται, είναι ίσο και στις δύο περιπτώσεις με $O(n^3)$.

Η τρίτη από τις παραπάνω μορφές, αποτελεί και το σχετικό κάτω όριο υπολογισμών σε παράλληλο περιβάλλον για το συγκεκριμένο πρόβλημα.

Πολλαπλασιασμός Πινάκων (παραλληλοποίηση)

- Το σύνολο των παραπάνω διαπιστώσεων ισχύουν αναφορικά με τον παράλληλο χρόνο ολοκλήρωσης των υπαγορευόμενων υπολογισμών. Σε περιβάλλοντα κατανεμημένης μνήμης ωστόσο πολύ σημαντικό παράγοντα διαδραματίζουν οι απαιτούμενες επικοινωνίες (ανταλλαγές μηνυμάτων) μεταξύ των επεξεργαστών.
- Η ανάγκη μέριμνας και αποδοτικής διαχείρισης των απαιτήσεων επικοινωνίας μεταξύ των επεξεργαστών σε ένα περιβάλλον κατανεμημένης μνήμης κατά την πράξη του πολλαπλασιασμού πινάκων, πηγάζει κατά κύριο λόγο από την ανάγκη χρήσης δυσανάλογου πλήθους στοιχείων εισόδου.
- Για παράδειγμα αν έχουμε στη διάθεσή μας $p=n$ επεξεργαστές και αναθέσουμε σε κάθε επεξεργαστή να εκτελέσει τους υπολογισμούς για μία γραμμή του πίνακα αποτελέσματος (n στοιχεία), αρκεί μεν από τον πίνακα A να έχει κάθε επεξεργαστής μόνο την αντίστοιχη γραμμή (n στοιχεία), θα χρειαστεί ωστόσο σε κάθε περίπτωση και ολόκληρο τον πίνακα B (n^2 στοιχεία) προκειμένου να ολοκληρώσει τους υπολογισμούς του. Η λύση να έχει κάθε επεξεργαστής ένα αντίγραφο του πίνακα B στην κατοχή του δεν είναι αποδεκτή.

Πολλαπλασιασμός Πινάκων
(παραλληλοποίηση - στόχοι)

Στο παραπάνω πλαίσιο, βασική επιδίωξη των διαφόρων παράλληλων αλγόριθμων πολλαπλασιασμού πινάκων αποτελεί, υποθέτοντας ότι τα δεδομένα εισόδου (πίνακες *A* και *B*) είναι ισοκατανεμημένα σε όλους τους επεξεργαστές (είτε εξαρχής είτε ως αποτέλεσμα άλλων πρότερων υπολογισμών), να φέρουν εις πέρας με βέλτιστο τρόπο από πλευράς υπολογισμών την απαιτούμενη πράξη:

(i) με τη μικρότερη δυνατή επιβάρυνση χώρου (κρατώντας το σύνολο των στοιχείων εισόδου κατά το δυνατόν ισοκατανεμημένα σε όλους τους επεξεργαστές και χωρίς πλεονασμούς καθ' όλη τη διάρκεια του αλγορίθμου), και

(ii) διακινώντας με τον αποδοτικότερο δυνατό τρόπο μέσω του δικτύου διασύνδεσης τα επιπρόσθετα στοιχεία εισόδου που χρειάζεται κάθε επεξεργαστής πέραν των αρχικών στοιχείων που έχει στην κατοχή του.

Πολλαπλασιασμός Πινάκων
(αναδρομικά - υποπίνακες)

Έστω ότι κάθε ένας από τους πίνακες *A* και *B* χωρίζεται σε *q* υποπίνακες διαστάσεων $(n/\sqrt{q})x(n/\sqrt{q})$. Στους υποπίνακες αυτούς μπορούμε να αναφερόμαστε για ευκολία ως $A_{i,j}$ και $B_{i,j}$ ($i,j=0\ldots\sqrt{q}-1$). Στο παραπάνω πλαίσιο, ο πολλαπλασιασμός των πινάκων *A* και *B* μπορεί να γραφτεί ως πολλαπλασιασμός υποπινάκων, όπως φαίνεται παρακάτω:

```
for (i=0; i<√q; i++)
    for (j=0; j<√q; j++)
        begin
            C[i,j] := 0;
            for (k=0; k<√q; k++)
                C[i,j] := C[i,j]+A[i,k]*B[k,j];
        end
```

Κάθε πολλαπλασιασμός υποπινάκων του εσωτερικού επαναληπτικού βρόγχου ($A_{i,k}*B_{k,j}$) μπορεί να γραφεί ως *ο αναλυτικός πολλαπλασιασμός των δύο αντίστοιχων υποπινάκων για διάσταση* (*n*) ίση με $n/\sqrt{q}$.

Πολλαπλασιασμός Πινάκων
(αναδρομικά - παράδειγμα)

$$\left[\begin{array}{cc|cc} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ \hline a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{array}\right] x \left[\begin{array}{cc|cc} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ \hline b_{20} & b_{21} & b_{22} & b_{23} \\ b_{30} & b_{31} & b_{32} & b_{33} \end{array}\right] = \left[\begin{array}{cc|cc} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ \hline c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{array}\right]$$

$$\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} x \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix} = \begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix}$$

$$C_{00} = A_{00}\, x\, B_{00} + A_{01}\, x\, B_{10} =$$

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

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

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

Σχήμα 5.16 Αναδρομικός υπολογισμός του πολλαπλασιασμού πινάκων

Πολλαπλασιασμός Πινάκων

(Παράλληλοι Αλγόριθμοι σε περιβάλλον κατανεμημένης μνήμης)

- Row-based Αλγόριθμος
- Ο Αλγόριθμος του Cannon
- Ο Αλγόριθμος του Fox
- Pipeline-based Αλγόριθμος

Row-based Αλγόριθμος (κατανομή κατά γραμμές)

Έστω δύο πίνακες A(NxN) και B(NxN) τους οποίους θέλουμε να πολλαπλασιάσουμε χρησιμοποιώντας p=N επεξεργαστές οι οποίοι είναι συνδεδεμένοι σε τοπολογία δακτυλίου, με αποτέλεσμα έναν νέο πίνακα C(NxN) και έστω επίσης ότι:

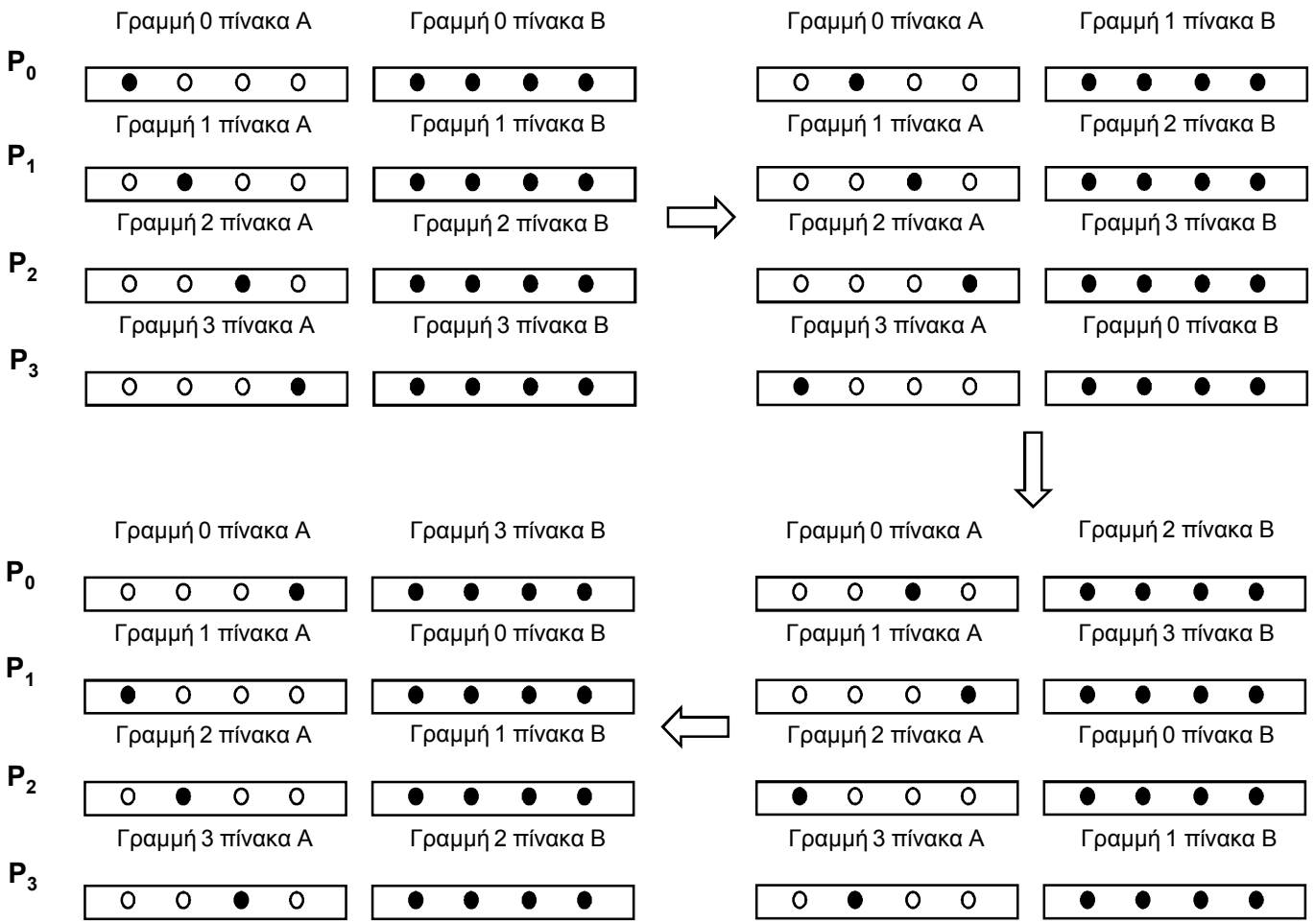
1. Κάθε επεξεργαστής P_i (i=0…N-1) έχει στην κατοχή του (στην τοπική του μνήμη) αρχικά τα στοιχεία της γραμμής i του πίνακα A και της γραμμής i του πίνακα B.

2. Κάθε επεξεργαστής είναι υπεύθυνος για τον υπολογισμό των στοιχείων της γραμμής i του πίνακα C.

Row-based Αλγόριθμος (βήματα εκτέλεσης)

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 στέλνει τη γραμμή του πίνακα *B* που κατέχει στον προηγούμενό του επεξεργαστή P_{i-1} (στον P_{p-1} αν $i=0$) στην τοπολογία δακτυλίου. Έτσι κάθε επεξεργαστής με το πέρας του βήματος αυτού, θα έχει λάβει μία νέα (την επόμενη στη σειρά) γραμμή του πίνακα *B* με βάση την οποία θα συνεχίσει του 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 γραμμές του πίνακα *B*.
4. Μετά από την τελευταία επανάληψη, επαναλαμβάνει άλλη μία φορά το βήμα 1 ολοκληρώνοντας έτσι τους απαιτούμενους υπολογισμούς με βάση την τελευταία γραμμή του πίνακα *B* που παρέλαβε από τον προηγούμενό του επεξεργαστή.

Row-based Αλγόριθμος - Παράδειγμα



Πολλαπλασιασμός πινάκων σε τοπολογία δακτυλίου

Row-based Αλγόριθμος - Επέκταση

Ο αλγόριθμος μπορεί εύκολα να επεκταθεί και για αριθμό επεξεργαστών μικρότερος από το *n*. Σε αυτήν την περίπτωση, υποθέτοντας χωρίς βλάβη της γενικότητας ότι το *n* είναι ακέραιο πολλαπλάσιο του *p*, θα πρέπει να θεωρήσουμε καταρχήν ότι κάθε επεξεργαστής 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 στη συνέχεια:

1. Κάθε επεξεργαστής P_i κατά το βήμα 1 θα κάνει υπολογισμούς για τα στοιχεία *n/p* γραμμών του πίνακα *C*, δηλαδή 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 επεξεργαστής κατά το βήμα 2 θα διακινεί προς τον προηγούμενό 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 επικοινωνίας/ανταλλαγής μηνυμάτων).

Row-based Αλγόριθμος – Χρόνος Ολοκλήρωσης

(για την γενική περίπτωση - αριθμός επεξεργαστών μικρότερος από το *n - n/p* γραμμές σε κάθε επεξεργαστή)*:*

Computation Time: $T_{comp} = O(n^3/p)$

Communication Time: $T_{comm} = O(n^2)$

Ο χρόνος επικοινωνίας με βάση το *πρότυπο του δικτύου* είναι ίσος με

$$T'_{comm} = (p-1)\, t_0 + (p-1)\, r\, n^2/p \approx p\, t_0 + r\, n^2$$

Ο Αλγόριθμος του Cannon

- Ο αλγόριθμος του Cannon αποτελεί τον πιο γνωστό από τους αλγόριθμους που αναπτύχθηκαν με σκοπό να εκμεταλλευτούν κατάλληλα το μεγαλύτερο αριθμό συνδέσμων επικοινωνίας που προσφέρει μια τοπολογία *δυσδιάστατου πλέγματος.*
- Εκμεταλλεύεται δε, στη γενική περίπτωση ($p<n^2$), τον αναδρομικό ορισμό του πολλαπλασιασμού πινάκων (διάσπαση σε υποπίνακες).
- Υποθέτουμε αρχικά ότι έχουμε στη διάθεσή μας αριθμό επεξεργαστών ίσο με το πλήθος των στοιχείων των πινάκων (δηλαδή, ένα στοιχείο ανά επεξεργαστή: p=n^2), οι οποίοι είναι οργανωμένοι σε τοπολογία δυσδιάστατου πλέγματος διαστάσεων √px√p (ή αλλιώς nxn).
- Αριθμούμε τους επεξεργαστές με βάση τη θέση τους (συντεταγμένες) στην τοπολογία του πλέγματος, αντί για την κλασσική αρίθμηση (P0,…Pp-1), δηλαδή ως Pi,j (i=0…n-1, j=0…n-1).
- Υποθέτουμε επίσης ότι αρχικά κάθε επεξεργαστής Pi,j κατέχει τα στοιχεία ai,j και bi,j από τους αρχικούς πίνακες A και B και ότι είναι υπεύθυνος για τον υπολογισμό του στοιχείου ci,j του πίνακα C.

Ο Αλγόριθμος του Cannon
(βήματα εκτέλεσης)

1. Αρχικά τα στοιχεία των δύο πινάκων μετακινούνται γειτονικά πάνω στο πλέγμα προκειμένου να βρεθούν τελικά σε κατάλληλες θέσεις για την εφαρμογή της διαδικασίας της ανά ζεύγη σταδιακής διάχυσής τους που θα ακολουθήσει. Πιο συγκεκριμένα, κάθε ολόκληρη γραμμή τάξης *i* του πίνακα *A* μετακινείται κυκλικά *i* θέσεις αριστερά, και κάθε ολόκληρη στήλη τάξης *j* του πίνακα *B* μετακινείται *j* θέσεις πάνω.

2. Έτσι τελικά, κάθε επεξεργαστής $P_{i,j}$ θα καταλήξει να κατέχει τα στοιχεία $a_{i,j+i}$, $b_{i+j,j}$, τα οποία αποτελούν ένα από τα ζεύγη που συμμετέχουν στο άθροισμα υπολογισμού της τελικής τιμής του στοιχείου αποτελέσματος $c_{i,j}$ για το οποίο είναι υπεύθυνος ο εν λόγω επεξεργαστής.

3. Κάθε επεξεργαστής πολλαπλασιάζει τα στοιχεία που κατέχει, ξεκινώντας έτσι τον υπολογισμό του αθροίσματος που απαιτείται για το στοιχείο αποτελέσματος $c_{i,j}$ για το οποίο είναι υπεύθυνος ($c_{i,j} += a_{i,j+i} x b_{i+j,j}$).

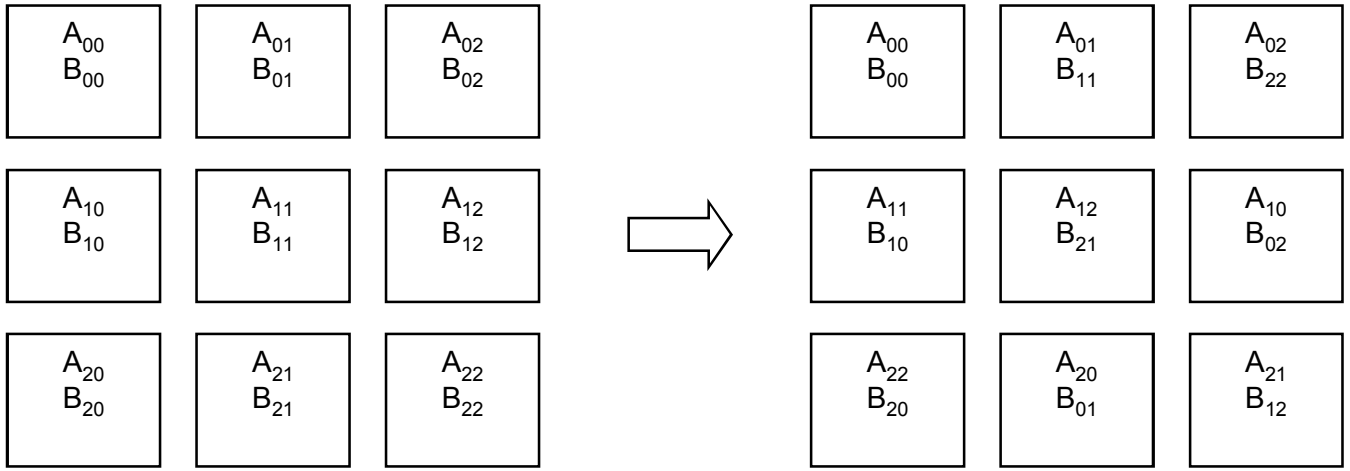
Ο Αλγόριθμος του Cannon
(βήματα - συνέχεια)

4. Κάθε επεξεργαστής στέλνει το στοιχείο που κατέχει από τον πίνακα *A* μία θέση αριστερά, στον επεξεργαστή $P_{i,j-1}$ (και κυκλικά - δηλαδή αν *j=0* στέλνει το στοιχείο στον επεξεργαστή $P_{i,n-1}$), και το στοιχείο που κατέχει από τον πίνακα *B* μία θέση πάνω, στον επεξεργαστή $P_{i-1,j}$ (και κυκλικά - δηλαδή αν *i=0* στέλνει τα στοιχείο στον επεξεργαστή $P_{n-1,j}$).

5. Κάθε επεξεργαστής πολλαπλασιάζει τα στοιχεία που μόλις παρέλαβε από τον δεξιό του και τον κάτω του επεξεργαστή - $P_{i,j+1}$ και $P_{i+1,j}$ αντίστοιχα), συμπληρώνοντας έτσι αθροιστικά έναν επιπλέον όρο για τον υπολογισμό του αθροίσματος που απαιτείται για το στοιχείο αποτελέσματος $c_{i,j}$ για το οποίο είναι υπεύθυνος ($c_{i,j}$+=$a_{i,j+i+1}$x$b_{i+j+1,j}$).

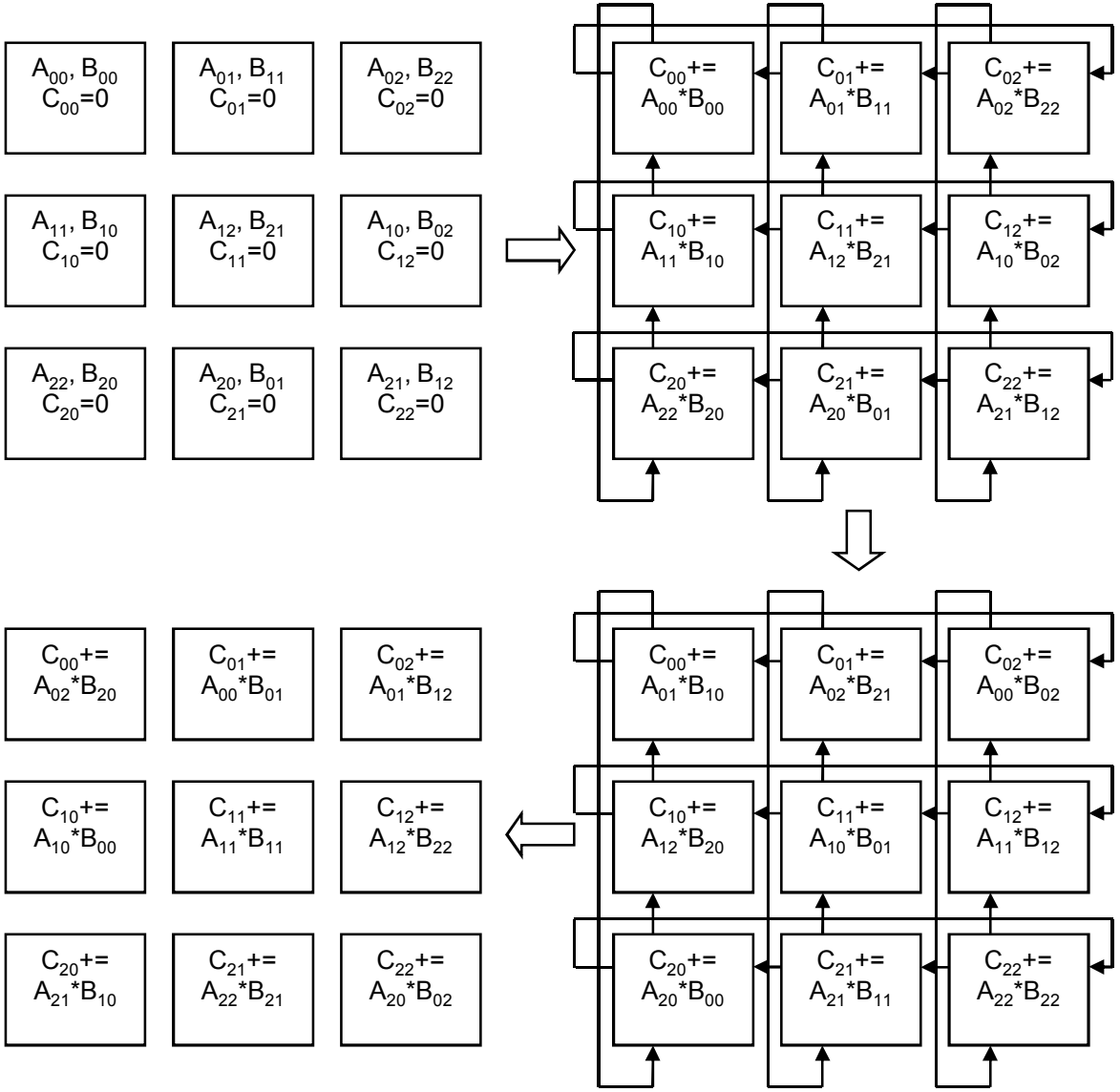
6. Τα ανωτέρω βήματα 4 & 5 επαναλαμβάνονται άλλες *n-2* φορές (συνολικά δηλαδή *n-1* επαναλήψεις των βημάτων αυτών - ή αλλιώς $\sqrt{p}$*-1 καθώς* $n^2=p$)), έτσι ώστε να έρθουν στην κατοχή κάθε επεξεργαστή σταδιακά όλα τα ζεύγη στοιχείων των δύο πινάκων, που απαιτούνται για τον υπολογισμό του στοιχείου αποτελέσματος $c_{i,j}$ για το οποίο είναι υπεύθυνος.

Αλγόριθμος του Cannon
Αρχική μετακίνηση γραμμών και στηλών



κάθε επεξεργαστής $P_{i,j}$ θα καταλήξει να κατέχει τα στοιχεία $a_{i,j+i}$, $b_{i+j,j}$,

Αλγόριθμος του Cannon - Βήματα υπολογισμών και επικοινωνίας



Ο Αλγόριθμος του Cannon - Επέκταση

Ο αλγόριθμος μπορεί να επεκταθεί κατάλληλα και για αριθμό επεξεργαστών μικρότερο από το πλήθος των στοιχείων των πινάκων *A* και *B* ($p<n^2$). Αρκεί να ανακαλέσουμε στη μνήμη μας τον *αναδρομικό* τρόπο έκφρασης του πολλαπλασιασμού πινάκων. Αρκεί πιο συγκεκριμένα να χωρίσουμε κάθε έναν από τους πίνακες *A* και *B* σε *p* υποπίνακες διαστάσεων $(n/\sqrt{p})x(n/\sqrt{p})$, να αναθέσουμε ένα ζεύγος σε κάθε επεξεργαστή (τους υποπίνακες $A_{i,j}$, $B_{i,j}$ στον επεξεργαστή $P_{i,j}$ | $i,j=0\ldots\sqrt{p}-1$ - όπως π.χ. φαίνεται στο σχήμα), και στη συνέχεια:

- Κάθε επεξεργαστής κατά το βήμα υπολογισμού κάθε επανάληψης, θα εκτελεί πολλαπλασιασμό και πρόσθεση υποπινάκων (μπλοκς) διαστάσεων $(n/\sqrt{p})x(n/\sqrt{p})$.
- Κάθε επεξεργαστής κατά το βήμα επικοινωνίας κάθε επανάληψης θα διακινεί προς τους γειτονικούς του επεξεργαστές ολόκληρα μπλοκς στοιχείων διαστάσεων $(n/\sqrt{p})x(n/\sqrt{p})$.
- Ο αριθμός των επαναλήψεων δε, παραμένει ο ίδιος ($\sqrt{p}$ βήματα υπολογισμών και $\sqrt{p}-1$ βήματα επικοινωνίας/ανταλλαγής μηνυμάτων).

Ο Αλγόριθμος του Cannon – Χρόνος Ολοκλήρωσης

$$\textit{Computation Time:} \qquad T_{comp} = O(n^3/p)$$

$$\textit{Communication Time:} \qquad T_{comm} = O(n^2/\sqrt{p})$$

Ο χρόνος επικοινωνίας με βάση το *πρότυπο του δικτύου* είναι ίσος με:

$$T'_{comm} = 2(\sqrt{p}-1)\, t_0 + 2r\, n^2/\sqrt{p} \approx 2\sqrt{p}\, t_0 + 2r\, n^2/\sqrt{p}$$

Όπως προκύπτει από τα παραπάνω, ο συνολικός χρόνος επικοινωνίας ο οποίος επιτυγχάνεται από τον αλγόριθμο του Cannon σε τοπολογία δυσδιάστατου πλέγματος είναι της τάξης του $O(n^2/\sqrt{p})$, δηλαδή εμφανώς καλύτερος από τον αντίστοιχο χρόνο του αλγόριθμου πολλαπλασιασμού κατά γραμμές σε τοπολογία δακτυλίου $O(n^2)$ που παρουσιάστηκε στην προηγούμενη παράγραφο.

Ο Αλγόριθμος του Fox

Η κύρια διαφοροποίησή του σε σχέση με τον αλγόριθμο του Cannon είναι ότι δεν επιτελεί αρχικά κάποια μαζική προετοιμασία (μετακίνηση) των στοιχείων των δύο πινάκων στο πλέγμα, αλλά αντίθετα προσπαθεί να κατανείμει το σύνολο των απαιτούμενων επικοινωνιών κατά τη διάρκεια των βημάτων εκτέλεσης του αλγορίθμου.

1. Επιλέγεται ένα στοιχείο $a_{i,j}$ του πίνακα *A* από κάθε γραμμή του πλέγματος. Πιο συγκεκριμένα, αρχικά επιλέγεται το στοιχείο $a_{0,0}$ από την 1η γραμμή, το στοιχείο $a_{1,1}$ από τη 2η κοκ, δηλαδή τα στοιχεία $a_{i,i}$ (*i=0…n-1*).

2. Ο επεξεργαστής κάθε γραμμής που κατέχει το στοιχείο του πίνακα *A* που επελέγη στο βήμα 1 (αρχικά δηλαδή οι επεξεργαστές $P_{i,i}$ (*i=0…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 $c_{i,j}$ για το οποίο είναι υπεύθυνος ($c_{i,j}$+=$a_{i,j+i}$x$b_{i+j,j}$).

Ο Αλγόριθμος του Fox (συνέχεια)

4. Κάθε επεξεργαστής στέλνει το στοιχείο που κατέχει από τον πίνακα *B* μία θέση πάνω, δηλαδή στον επεξεργαστή $P_{i-1,j}$

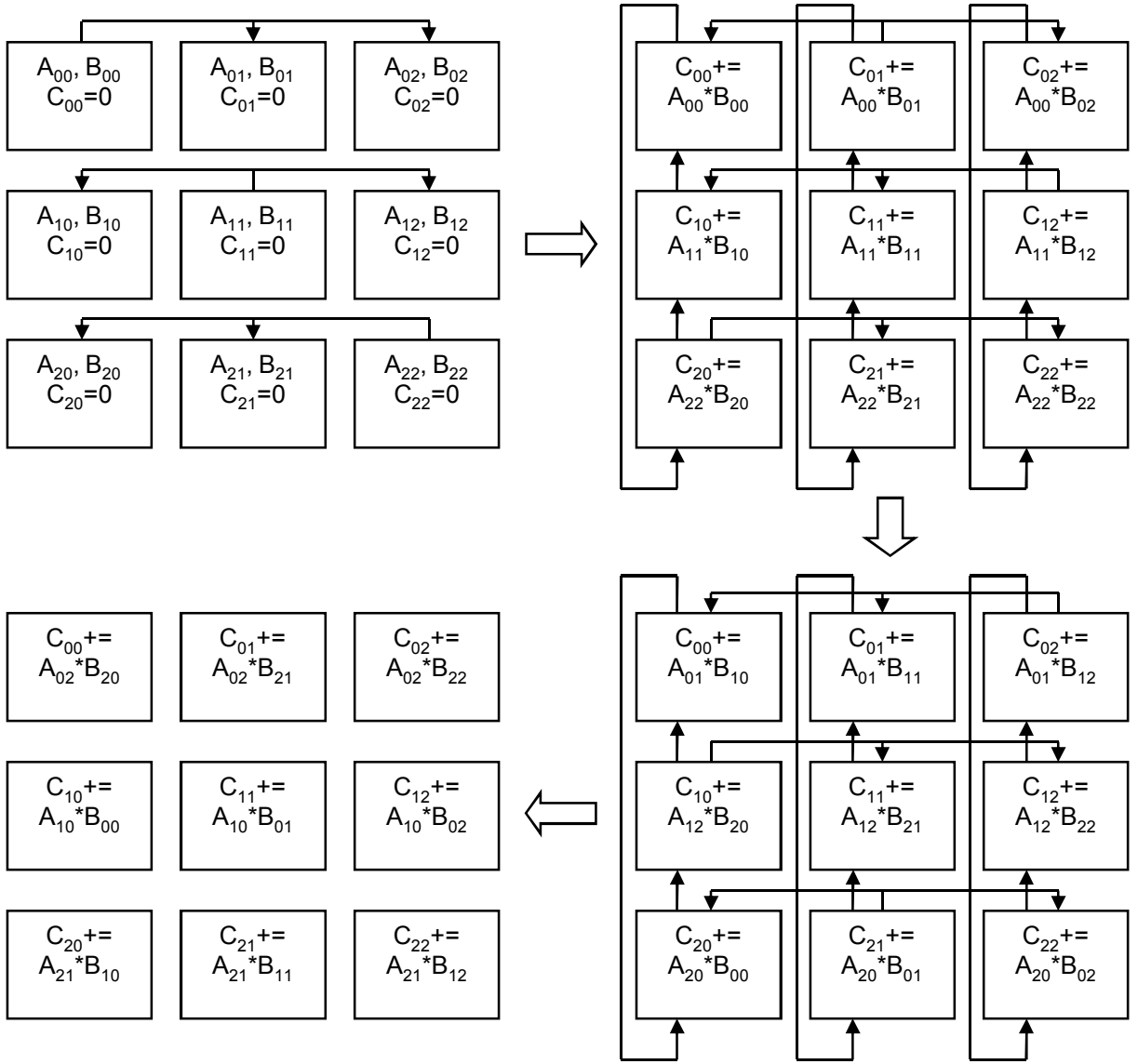
   (και κυκλικά - δηλαδή αν *i=0* στέλνει το στοιχείο στον επεξεργαστή $P_{n-1,j}$).

5. Τα ανωτέρω βήματα 1 έως 4 επαναλαμβάνονται άλλες *n-1* φορές (συνολικά δηλαδή *n* επαναλήψεις, εκ των οποίων στην τελευταία δεν είναι απαραίτητο να εκτελεστεί το βήμα 4), επιλέγοντας σε κάθε επανάληψη του βήματος 1 από κάθε γραμμή για αποστολή, το αμέσως επόμενο στη σειρά στοιχείο του πίνακα *A*.

   Πιο συγκεκριμένα σε κάθε βήμα *k* (*k=0…n-1*) του αλγορίθμου πρέπει να επιλέγεται από κάθε γραμμή του πίνακα *A* το στοιχείο $a_{i,(i+k)\%n}$.

(*) Ο αλγόριθμος μπορεί να επεκταθεί κατάλληλα και για $p<n^2$, κατά παρόμοιο τρόπο όπως και ο αλγόριθμος του Cannon.

Αλγόριθμος του Fox - Βήματα Εκτέλεσης



Ο Αλγόριθμος του Fox – Χρόνος Ολοκλήρωσης

Computation Time: $T_{comp} = O(n^3/p)$

Communication Time: $T_{comm} = O(n^2)$

Ο χρόνος επικοινωνίας με βάση το *πρότυπο του δικτύου* είναι ίσος με:

$$T'_{comm} = p\, t_0 + r\, n^2$$

Όπως προκύπτει από τα παραπάνω, ο συνολικός χρόνος επικοινωνίας ο οποίος επιτυγχάνεται από τον αλγόριθμο του Fox είναι θεωρητικά χειρότερος από τον αντίστοιχο του αλγορίθμου του Cannon (*$O(n^2)$ έναντι $O(n^2/\sqrt{p})$*).

Έχει αποδειχθεί ωστόσο ότι εφαρμόζοντας κατάλληλα την τεχνική της σωλήνωσης (ξεκινώντας π.χ. κάθε επόμενη στη σειρά προς όλους αποστολή, πριν τελειώσει η τρέχουσα - όσο το δυνατόν παράλληλα), ο χρόνος επικοινωνίας του αλγόριθμου του Fox παραμένει μεν χειρότερος αλλά φτάνει πολύ κοντά θεωρητικά σε αυτόν του αλγορίθμου του Cannon. Γίνεται πιο συγκεκριμένα ίσος με:

$$T'_{comm} = p\, t_0 + 2r\, n^2/\sqrt{p}$$

Pipeline-based Αλγόριθμος

- Στη συνέχεια παρουσιάζεται ένας αλγόριθμος ο οποίος αξιοποιεί την τεχνική της σωλήνωσης σε δύο διαστάσεις, και είναι κατάλληλος για αντίστοιχες αρχιτεκτονικές (γνωστές και ως *συστολικές διατάξεις).*

- Ο αλγόριθμος αυτός προτάθηκε μεν κατά κύριο λόγο για *συστολικές διατάξεις*, ωστόσο η λογική του μπορεί να χρησιμοποιηθεί (διαμορφωμένη κατάλληλα) και σε άλλες πιο κλασσικές τοπολογίες.

- Τα στοιχεία των δύο πινάκων εισόδου *A* και *B*, θεωρούμε ότι διοχετεύονται στοιχείο προς στοιχείο στους επεξεργαστές από τις εισόδους της *συστολικής διάταξης*, και πιο συγκεκριμένα, από τις εισόδους της οριζόντιας διάστασης θεωρούμε ότι διοχετεύονται τα στοιχεία του πίνακα *A* (η γραμμή τάξης *i* στον επεξεργαστή $P_{i,0}$ – ξεκινώντας από το στοιχείο $a_{i,0}$), ενώ από τις εισόδους της κάθετης διάστασης θεωρούμε ότι διοχετεύονται τα στοιχεία του πίνακα *B* (η στήλη τάξης *j* στον επεξεργαστή $P_{0,j}$ - ξεκινώντας από το στοιχείο $b_{0,j}$).

Pipeline-based Αλγόριθμος
(βήματα εκτέλεσης)

1. Κάθε επεξεργαστής σε κάθε βήμα παραλαμβάνει ένα στοιχείο του πίνακα *A* από τον προηγούμενό του επεξεργαστή στην οριζόντια διάσταση ($P_{i,j-1}$), και ένα στοιχείο του πίνακα *B* από τον προηγούμενό του επεξεργαστή στην κάθετη διάσταση ($P_{i-1,j}$), τα πολλαπλασιάζει και δημιουργεί κατ' αυτόν τον τρόπο έναν ακόμα όρο του αθροίσματος που απαιτείται για τον υπολογισμό του στοιχείου αποτελέσματος $c_{i,j}$ για το οποίο είναι υπεύθυνος.
2. Παράλληλα διοχετεύει το στοιχείο του πίνακα *A* που μόλις παρέλαβε στον επόμενό του επεξεργαστή στην οριζόντια διάσταση ($P_{i,j+1}$) της συστολικής διάταξης, και το στοιχείο του πίνακα *B* που μόλις παρέλαβε στον επόμενό του επεξεργαστή στην κάθετη διάσταση ($P_{i+1,j}$).
3. Όλοι οι επεξεργαστές $P_{i,j}$ εκτελούν επαναληπτικά το βασικό τους βήμα (όπως περιγράφηκε παραπάνω - σημεία 1 και 2) μέχρι να μην υπάρχουν άλλα στοιχεία της αντίστοιχης γραμμής του πίνακα *A* και της αντίστοιχης στήλης του πίνακα *B* προς παραλαβή και εκτέλεση υπολογισμών.

Pipeline-based Αλγόριθμος (συνέχεια)

- Για να διατηρηθεί ο απαιτούμενος συγχρονισμός (δηλαδή κάθε επεξεργαστής να παραλαμβάνει κάθε χρονική στιγμή το κατάλληλο επόμενο στοιχείο της αντίστοιχης γραμμής του πίνακα *A* και της αντίστοιχης στήλης του πίνακα *B*), τόσο τα στοιχεία κάθε γραμμής στην οριζόντια διάσταση όσο και τα στοιχεία κάθε στήλης στην κάθετη διάσταση, πρέπει να διοχετεύονται σταδιακά στους επεξεργαστές με αντίστοιχους κύκλους καθυστέρησης [η διοχέτευση τόσο των στοιχείων κάθε γραμμής τάξης *i* όσο και κάθε στήλης τάξης *i*, πρέπει να αρχίζει με *i* κύκλους καθυστέρησης].

- Ο αλγόριθμος μπορεί να επεκταθεί και για αριθμό επεξεργαστών μικρότερο από το πλήθος των στοιχείων των πινάκων ($p<n^2$), κατά παρόμοιο τρόπο όπως και οι αλγόριθμοι του Cannon και του Fox. Αρκεί δηλαδή να χωρίσουμε κάθε έναν από τους πίνακες *A* και *B* σε *p* υποπίνακες διαστάσεων $(n/\sqrt{p})x(n/\sqrt{p})$, και εν συνεχεία, να διοχετεύουμε πλέον από τις εισόδους της συστολικής διάταξης στους αντίστοιχους επεξεργαστές έναν-έναν τους παραπάνω υποπίνακες (αντί για μεμονωμένα στοιχεία), κλπ. Π.χ.

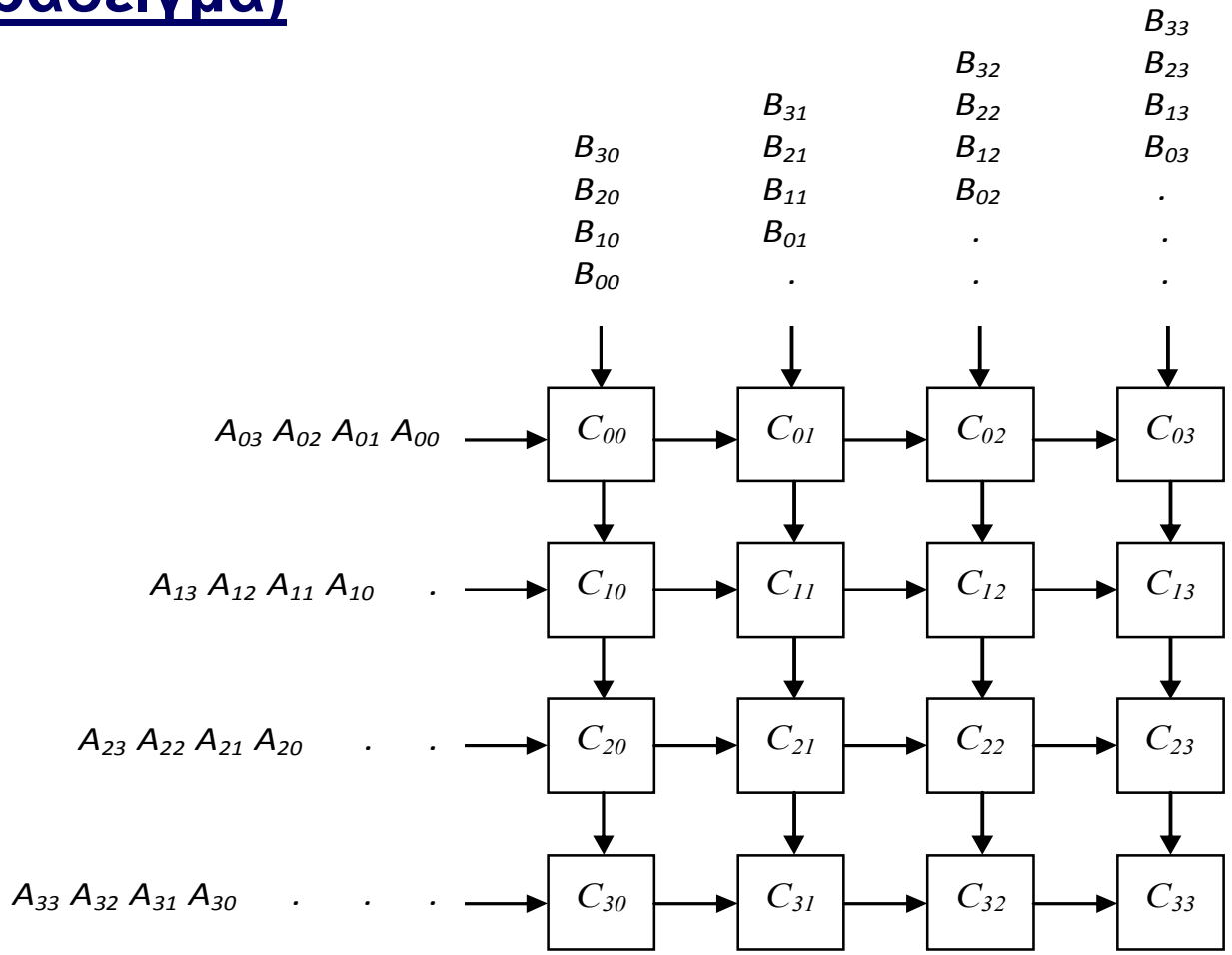
$$C_{2,3} = A_{2,0}xB_{0,3} + A_{2,1}xB_{1,3} + A_{2,2}xB_{2,3} + A_{2,3}xB_{3,3}$$

Pipeline-based Αλγόριθμος
(ψευδοκώδικας)

Σε επίπεδο ψευδοκώδικα η απαιτούμενη ανταλλαγή μηνυμάτων και υπολογισμός που εκτελεί κάθε επεξεργαστής $P_{i,j}$ σε κάθε κύκλο, από τη στιγμή που παραλαμβάνει τους πρώτους υποπίνακες των πινάκων *A* και *B*, φαίνεται παρακάτω:

```
for (k=0; k<√p-1; k++) do
    begin
        receive (A_{i,k}, left_processor);
        receive (B_{k,j}, up_processor_up);
        C_{i,j} := C_{i,j} + A_{i,k}*B_{k,j};
        send (A_{i,k}, right_processor);
        send (B_{k,j}, down_processor);
    end
```

Pipeline-based Αλγόριθμος (παράδειγμα)



Σχήμα 5.21 Πολλαπλασιασμός πινάκων σε διάταξη σωλήνωσης δύο διαστάσεων

Pipeline-based Αλγόριθμος
(Χρόνος Ολοκλήρωσης)

Ο συνολικός αριθμός δε, των απαιτούμενων κύκλων επικοινωνίας και υπολογισμών μέχρι την ολοκλήρωση του αλγορίθμου, είτε στην περίπτωση των μεμονωμένων στοιχείων ανά επεξεργαστή είτε στην περίπτωση των υποπινάκων, είναι εύκολο να διαπιστωθεί ότι είναι ίσος με $3\sqrt{p}-2$ (ή $3n-2$ στην απλή περίπτωση που έχουμε τόσους επεξεργαστές όσο και το μέγεθος του προβλήματος, δηλ. $p=n^2$).

Computation Time: $T_{comp} = O(n^3/p)$

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

Ο χρόνος επικοινωνίας με βάση το *πρότυπο του δικτύου* είναι ίσος με:

$$T'_{comm} = (3\sqrt{p}-2)\, t_0 + r\, n^2/\sqrt{p}$$

Απαιτεί μεν περισσότερους κύκλους υπολογισμού και επικοινωνιών σε απόλυτους αριθμούς (καθώς από τη φύση του ως αλγόριθμος σωλήνωσης για αντίστοιχες διατάξεις, δεν είναι συμμετρικός ως προς το φόρτο των επεξεργαστών), είναι ωστόσο σε τάξη πολυπλοκότητας ισοδύναμος με τον αλγόριθμο του Cannon.