

1. Επίλυση Συστημάτων Γραμμικών Εξισώσεων

Η ενότητα αυτή ασχολείται με το πρόβλημα της επίλυσης συστημάτων γραμμικών εξισώσεων της μορφής:

$$\left. \begin{aligned} a_{0,0}x_0 + a_{0,1}x_1 + \dots + a_{0,n-1}x_{n-1} &= b_0 \\ a_{1,0}x_0 + a_{1,1}x_1 + \dots + a_{1,n-1}x_{n-1} &= b_1 \\ \vdots &\vdots \\ a_{n-1,0}x_0 + a_{n-1,1}x_1 + \dots + a_{n-1,n-1}x_{n-1} &= b_{n-1} \end{aligned} \right\} \Leftrightarrow Ax = b$$

Εδώ ο A είναι ένας $n \times n$ πίνακας συντελεστών τέτοιος ώστε $A[i,j] = a_{i,j}$, b είναι ένα $n \times 1$ διάνυσμα $[b_0, b_1, \dots, b_{n-1}]^T$, και x είναι το διάνυσμα της επιθυμητής λύσης $[x_0, x_1, \dots, x_{n-1}]^T$. Στη συνέχεια θα γράφουμε $a_{i,j}$ αντί $A[i,j]$ και x_i αντί $x[i]$.

Ενα σύστημα εξισώσεων $Ax=b$ συνήθως λύνεται σε 2 φάσεις. Αρχικά, μέσω μιας σειράς αλγεβρικών χειρισμών, το αρχικό σύστημα εξισώσεων μετατρέπεται σε ένα άνω τριγωνικό σύστημα της μορφής

$$\begin{aligned} x_0 + u_{0,1}x_1 + u_{0,2}x_2 + \dots + u_{0,n-1}x_{n-1} &= y_0 \\ x_1 + u_{1,2}x_2 + \dots + u_{1,n-1}x_{n-1} &= y_1 \\ \vdots &\vdots \\ x_{n-1} &= y_{n-1}. \end{aligned}$$

Γράφουμε αυτό σαν $Ux=y$, όπου U είναι ένα τριγωνικό πίνακα με μονάδες στη διαγώνιο. Δηλ. $U[i,j]=0$, αν $i>j$, διαφορετικά $U[i,j]=u_{i,j}$. Επιπλέον, $U[i,i]=1$, for $0 \leq i < n$.

Στη 2η φάση επίλυσης του συστήματος γραμμικών εξισώσεων το άνω τριγωνικό σύστημα λύνεται με προς τα πίσω αντικατάσταση από το $x[n-1]$ στο $x[0]$ με τη συνάρτηση BACK-SUBSTITUTION.

```

procedure BACK-SUBSTITUTION( $U, x, y$ )  (σειριακός αλγόριθμος).
  for  $k=n-1$  downto  $0$  do /* Main loop */
     $x[k] = y[k]$ 
    for  $i=k-1$  downto  $0$  do
       $y[i] = y[i] - x[k] \times U[i,k]$ 
    end for
  end for

```

↑ ^{end for} Πρόγραμμα 5.5

: Σειριακός αλγόριθμος προς τα πίσω αντικατάσταση

Στη συνέχεια μελετάμε παράλληλες μορφές του Gaussian-

Elimination μεθόδου για την άνω τριγωνοποίηση, στις

ενότητες 1.1 και 1.2. Στην ενότητα 1.1 περιγράφουμε ένα απλό

(straight-forward) Gaussian elimination αλγόριθμο υποθέτοντας ότι

ο πίνακας συντελεστών είναι μη ιδιάζων, και ότι οι γραμμές και οι

στήλες μετακρίνονται με τέτοιο τρόπο ώστε ο αλγόριθμος να είναι

αριθμητικά σταθερός. Στην ενότητα 1.2 μελετάμε την περίπτωση

που μία αριθμητικά σταθερή λύση του συστήματος των εξισώσεων

απαιτεί μεταθέσεις των στηλών του πίνακα κατά τη διαδρομή

επιτέλεσης του αλγορίθμου.

Αν και συγκατα την Gaussian Elimination σε σχέση με την άνω τριγωνοποίηση, μια παρόμοια συνάρτηση μπορεί να χρησιμοποιηθεί για την παραγοντοποίηση του πίνακα A σε γινόμενο του κάτω τριγωνικού πίνακα L και του άνω τριγωνικού πίνακα U με μονάδες στη διαγώνιο, έτσι ώστε $A = LU$, δηλ. συγκατα την LU παραγοντοποίηση. Η LU είναι πολύ χρήσιμη αν έχουμε πολλά συστήματα εξισώσεων με το ίδιο πίνακα A .

```

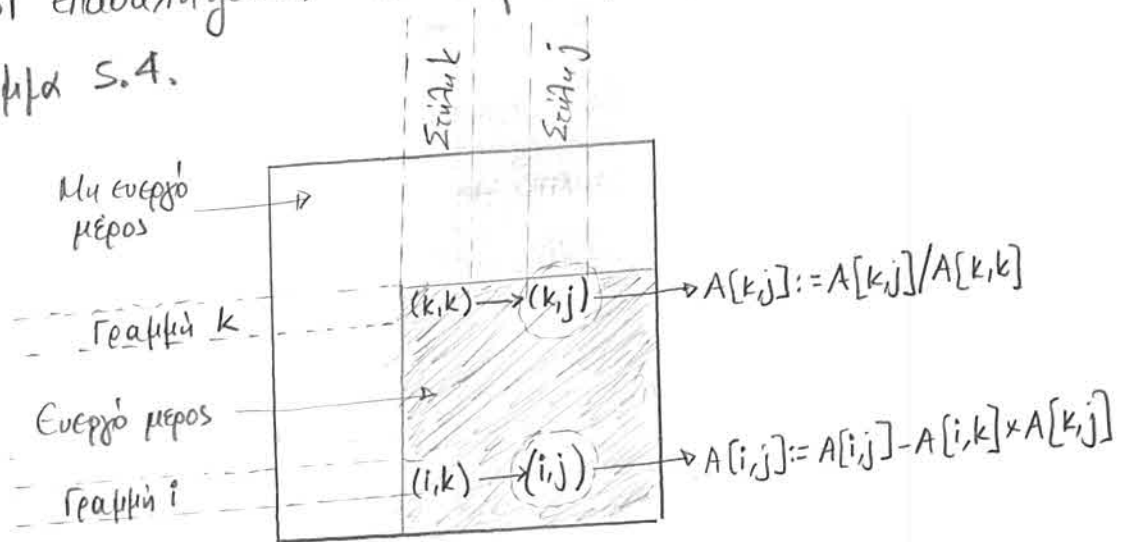
1. procedure GAUSSIAN-ELIMINATION (A, b, y)
2. begin
3.   for k:=0 to n-1 do /* Outer loop */
4.     begin
5.       for j:=k+1 to n-1 do
6.          $A[k, j] := A[k, j] / A[k, k];$  /* Division step */
7.        $y[k] := b[k] / A[k, k];$ 
8.        $A[k, k] := 1;$ 
9.       for i:=k+1 to n-1 do
10.        begin
11.          for j:=k+1 to n-1 do
12.             $A[i, j] := A[i, j] - A[i, k] \times A[k, j];$  /* Elimination step */
13.           $b[i] := b[i] - A[i, k] \times y[k];$ 
14.           $A[i, k] := 0;$ 
15.        end for /* Line 9 */
16.      end for /* Line 3 */
17. end GAUSSIAN-ELIMINATION

```

↑ Πρόγραμμα 3.4 : Σειριακός αλγόριθμος ο οποίος μετατρέπει το $Ax = y$ στο $Ux = y$ όπου θεωρούμε ότι $A[k, k] \neq 0$ ως γραμμές 6, 7.

1.1. Ένας απλός Gaussiana Elimination Αλγόριθμος

Ο απλοποιημένος Gaussiana Elimination αλγόριθμος έχει τρεις εμφωλευμένες επαναλήψεις. Υπάρχουν διάφορες παραλλαγές του αλγορίθμου, που εξαρτώνται από τη διάταξη με την οποία είναι στη σειρά οι επαναλήψεις. Θα ασχοληθούμε με τη μορφή που φαίνεται στο Πρόγραμμα 5.4.



Σχ 5.13: Ένας τυπικός υπολογισμός του Gaussiana Elimination.

Το πρόγραμμα μετατρέπει ένα σύστημα γραμμικών εξισώσεων $Ax=b$ στο άνω τριγωνικό σύστημα (με βροαδική στη διαγώνιο) $Ux=y$. Θεωρούμε ότι ο πίνακας U μοιράζεται τον αποθηκευτικό χώρο με τον A και γράφει πάνω του το άνω τριγωνικό μέρος του A . Το στοιχείο $A[k,j]$ που υπολογίζεται στη γραμμή 6 του προγράμματος 5.4 είναι ουσιαστικά το $U[k,j]$. Όμοια, το στοιχείο $A[k,k]$ που υπολογίζεται με 1 στη γραμμή 8 είναι το $U[k,k]$. Το πρόγραμμα 5.4 υποδέχει ότι το $A[k,k] \neq 0$ όταν χρησιμοποιείται στο διαίρεμα στις γραμμές 6 και 8. Αν τα βήματα στις γραμμές 7, 8, 13 και 14 δεν εκτελεστούν τότε το πρόγραμμα 5.4 οδηγεί στην LU παραγοντοποίηση

Η Gaussian Elimination κάνει περίπου $n^2/2$ διαιρέσεις (γραφή 6) και περίπου $(n^3/3) - (n^2/2)$ αφαιρέσεις και πολλαπλασιασμούς (γραφή 12). Υποθέτουμε ότι κάθε αριθμητική πράξη δέλει μία χρονική μονάδα. Άρα ο αμοιβαίος χρόνος εκτέλεσης είναι περίπου $2n^3/3$ (για μεγάλα n). (*1)

Παράλληλη υλοποίηση με διαχωρισμό σε ορίζοντες/κατακόρυφες λωρίδες

Ο πίνακας συντελεστών A κατανέμεται κατά ^{ορίζοντες} λωρίδες μεταξύ των επεξεργαστών. Παρόμοια υλοποίηση είναι και ο διαχωρισμός του A σε κατακόρυφες λωρίδες.

Έστω ότι μία γραμμή του A εκχωρείται σε κάθε επεξεργαστή και ο n ης πίνακας συντελεστών A έχει μοιραστεί στον n επεξεργαστή P_0 ως P_{n-1} . Με αυτή την απεικόνιση ο P_i αρχικά αποθηκεύει τα στοιχεία $A[i, j]$ για $0 \leq j < n$. Το σχ. 5.14 δείχνει αυτή την απεικόνιση του πίνακα στον επεξεργαστή για $n=8$. Φαίνονται επίσης τα βήματα υπολογισμού και επιμοιωνίας ως επαναλήψεις του ερωτησιακού βρόχου για $k=3$.

Το Πρόγραμμα 5.4 και το σχημα 5.13 δείχνουν ότι τα $A[k, k+1], A[k, k+2], \dots, A[k, n-1]$ διαιρούνται με το $A[k, k]$ (γραφή 6) στην αρχή της k επανάληψης. Όλα τα στοιχεία του πίνακα που συρρικνώνουν β' αυτή τη λειτουργία (φαίνονται γραφροσυσταγμένα στον πίνακα του σχ. 5.14(a)) βρίσκονται στον ίδιο επεξεργαστή. Επομένως, αυτό το βήμα δεν απαιτεί καμιά επιμοιωνία. Στο 2ο βήμα υπολογισμού του αλγορίθμου (το βήμα ανατοφής στη γραφή 12), τα τροποποιημένα (μετά τη διαίρεση

στοιχεία της k γραμμής χρησιμοποιούνται από όλες τις άλλες γραμμές του ενεργού μέρους του πίνακα. Στο σχ. 5.14(b) φαίνεται ότι αυτή η διαδικασία απαιτεί μια ενα-προς-όλους broadcast επιμοιωνυμία του ενεργού μέρους της k γραμμής με τους επεξεργαστές που έχουν αποδοκιμωμένες τις γραμμές $k+1$ ως $n-1$. Τελικά, ο υπολογισμός $A[i,j] := A[i,j] - A[i,k] \times A[k,j]$ πραγματοποιείται στο υπόλοιπο ενεργό μέρος του πίνακα, το οποίο φαίνεται γραμμικοποιημένο στο σχ. 5.14(c).

Το βήμα υπολογισμού που αντιστοιχεί στο σχ. 5.14(a) είναι k επανάληψη απαιτεί $n-k-1$ διαιρέσεις στον επεξεργαστή P_k . Όμοια, το βήμα υπολογισμού του σχ. 5.14(c) κάνει $n-k-1$ πολλαπλασιασμούς και αφαιρέσεις στην k οστή επανάληψη σε όλους τους επεξεργαστές P_i , όπου $k < i < n$.

Θεωρώντας ότι μία απλή αριθμητική πράξη θέλει 1 χρονική μονάδα, ο συνολικός χρόνος που καταναλώνεται σε υπολογισμούς της k οστής επανάληψης είναι $3(n-k-1)$. Να σημειωθεί ότι όταν ο P_k κάνει τις διαιρέσεις, οι υπόλοιποι $n-1$ επεξεργαστές είναι ανενεργοί, και όσο οι επεξεργαστές $P_{k+1}, \dots, P_{n-1}$ εκτελούν το βήμα απαλοιφής, οι επεξεργαστές $P_0, \dots, P_k$ είναι ανενεργοί. Επομένως, ο συνολικός χρόνος που καταναλώνεται στα βήματα υπολογισμών φαίνεται στα σχ. 5.14(x) και (c), και είναι

$$T_{\text{comp}} = 3 \sum_{k=0}^{n-1} (n-k-1) = 3n(n-1)/2.$$

Επίσης, το βήμα επιμοιωνυμίας του σχ. 5.14(b) για τον υπερκύβο, κοστίζει $(t_s + t_w(n-k-1)) \log n$ χρόνο. Επομένως

$$T_{\text{comm}} = \sum_{k=0}^{n-1} (t_s + t_w(n-k-1)) \log n = t_s n \log n + t_w \frac{n(n-1)}{2} \log n \quad (*)$$

Αρα,

$$T_p = \frac{3}{2}n(n-1) + t_s n \log n + \frac{1}{2}t_w n(n-1) \log n.$$

Αφού ο αριθμός των επεξεργασιών είναι n , το κόστος είναι $\Theta(n^3 \log n)$ (λόγω του όρου που σχετίζεται με το t_w στην εξίσωση (*))

Το κόστος αυτό είναι ασυμπτωτικά μεγαλύτερο από τον ασυμπτωτικό χρόνο εκτέλεσης (εξίσωση (*1)). Επομένως, αυτή η παράλληλη υλοποίηση δεν είναι βέλτερου κόστους.

(a) Computation: (i) $A[k,j] = A[k,i] / A[k,k]$, $k \leq j < n$
(ii) $A[k,k] = 1$

P_0	1	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
P_1	0	1	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
P_2	0	0	1	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
P_3	0	0	0	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
P_4	0	0	0	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
P_5	0	0	0	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
P_6	0	0	0	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
P_7	0	0	0	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

(b) Communication: One to all broadcast of row $A[k,*]$

P_0	1	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
P_1	0	1	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
P_2	0	0	1	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
P_3	0	0	0	1	(3,4)	(3,5)	(3,6)	(3,7)
P_4	0	0	0	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
P_5	0	0	0	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
P_6	0	0	0	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
P_7	0	0	0	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

P_0	1	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
P_1	0	1	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
P_2	0	0	1	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
P_3	0	0	0	1	(3,4)	(3,5)	(3,6)	(3,7)
P_4	0	0	0	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
P_5	0	0	0	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
P_6	0	0	0	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
P_7	0	0	0	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

(c) Computation

(i) $A[i,j] = A[i,j] - A[i,k] \times A[k,j]$
for $k \leq i < n$ and $k < j < n$

(ii) $A[i,k] = 0$, for $k < i < n$

Υπολογισμός και Εισαγωγή με την Τεχνική της Σωλήνωσης

Παρουσιάζουμε τώρα μία παράλληλη υλοποίηση της ΓΕ η οποία είναι βέλτερου κόστους σε έναν γραμμικό πίνακα με n επεξεργαστές, σε μέγεθος n σε υπερκύβο για $n \times n$ πίνακα συντελεστών.

Στον παράλληλο αλγόριθμο της ΓΕ που μόλις παρουσιάστηκε, οι επαναλήψεις του εξωτερικού βρόχου του προγράμματος 5.4 εκτελούνται μοιραστικά. Κάθε βύσμα, όλοι οι επεξεργαστές δουλεύουν στην ίδια παράλληλη. Η $(k+1)^{\text{οστή}}$ επανάληψη αρχίζει μόνο αφού όλοι οι υπολογισμοί και οι επικοινωνίες της $k^{\text{οστή}}$ επανάληψης ολοκληρωθούν. Η αποδοτικότητα του αλγορίθμου μπορεί να βελτιωθεί συστηματικά αν οι επεξεργαστές δουλεύουν ασύγχρονα, δηλ. καμείς επεξεργαστές δεν περιμένει τους άλλους να ολοκληρώσουν την επανάληψη, για να ξεκινήσει την επόμενη.

Αυτή ονομάζεται ασύγχρονη ΓΕ ή ΓΕ με την τεχνική της σωλήνωσης.

Στο βχ. 5.15 φαίνεται το πρόγραμμα 5.4 με την τεχνική της σωλήνωσης για ένα πίνακα 5×5 χωρισμένο σε τμήματα σε ένα λογικό γραμμικό πίνακα 5 επεξεργαστών.

Κατά την $k^{\text{οστή}}$ επανάληψη του Προγράμματος 5.4, ο επεξεργαστής P_k στέλνει με broadcast μέρος της $k^{\text{οστή}}$ γραμμής του πίνακα στους επεξεργαστές $P_{k+1}, \dots, P_{n-1}$ (βχ. 5.14(b)). Υποθέτουμε ότι όλοι οι επεξεργαστές συνδέονται με ένα γραμμικό πίνακα και ο P_{k+1} είναι ο πρώτος επεξεργαστής που λαμβάνει την $k^{\text{οστή}}$ γραμμή από τον επεξεργαστή P_k . Τότε, ο επεξεργαστής P_{k+1} πρέπει να προωθήσει τα δεδομένα στον P_{k+2} . Όμως, αφού προωθήσει την $k^{\text{οστή}}$ γραμμή

στον P_{k+2} , ο επεξεργαστής P_{k+1} δεν χρειάζεται να περιμένει να εκτελέσει το βήμα αναλοφής (γραμμή 12) μέχρι όλοι οι επεξεργαστές μέχρι του P_{k-1} να έχουν λάβει την $k^{οστή}$ γραμμή. Όμοια, ο P_{k+2} μπορεί να αρχίσει τους υπολογισμούς του αμέσως μόλις προωθήσει την $k^{οστή}$ γραμμή στον P_{k+3} , κ.ο.κ. Στο μεταξύ αφού ευθυγραμμεί ο υπολογισμός στην $k^{οστή}$ επανάληψη, ο P_{k+1} μπορεί να εκτελέσει το βήμα διαίρεσης (γραμμή 6) και να αρχίσει την broadcast μετάδοση της $(k+1)^{οστής}$ γραμμής δέχοντας την στον P_{k+2} . Στην ΓΕ με την τεχνική της σωλήνωσης, κάθε επεξεργαστής ανεξάρτητα εκτελεί επαναλαμβανόμενα την αυόλουδη σειρά ενεργειών μέχρι ότου να ευθυγραμμωθούν και οι η επαναλήψεις. Για λόγους απλότητας, υποθέτουμε ότι τα βήματα 1 και 2 χρειάζονται τον ίδιο χρόνο. Η υπόθεση αυτή δεν επηρεάζει το αποτέλεσμα της ανάλυσης.

- (1). Αν ένας επεξεργαστής έχει δεδομένα που προορίζονται για άλλους επεξεργαστές, δέχεται αυτά τα δεδομένα στον κατάλληλο επεξεργαστή
- (2). Αν ο επεξεργαστής μπορεί να εκτελέσει κάποιους υπολογισμούς χρησιμοποιώντας τα δεδομένα που έχει, το κάνει
- (3). Διαφορετικά, ο επεξεργαστής περιμένει να λάβει δεδομένα για να τα χρησιμοποιήσει με έναν από τους παραπάνω τρόπους.

Στο σχήμα S.15 φαίνονται τα 16 βήματα της παράλληλης εκτέλεσης της ΓΕ με την τεχνική της σωλήνωσης για έναν 5×5 πίνακα χωρισμένο σε λωρίδες μεταξύ 5 επεξεργαστών. Όπως δείχνει το σχ. S.15(a), το πρώτο βήμα είναι η εκτέλεση της διαίρεσης επί 6 παρά 0, στον επεξεργαστή P_0 . Η τροποποιημένη γραφή 0 δέλεται στον P_1 (σχ. S.15(b)), ο οποίος την προωθεί στον P_2 (σχ. S.15(c)). Τώρα, ο P_1 είναι ελεύθερος να εκτελέσει το βήμα της αναλοφής χρησιμοποιώντας τη γραφή 0 (σχ. S.15(d)). Στο επόμενο βήμα (σχ. S.15(e)), ο P_2 εκτελεί το βήμα αναλοφής χρησιμοποιώντας τη γραφή 0. Στο ίδιο βήμα, ο P_1 , έχοντας τελειώσει τους υπολογισμούς του για την επανάληψη 0, ξεκινάει το βήμα της διαίρεσης για την επανάληψη 1. Σε οποιαδήποτε χρονική στιγμή διαφορετικές φάσεις της ίδιας επανάληψης μπορούν να γίνουν σε διαφορετικούς επεξεργαστές. Για παράδειγμα, στο σχ. S.15(h) ο επεξεργαστής P_2 εκτελεί το βήμα αναλοφής της επανάληψης 1 ενώ οι επεξεργαστές P_3 και P_4 ασχολούνται με την επικομνωσία για την ίδια επανάληψη. Επιπλέον, περιβόστερες από μία επανάληψη μπορεί να είναι ενεργές ταυτόχρονα σε διαφορετικούς επεξεργαστές. Για παράδειγμα, στο σχ. S.15(i) ο επεξεργαστής P_2 εκτελεί το βήμα διαίρεσης της επανάληψης 2 ενώ ο επεξεργαστής P_3 εκτελεί το βήμα αναλοφής της επανάληψης 1. Θα δείξουμε τώρα ότι σε αντίθεση με τον αλγόριθμο συγχρονισμού όπου όλοι οι επεξεργαστές δουλεύουν στην ίδια επανάληψη, την ίδια χρονική στιγμή, η ΓΕ με την τεχνική της σωλήνωσης είναι βέλτιστου μήκους. Όπως φαίνεται στο σχ. S.15, η έναρξη

διαδοχικών επαναλήψεων του εξωτερικού βρόχου του Παραδείγματος 5.4 διαχωρίζεται από ένα σταθερό αριθμό βημάτων. Ένα σύνολο η τέτοιων επαναλήψεων ζευώνει. Η τελευταία επανάληψη τροποποιεί μόνο το στοιχείο στην κάτω-δεξιά γωνία του πίνακα συντελεστών; οπότε συμπληρώνεται σε σταθερό χρόνο από την έναρξη της. Επομένως, ο συνολικός αριθμός βημάτων ολοκλήρωσης της συνάρτησης με την τεχνική της σωλήνωσης είναι $\Theta(n)$. Σε κάθε βήμα, είτε $O(n)$ στοιχεία επιμοιωνούν μεταξύ τους με άμεσα-συνδεδεμένους επεξεργαστές ή εκτελείται το βήμα της διαίρεσης στα $O(n)$ στοιχεία της γραμμής, ή εκτελείται ένα βήμα αναλοφής στα $O(n)$ στοιχεία της γραμμής. Κάθε μία από αυτές τις λειτουργίες θέλει $O(n)$ χρόνο. Επομένως, όλη η συνάρτηση αποτελείται από $\Theta(n)$ βήματα, πολυπλοκότητας $O(n)$ το καθένα και ο παράλληλος χρόνος εκτέλεσης είναι $O(n^2)$. Αφού χρησιμοποιούνται η επεξεργαστές, το κόστος είναι $O(n^3)$, το οποίο είναι της ίδιας τάξης με την ακολουδιατική πολυπλοκότητα της ΓΕ. Άρα η ΓΕ που χρησιμοποιεί την τεχνική της σωλήνωσης και χωρίζει τον πίνακα συντελεστών σε λωρίδες είναι βέλτιστου κόστους σε ένα γραμμικό πίνακα επεξεργασιών. Αφού όμως ο γραμμικός πίνακας μπορεί να ερμηνευθεί μέσα σε λέξη ή υπερκύβο, ο αλγόριθμος είναι βέλτιστου κόστους και β' αυτές τις αρχιτεκτονικές.

1	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
1	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
2	(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
3	(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
4	(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

(a) Iteration k=0 starts

1	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

(b)

1	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

(c)

1	(0,1)	(0,2)	(0,3)	(0,4)	
1	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
2	(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
3	(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
4	(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

(d)

1	(0,1)	(0,2)	(0,3)	(0,4)	
0	(1,0)	(1,2)	(1,3)	(1,4)	
2	(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
3	(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
4	(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

(e) Iteration k=1 starts

1	(0,1)	(0,2)	(0,3)	(0,4)	
0	1	(1,2)	(1,3)	(1,4)	
0	(2,1)	(2,2)	(2,3)	(2,4)	
3	(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
4	(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

(f)

1	(0,1)	(0,2)	(0,3)	(0,4)
0	(1,1)	(1,2)	(1,3)	(1,4)
0	(2,1)	(2,2)	(2,3)	(2,4)
0	(3,1)	(3,2)	(3,3)	(3,4)
1	(4,1)	(4,2)	(4,3)	(4,4)

(g) Iteration k=0 ends

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	(2,1)	(2,2)	(2,3)	(2,4)
0	(3,1)	(3,2)	(3,3)	(3,4)
0	(4,1)	(4,2)	(4,3)	(4,4)

(h)

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	0	(2,2)	(2,3)	(2,4)
0	(3,1)	(3,2)	(3,3)	(3,4)
0	(4,1)	(4,2)	(4,3)	(4,4)

(i) Iteration k=2 starts

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	0	1	(2,3)	(2,4)
0	0	(3,2)	(3,3)	(3,4)
0	(4,1)	(4,2)	(4,3)	(4,4)

(j) Iteration k=1 ends

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	0	1	(2,3)	(2,4)
0	0	(3,2)	(3,3)	(3,4)
0	0	(4,2)	(4,3)	(4,4)

(k)

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	0	1	(2,3)	(2,4)
0	0	(3,2)	(3,3)	(3,4)
0	0	(4,2)	(4,3)	(4,4)

(l)

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	0	1	(2,3)	(2,4)
0	0	0	(3,3)	(3,4)
0	0	(4,2)	(4,3)	(4,4)

(m) Iteration $k=3$ starts

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	0	1	(2,3)	(2,4)
0	0	0	1	(3,4)
0	0	0	(4,3)	(4,4)

(n)

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	0	1	(2,3)	(2,4)
0	0	0	1	(3,4)
0	0	0	(4,3)	(4,4)

(o) Iteration $k=3$ ends

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	0	1	(2,3)	(2,4)
0	0	0	1	(3,4)
0	0	0	0	(4,4)

(p) Iteration $k=4$

→ Communication for $k=0,3$

→ Communication for $k=1$

→ Communication for $k=2$

□ Computation for $k=0,3$

□ Computation for $k=1,4$

□ Computation for $k=2$

Χρήση λιγότερων από n επεξεργαστές

Η προηγούμενη τεχνική της διαίτησης για την υλοποίηση της παράλληλης ΓΕ μπορεί εύκολα να προεκταθεί στην περίπτωση όπου $p < n$.

Έστω ένας $n \times n$ block πίνακας χωρισμένος σε p λωρίδες μεταξύ των p επεξεργαστών ($p < n$) ώστε σε κάθε επεξεργαστή να εικωρούνται n/p συνεχείς γραμμές του πίνακα. Στο βλ. 5.16 φαίνονται τα βήματα επιμοιουσίας μιας τυπικής επανάληψης της ΓΕ με αυτή την απεικόνιση. Όπως φαίνεται, η $k^{\text{οστή}}$ επανάληψη του αλγορίθμου απαιτεί το ενεργό μέρος της $k^{\text{οστή}}$ γραμμής να σταλεί στους επεξεργαστές που αποδιδούνται τις γραμμές $k+1, k+2, \dots, n-1$.

P_0	1	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
	0	1	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
P_1	0	0	1	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
	0	0	0	1	(3,4)	(3,5)	(3,6)	(3,7)
P_2	0	0	0	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
	0	0	0	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
P_3	0	0	0	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
	0	0	0	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

Σχ. 5.16: Η επιμοιουσία στην ΓΕ για $k=3$ για 8×8 πίνακα καταμετρήσιμο σε 4 επεξεργαστές χρησιμοποιώντας block διαχωρισμό σε p λωρίδες

Στο Σχ. 5.14(a) φαίνεται ότι, με block-striped διαγέριση, ένας επεξεργαστής με όλες τις γραμμές να αυξήσουν στο ευεργό μέρος του πίνακα εκτελεί $(n-k-1)n/p$ πολλαπλασιασμούς και αφαιρέσεις κατά το βήμα αναλοφής στην $k^{οστη}$ επανάληψη (ως τελευταίες $(n/p)-1$ επαναλήψεις, κανείς επεξεργαστής δεν έχει όλες τις γραμμές ενεργές, αλλά αγνοούμε αυτή την ανωβαθμίδα). Αν χρησιμοποιηθεί η τεχνική της σωλήνωσης, τότε το πλήθος των αριθμητικών πράξεων σε έναν επεξεργαστή με μέγιστη φόρτιση στην $k^{οστη}$ επανάληψη $(2(n-k-1)n/p)$ είναι πολύ μεγαλύτερο από το πλήθος των πράξεων επικοινωνίας $(n-k-1)$ με ένα επεξεργαστή στην ίδια επανάληψη. Επομένως, για αρκετά μεγάλες τιμές του n , σε σχέση με το p , οι υπολογισμοί υπερτερούν της επικοινωνίας σε κάθε επανάληψη. Υποθέτοντας ότι κάθε βήμας βαθμωτού πολλαπλασιασμού και αφαιρέσεως χρειάζεται 1 χρονική μονάδα, ο συνολικός παράλληλος χρόνος εκτέλεσης του αλγορίθμου (αγνοώντας το επιπλέον κόστος επικοινωνίας) είναι $2(n/p) \sum_{k=0}^{n-1} (n-k-1)$, το οποίο είναι προεξοφτικά ίσο με n^3/p .

Το κόστος = χρόνος $\times$ επεξεργαστής του αλγορίθμου είναι n^3 , ακόμη και αν τα κόστη επικοινωνίας αγνοούνται. Επομένως, το κόστος του παράλληλου αλγορίθμου είναι υψηλότερο από του ακολουδιατικού χρόνου εκτέλεσης (εξίσωση (*1)) κατά ένα παράγοντα $3/2$.

Η αναποτελεσματικότητα της ΓΕ με block διαμέριση οφείλεται στην αδρανοποίηση του επεξεργαστή που προκύπτει από την άεργη κατανομή του φορτίου. Στο Σχ. 5.17(a) φαίνεται για ένα 8×8 πινάκω και 4 επεξεργαστές, κατά τη διάρκεια της επαναλήψης του αυτετοιχεί στο $k=3$ (έξωτερικός βρόχος προγράμματος 5.4), ένας επεξεργαστής είναι απολύτως ανενεργός, ο ένας έχει μερικό φορτίο και οι άλλοι δύο είναι πλήρως ενεργοί. Όταν ολοκληρωθούν οι μισές επαναλήψεις του εξωτερικού βρόχου, μόνο οι μισοί από τους επεξεργαστές είναι πλήρως ενεργοί. Όταν ολοκληρωθούν οι μισές επαναλήψεις του εξωτερικού βρόχου, μόνο οι μισοί επεξεργαστές είναι ενεργοί. Οι υπόλοιποι αδρανείς επεξεργαστές κάνουν τελικά τον παράλληλο αλγόριθμο χειρότερου κόστους από τον ακολουδιατικό.

	1	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
P_0	0	1	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
P_1	0	0	1	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
P_2	0	0	0	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
P_3	0	0	0	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
	0	0	0	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
	0	0	0	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
	0	0	0	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

(a) Block-stripped απεικόνιση

1	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)	P_0
0	0	0	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)	
0	1	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	P_1
0	0	0	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)	
0	0	1	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)	P_2
0	0	0	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)	
0	0	0	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)	P_3
0	0	0	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)	

(b) Cyclic-stripped απεικόνιση

Σχ. 5.17: Υπολογισμός φορτίου σε διαφορετικούς επεξεργαστές με block-stripped και cyclic-stripped διαμερίση σε 8×8 πίνακα με 4 επεξεργαστές για την ΓΕ, $k=3$

Αυτό το πρόβλημα μπορεί να μετριάσει αν ο πύσσας χωριστεί κατά μήκος των επεξεργαστών χρησιμοποιώντας cyclic-stripped απεικόνιση, όπως φαίνεται στο βλ. 5.17(b). Με την cyclic-stripped διαμέριση, η διαφορά μεταξύ των υπολογιστικών φορτίων του επεξεργαστή με το μέγιστο φορτίο και του επεξεργαστή με το ελάχιστο φορτίο σε κάθε επανάληψη είναι το πολύ μία γραμμή (δηλ. $O(n)$ αριθμητικές πράξεις). Αφού υπάρχουν η επαναλήψεις, το επιπλέον κόστος λόγω ανενεργών επεξεργαστών είναι $O(n^2 p)$ με την κυκλική απεικόνιση, συγκριμένο με το $\Theta(n^3)$ της block απεικόνισης.

Παράλληλη υλοποίηση με διαχωρισμό σε τετράγωνα

Περιγράφουμε τώρα την παράλληλη υλοποίηση του προγράμματος S.4 όταν ο $n \times n$ πίνακας A απεικονίζεται σε ένα $n \times n$ πλέγμα επεξεργαστών ώστε κάθε επεξεργαστής $P_{i,j}$ αρχικά να αποθηκεύει το $A[i,j]$.

Τα βήματα επικοινωνίας και υπολογισμών της επανάληψης που αντικειμενών στον εξωτερικό βρόχο για $k=3$ φαίνονται στο σχήμα S.18 για $n=8$. Το πρόγραμμα S.4 και τα σχήματα S.13 και S.18 δείχνουν ότι στην $k^{\text{οστή}}$ επανάληψη του εξωτερικού βρόχου, το $A[k,k]$ απαιτείται από τους επεξεργαστές $P_{k,k+1}, P_{k,k+2}, \dots, P_{k,n-1}$ για τη διαίρεση με τα $A[k,k+1], A[k,k+2], \dots, A[k,n-1]$, αντίστοιχα. Μετά τη διαίρεση στη γραμμή k , τα τροποποιημένα στοιχεία της $k^{\text{οστή}}$ γραμμής χρησιμοποιούνται για την εκτέλεση του βήματος αναδοίφης από όλες τις άλλες γραμμές στο ενεργό μέρος του πίνακα. Τα τροποποιημένα (μετά τη διαίρεση στη γραμμή k) στοιχεία της $k^{\text{οστή}}$ γραμμής χρησιμοποιούνται από όλες τις άλλες γραμμές του ενεργού μέρους του πίνακα. Όμοια, τα στοιχεία της $k^{\text{οστή}}$ στήλης χρησιμοποιούνται από άλλες στήλες του ενεργού μέρους του πίνακα για το βήμα αναδοίφης. Όπως φαίνεται στο σχήμα S.18, η επικοινωνία της $k^{\text{οστή}}$ επανάληψης απαιτεί μία ένα-προς-όλους μετάδοση του $A[i,k]$ κατά μήκος της $i^{\text{οστή}}$ γραμμής (σχήμα S.18(a)) για $k \leq i < n$, και μιας ένα-προς-όλους μετάδοσης του $A[k,j]$ κατά μήκος της $j^{\text{οστή}}$ στήλης (σχήμα S.18(b)) για $k < j < n$.

Όπως και στην περίπτωση του διαχωρισμού σε λωρίδες, αν αυτές οι broadcast μεταδόσεις εκτελούνται ταυτόχρονα, ^{σε όλους τους επεξεργαστές} το αποτέλεσμα είναι μια παράλληλη φόρμουλα που δεν είναι βέλτεστη από ηθετική ούτως.

1	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
0	1	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
0	0	1	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
0	0	0	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
0	0	0	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
0	0	0	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
0	0	0	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
0	0	0	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

(a) Κατά σειρές (rowwise) broadcast μεταδοση του $A[i,k]$ για $(k-1) < i < n$.

1	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
0	1	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
0	0	1	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
0	0	0	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
0	0	0	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
0	0	0	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
0	0	0	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
0	0	0	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

(b) $A[k,j] := A[k,j] / A[k,k]$ για $k < j < n$

1	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
0	1	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
0	0	1	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
0	0	0	1	(3,4)	(3,5)	(3,6)	(3,7)
0	0	0	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
0	0	0	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
0	0	0	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
0	0	0	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

(c) Κατά στήλες (columnwise) broadcast του $A[k,j]$ για $k < j < n$

1	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
0	1	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
0	0	1	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
0	0	0	1	(3,4)	(3,5)	(3,6)	(3,7)
0	0	0	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
0	0	0	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
0	0	0	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
0	0	0	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

(d) $A[i,j] := A[i,j] - A[i,k] \times A[k,j]$
για $k < i < n$ και $k < j < n$

Σχ. 5.18: Διάφορα βήματα της GE επανάληψης που αντιστοιχούν σε $k=3$ για 8×8 πίνακα με 64 επεξεργαστές σε ένα διεδδιάστατο πλέγμα

Επιμοιωνία και υπολογισμός με την τεχνική της βωλήνωσης

Όπως φαίνεται στο σχ. 5.18, στην $k^{\text{οστή}}$ επανάληψη του εξωτερικού βρόχου (γραμμές 3-16 του Προγράμματος 5.4) το $A[k,k]$ στέλνεται προς τα δεξιά από τον $P_{k,k}$ στον $P_{k,k+1}$, στον $P_{k,k+2}$, κ.ο.κ. μέχρι να φτάσει στον $P_{k,n-1}$.

Ο επεξεργαστής $P_{k,k+1}$ εκτελεί τη διαίρεση $A[k,k+1]/A[k,k]$ μόλις λάβει το $A[k,k]$ από τον $P_{k,k}$. Δεν χρειάζεται να περιμένει το $A[k,k]$ να φτάσει μέχρι τον $P_{k,n-1}$ πριν εκτελέσει τους τωρινούς υπολογισμούς. Όμοια, οποιοδήποτε επόμενος επεξεργαστής $P_{k,j}$ της $k^{\text{οστή}}$ γραμμής μπορεί να εκτελέσει τη διαίρεσή του μόλις λάβει το $A[k,k]$. Μετά την εκτέλεση της διαίρεσης, ο $A[k,j]$ είναι έτοιμος να επιμοιωνήσει προς τα κάτω στην $j^{\text{οστή}}$ στήλη. Καθώς το $A[k,j]$ μετακινείται προς τα κάτω, κάθε επεξεργαστής που περνάει είναι ελεύθερος να χρησιμοποιηθεί για υπολογισμούς. Οι επεξεργαστές στην $j^{\text{οστή}}$ στήλη δεν χρειάζεται να περιμένουν μέχρι ο $A[k,j]$ να φτάσει τον τελευταίο επεξεργαστή της στήλης. Έτσι, ο $P_{i,j}$ εκτελεί το βήμα αναδοχής $A[i,j] := A[i,j] - A[i,k] \times A[k,j]$ μόλις τα $A[i,k]$ και $A[k,j]$ είναι διαθέσιμα. Αφού κάποιοι επεξεργαστές εκτελούν τον υπολογισμό τους για μία δεδομένη επανάληψη νωρίτερα από τους άλλους επεξεργαστές, αρχίζουν να ερχάζονται νωρίτερα στις επόμενες επαναλήψεις.

Η επιμοιωνία και ο υπολογισμός με την τεχνική της βωλήνωσης μπορούν να γίνουν με διάφορους τρόπους. Παρουσιάζουμε έναν ο οποίος φαίνεται στο σχήμα 5.19.

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

(a) Iteration $k=0$ starts

1	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

(b)

1	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

(c)

1	(0,1)	(0,2)	(0,3)	(0,4)
0	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

(d)

1	(0,1)	(0,2)	(0,3)	(0,4)
0	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

(e)

1	(0,1)	(0,2)	(0,3)	(0,4)
0	(1,1)	(1,2)	(1,3)	(1,4)
0	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

(f)

1	(0,1)	(0,2)	(0,3)	(0,4)
0	(1,1)	(1,2)	(1,3)	(1,4)
0	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

(g) Iteration $k=1$ starts

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	(2,1)	(2,2)	(2,3)	(2,4)
0	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

(h)

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	(2,1)	(2,2)	(2,3)	(2,4)
0	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

(i)

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	0	(2,2)	(2,3)	(2,4)
0	(3,1)	(3,2)	(3,3)	(3,4)
0	(4,1)	(4,2)	(4,3)	(4,4)

(j)

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	0	(2,2)	(2,3)	(2,4)
0	(3,1)	(3,2)	(3,3)	(3,4)
0	(4,1)	(4,2)	(4,3)	(4,4)

(k)

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	0	(2,2)	(2,3)	(2,4)
0	0	(3,2)	(3,3)	(3,4)
0	(4,1)	(4,2)	(4,3)	(4,4)

(l)

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	0	(2,2)	(2,3)	(2,4)
0	0	(3,2)	(3,3)	(3,4)
0	(4,1)	(4,2)	(4,3)	(4,4)

(m) Iteration $k=2$ starts

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	0	1	(2,3)	(2,4)
0	0	(3,2)	(3,3)	(3,4)
0	0	(4,2)	(4,3)	(4,4)

(n)

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	0	1	(2,3)	(2,4)
0	0	(3,2)	(3,3)	(3,4)
0	0	(4,2)	(4,3)	(4,4)

(o)

1	(0,1)	(0,2)	(0,3)	(0,4)
0	1	(1,2)	(1,3)	(1,4)
0	0	1	(2,3)	(2,4)
0	0	0	(3,3)	(3,4)
0	0	(4,2)	(4,3)	(4,4)

(p) Iteration $k=0$ ends

→ Communication for $k=0$

→ Communication for $k=1$

→ Communication for $k=2$

□ Computation for $k=0$

□ Computation for $k=1$

□ Computation for $k=2$

Σχ. 5.19: GE με την τεχνική των ερωτήσεων για 5x5 πίνακα με 25 ερωτήσεις.

Στο σχήμα 5.19(a), η επανάληψη του εξωτερικού βρόχου για $k=0$ ξεκινάει στον επεξεργαστή $P_{0,0}$ όπου ο $P_{0,0}$ βάζει το $A[0,0]$ στον $P_{0,1}$. Μετά την παραλαβή του $A[0,0]$, ο $P_{0,1}$ υπολογίζει το $A[0,1] := A[0,1]/A[0,0]$ (βλ. 5.19(b)). Τώρα, ο $P_{0,1}$ προωθεί το $A[0,0]$ στον $P_{0,2}$ και επίσης βάζει το ευμερωμένο $A[0,1]$ κάτω στο $P_{1,1}$ (σχ. 5.19(c)). Την ίδια χρονιά στιγμή, ο $P_{1,0}$ βάζει το $A[1,0]$ στον $P_{1,1}$. Έχοντας λάβει τα $A[0,1]$ και $A[1,0]$, ο $P_{1,1}$ εκτελεί το βήμα απαλοιφής $A[1,1] := A[1,1] - A[1,0] \times A[0,1]$ και έχοντας λάβει το $A[0,0]$, ο $P_{0,2}$ εκτελεί το βήμα διαίρεσης $A[0,2] := A[0,2]/A[0,0]$ (σχ. 5.19(d)). Μετά από αυτό το βήμα υπολογισμού, ένα άλλο σετ επεξεργαστών (δηλ. οι επεξεργαστές $P_{0,2}$, $P_{1,1}$ και $P_{2,0}$) είναι έτοιμοι να αρχίσουν επιμοιωνία (σχ. 5.19(e)).

Όλοι οι επεξεργαστές που εκτελούν επιμοιωνία ή υπολογισμούς κατά τη διάρκεια μιας συγκεκριμένης επανάληψης βρίσκονται κατά μήκος μιας διαγώνιας γραμμής από κάτω αριστερά προς πάνω δεξιά (για παράδειγμα οι $P_{0,2}$, $P_{1,1}$ και $P_{2,0}$ εκτελούν επιμοιωνία στο σχήμα 5.19(e) και οι $P_{0,3}$, $P_{1,2}$ και $P_{2,1}$ εκτελούν υπολογισμούς στο σχήμα 5.19(f)). Καθώς προχωρά ο παράλληλος αλγόριθμος προχωρά, αυτή η διαγώνια γραμμή μετακινείται προς την κάτω δεξιά γωνία του πλέγματος του επεξεργαστή. Επομένως, ο υπολογισμός και η επιμοιωνία για κάθε επανάληψη μέσω του πλέγματος από πάνω αριστερά προς κάτω δεξιά ως "μέτωπο". Αφού το "μέτωπο" που (front)

αυτιςτοιχεί σε μία συγκεκριμένη επανάληψη περάει από έναν επεξεργαστή, είναι ελεύθερο να εκτελέσει τις επόμενες επαναλήψεις. Για παράδειγμα, στο σχήμα 5.19 (g), αφού το "μέτωπο" για $k=0$ έχει περάσει τον $P_{1,1}$, ξεκινάει η επανάληψη για $k=1$ εκτελώντας το $A[1,1]$ στον $P_{1,2}$. Αυτό αρχικοποιεί το "μέτωπο" για $k=1$, το οποίο ακολουθεί πάλι το "μέτωπο" για $k=0$.

Όμοια, ένα τρίτο μέτωπο για $k=2$ ξεκινάει στον $P_{2,2}$ (σχήμα 5.19 (u)). Έτσι, πολλαπλά "μέτωπα" που αυτοεικοιχούν σε διαφορετικές επαναλήψεις ενεργοποιούνται ταυτόχρονα.

Κάθε βήμα μιας επανάληψης, όπως η διαίρεση, η κλάση ή η μετάδοση μιας τιμής στο γειτονικό επεξεργαστή, είναι λειτουργία σταθερού χρόνου. Επομένως, το μέτωπο μετακινείται ένα βήμα πιο κοντά στην κάτω δεξιά γωνία του πίνακα σε σταθερό χρόνο (ισοδυναμεί σε δύο βήματα του σχήματος 5.19). Το "μέτωπο" για $k=0$ δέλει $\Theta(n)$ χρόνο να φτάσει στον $P_{n-1,n-1}$ μετά την αρχικοποίηση του στον $P_{0,0}$. Ο αλγόριθμος

αρχικοποιεί η μέτωπα για τις n επαναλήψεις του εξωτερικού βρόχου. Κάθε "μέτωπο" υστερεί κατά ένα βήμα από το προηγούμενο βήμα.

Αρα, το τελευταίο "μέτωπο" περνάει την κάτω-δεξιά γωνία του πίνακα σε $\Theta(n)$ βήματα μετά το πρώτο. Ο συνολικός χρόνος που έχει περάσει μεταξύ του 1^{ου} μετώπου που ξεκινάει στον $P_{0,0}$ και του τελευταίου είναι $\Theta(n)$. Η συνάρτηση είναι πλήρης αφού τα τελευταία "μέτωπα" περάσουν την κάτω δεξιά γωνία του πίνακα, οπότε ο συνολικός παράλληλος χρόνος είναι $\Theta(n)$. Αφού χρησιμοποιούνται n^2 επεξεργαστές,

Το κόστος της τεχνικής με εωλύνωση της ΓΕ είναι $\Theta(n^3)$, το οποίο είναι το ίδιο με τον ασυμπτωτικό χρόνο του αλγορίθμου. Άρα, η ΓΕ για την τεχνική της εωλύνωσης με τετράγωνα blocks είναι βέλτεστος υόδους.

Διαχωρισμός σε τετράγωνα blocks με λιγότεροι από n^2 επεξεργαστές

Θεωρούμε την περίπτωση όπου p επεξεργαστές χρησιμοποιούνται έτσι ώστε $p < n^2$ και ο πίνακας να απεικονίζεται σε ένα $\sqrt{p} \times \sqrt{p}$ πλέγμα, χρησιμοποιώντας διαχωρισμό σε τετράγωνα blocks. Στο σχήμα 5.20 φαίνεται μια τυπική παράλληλη Gaussian επανάληψη η οποία περιλαμβάνει επικοινωνία κατά γραμμές και στήλες σε $n/\sqrt{p}$ τμές. Στο σχήμα 5.21(a) φαίνεται η κατανομή του φορτίου στην απεικόνιση με τα τετράγωνα blocks για $n=8$ και $p=16$.

$\xleftarrow{\quad n \quad} \xrightarrow{\quad}$

1	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
0	1	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
0	0	1	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
0	0	0	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
0	0	0	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
0	0	0	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
0	0	0	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
0	0	0	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

$\uparrow \downarrow \sqrt{p}$

(a) Κατά γραμμές (rowwise) broadcast μετάδοση του $A[i,k]$ για $i = k:(n-1)$

$\xleftarrow{\quad n/\sqrt{p} \quad} \xrightarrow{\quad}$

1	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
0	1	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
0	0	1	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
0	0	0	1	(3,4)	(3,5)	(3,6)	(3,7)
0	0	0	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
0	0	0	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
0	0	0	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
0	0	0	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

$\uparrow \downarrow n$

(b) Κατά στήλες (columnwise) broadcast μετάδοση του $A[k,j]$ για $j = (kn):(n-1)$

Σχ. 5.20: Τα βήματα επικοινωνίας της ΓΕ επανάληψης που απαιτούν εως $k=3$ για έναν 8×8 πίνακα σε 16 επεξεργαστές ενός διεδιέκτατου πλέγματος.

1	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
0	1	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
0	0	1	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
0	0	0	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
0	0	0	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
0	0	0	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
0	0	0	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
0	0	0	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

(a) Block-checkerboard mapping

1	(0,4)	(0,1)	(0,5)	(0,2)	(0,6)	(0,3)	(0,7)
0	(4,4)	0	(4,5)	0	(4,6)	(4,3)	(4,7)
0	(1,4)	1	(1,5)	(1,2)	(1,6)	(1,3)	(1,7)
0	(5,4)	0	(5,5)	0	(5,6)	(5,3)	(5,7)
0	(2,4)	0	(2,5)	1	(2,6)	(2,3)	(2,7)
0	(6,4)	0	(6,5)	0	(6,6)	(6,3)	(6,7)
0	(3,4)	0	(3,5)	0	(3,6)	(3,3)	(3,7)
0	(7,4)	0	(7,5)	0	(7,6)	(7,3)	(7,7)

(b) Cyclic-checkerboard mapping

Σχήμα 5.21: Υπολογιστικό φορτίο σε διαφορετικούς επεξεργαστές στην block και στην cyclic-checkerboard απεικόνιση ενός 8×8 πίνακα σε 16 επεξεργαστές κατά την επανάληψη της GE που ακολουθεί με $k=3$.

Τα σχήματα 5.20 και 5.21(a) δείχνουν ότι ο επεξεργαστής που περιέχει ένα πλήρως ενεργό μέρος του πίνακα εκτελεί ή/ρ πολλαπλασιασμούς και αφαιρέσεις, και επιμοιουώνει ή/ρ λέξεις κατά μήκος της γραμμής και της στήλης του. (αδιαφορώντας για το γεγονός ότι οι τελευταίες $(n/np)-1$ επαναλήψεις, το ενεργό μέρος του πίνακα γίνεται μικρότερο από το μέγεθος ενός block, και καμείς επεξεργαστής δεν περιέχει ένα πλήρως ενεργό μέρος του πίνακα). Αν χρησιμοποιηθεί η τεχνική της σωφίνωσης στον αλγόριθμο, τότε ο αριθμός των αριθμητικών

πράξων ανά επεξεργασία ($2n^2/p$) είναι μια τάξη μεγέθους υψηλότερος από τον αριθμό των λέξεων επικοινωνίας ανά επεξεργασία (n/p) σε κάθε επανάληψη. Έτσι, για αρκετά μεγάλες τιμές του n^2 ως προς το p , η επικοινωνία σε κάθε επανάληψη κυριαρχείται από τους υπολογισμούς. Υποθέτουμε ότι κάθε βαθμωτή αριθμητική πράξη χρειάζεται 1 χρονική μονάδα, ο συνολικός παράλληλος χρόνος εκτέλεσης του αλγορίθμου (αδιαφορώντας για το κόστος επικοινωνίας) είναι $(2n^2/p) \times n$, το οποίο ισοδυναμεί με $2n^3/p$. Το κόστος = χρόνος $\times$ επεξεργασία είναι $2n^3$, το οποίο είναι ίσο με το τριπλάσιο του κόστους του βεριανού αλγορίθμου (εξίσωση (*1)). Σαν αποτέλεσμα, υπάρχει ένα άνω γράμμα του $1/3$ στην αποδοτικότητα του παράλληλου αλγορίθμου.

Όπως και στην περίπτωση της block διαμερίσεως σε λωρίδες, η αποδοτικότητά της GE με την block διαμερίση σε τετράγωνα (block-checkerboard) ^(block-stripped) φαίνεται στην αδράμια του επεξεργαστή που προκύπτει από την άνιση κατανομή του φορτίου. Στο σχήμα 5.21(a) φαίνεται το ενεργό μέρος ενός 8×8 πίνακα συντελεστών στην επανάληψη του εξωτερικού βρόχου για $k=3$ όταν ο πίνακας είναι block-checkerboard σε 16 επεξεργαστές. Όπως φαίνεται στο σχήμα, 7 από 16 επεξεργαστές είναι εντελώς ανενεργοί, 5 έχουν μικρό φορτίο και μόνο 4 είναι πλήρως ενεργοί. Μέχρι τη στιγμή που ολοκληρώνονται οι μικτές επαναλήψεις του εξωτερικού βρόχου, μόνο το $1/4$ των επεξεργασιών είναι ενεργό. Οι υπόλοιποι ανενεργοί επεξεργαστές κάνουν τον παράλληλο αλγόριθμο πολύ

πιο δαπανηρό από τον βερσιού αλγόριθμο.

Αυτό το πρόβλημα μπορεί να μεταβεί αν ο πίνακας είναι χωρισμένος σε τετράγωνα blocks (checkerboarded) με κυκλικό τρόπο, όπως φαίνεται στο σχήμα 5.21(b). Με την cyclic-checkerboarded διαμέριση, η μεγαλύτερη διαφορά στο υπολογιστικό φορτίο μεταξύ δύο οποιωνδήποτε επεξεργασιών σε κάθε επανάληψη είναι αυτή της επιμέρους της μιας γραμμής και μιας στήλης. Για παράδειγμα, στο σχήμα 5.21(b), n^2/p στοιχεία του πίνακα είναι ενεργά στον κάτω-δεξιά επεξεργασία, και $(n-1)^2/p$ στοιχεία είναι ενεργά στον πάνω αριστερά επεξεργασία. Η διαφορά στο φόρτο εργασίας μεταξύ οποιωνδήποτε δύο επεξεργασιών είναι το πολύ $\Theta(n/\sqrt{p})$ σε κάθε επανάληψη, η οποία συνεπώς έχει $\Theta(n/\sqrt{p})$ στην παραπάνω συνάρτηση. Αφού έχουμε η επανάληψη, η παραπάνω συββαρτίωση που οφείλεται στους ανενεργούς επεξεργαστές είναι μόνο $\Theta(n^2/p)$ με την κυκλική απεικόνιση σε αντίθεση με το $\Theta(n^3)$ με την block απεικόνιση.

Από τα προηγούμενα καταλήγουμε ότι η παράλλα GE με την τεχνική της σωλήνωσης για έναν $n \times n$ πίνακα δέλει $\Theta(n^3/p)$ χρόνο σε p επεξεργαστές και με τη διαμέριση σε λωρίδες και με αυτή σε τετράγωνα blocks. Η διαμέριση σε τετράγωνα μπορεί να χρησιμοποιεί περισσότερους επεξεργαστές ($O(n^2)$) από τη διαμέριση σε λωρίδες ($O(n)$) για έναν $n \times n$ πίνακα συνεπώς, η αλοποίηση με διαμέριση σε τετράγωνα είναι πιο επετάρσιμη.

3. Βρίσκοντας ρίζες μη γραμμικών εξισώσεων

Σε πολλές επιστημονικές εφαρμογές συχνά ζητάμε να βρούμε τη ρίζα μιας εξίσωσης μιας μεταβλητής, όπως

$$x^5 - x^3 + 7 = 0$$

$$\sin x - e^x = 0$$

$$x^2 - \cos x = 0.$$

Το να βρούμε η ρίζα μιας τέτοιας εξίσωσης αναλυτικά είναι συχνά αδύνατο, οπότε καταφεύγουμε σε αριθμητικούς αλγόριθμους για να πάρουμε μια προσεγγιστική λύση.

3.1: Ένας SIMD Αλγόριθμος (Μέθοδος διχοτόμησης)

Ένας γνωστός υπολογιστικός αλγόριθμος για εύρεση ρίζας είναι η "μέθοδος της διχοτόμησης". Έστω $f(x)$ μια συνεχής συνάρτηση και έστω a_0 και b_0 δύο τιμές της μεταβλητής x τέτοιες ώστε οι $f(a_0)$ και $f(b_0)$ να έχουν αντίθετα πρόσημα, δηλ.

$$f(a_0)f(b_0) < 0$$

Σε αυτή την περίπτωση είναι βέβαιο ότι υπάρχει ρίζα της f , η x τέτοια ώστε $f(x) = 0$. Τώρα, το διάστημα (a_0, b_0) χωρίζεται σε δύο, όπου υπολογίζεται το μεσαίο σημείο, δηλ. το

$$m_0 = \frac{1}{2}(a_0 + b_0).$$

Εν $f(a_0)f(m_0) < 0$, τότε το x πρέπει να βρίσκεται στο διάστημα $(a_1, b_1) = (a_0, m_0)$, διαφορετικά βρίσκεται στο διάστημα $(a_1, b_1) = (m_0, b_0)$. Αυτό συνεχίζεται επαναλαμβάνουμε τώρα τη διαδικασία στο διάστημα (a_1, b_1) . Αυτό συνεχίζεται μέχρι να λάβουμε μια αποδεκτή προσέγγιση της ρίζας x , δηλ. μέχρι για κάποιο $n \geq 0$,

$$(i) \text{abs}(b_n - a_n) < c \quad \text{ή}$$

$$(ii) \text{abs}(f(m_n)) < c'$$

όπου c και c' είναι μικροί θετικοί αριθμοί επιλεγμένοι έτσι ώστε να πετύχουμε την επιθυμητή ακρίβεια

Ο αλγόριθμος που χρησιμοποιεί το κριτήριο (i) δίνεται από την ακόλουθη συνάρτηση BISECTION. Αρχικά, $a = a_0$ και $b = b_0$. Όταν σταματήσει η εκτέλεση της συνάρτησης, έχει βρεθεί ρίζα στο (a, b) .

procedure BISECTION (f, a, b, c)

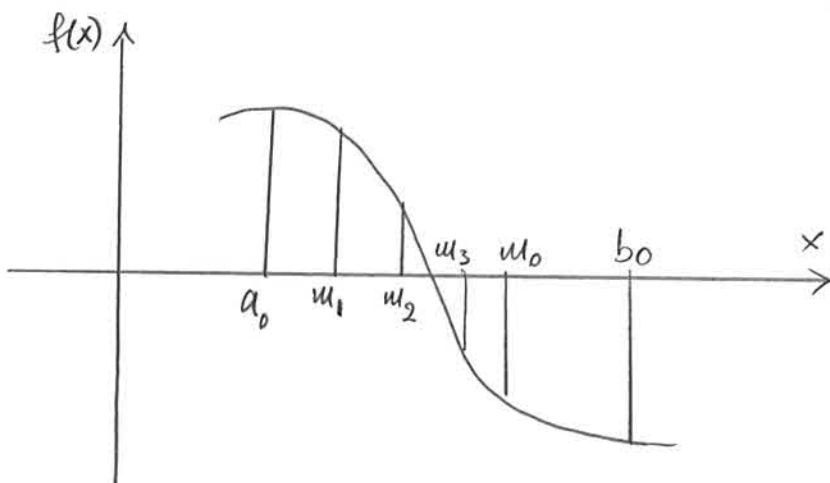
while $\text{abs}(b-a) \geq c$ do

$$(1) \quad m \leftarrow \frac{1}{2}(a+b)$$

(2) if $f(a)f(m) < 0$ then $b \leftarrow m$
else $a \leftarrow m$.

end if

end while. "



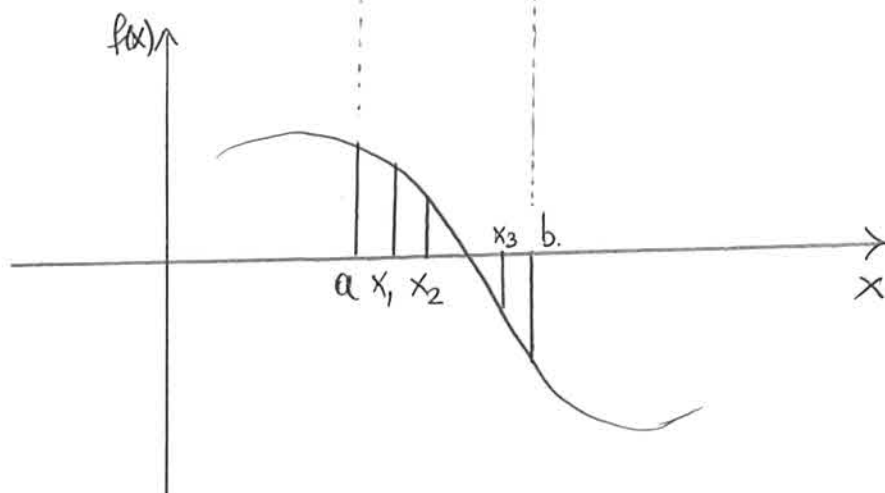
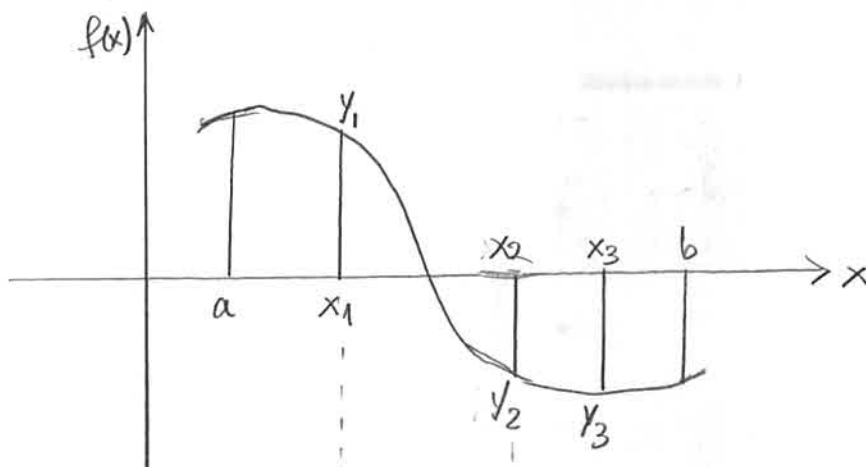
Σχ. 8.1: Ένθεση ρίζας με χρήση της συνάρτησης BISECTION

Αφού το διάστημα αναζητήσεως κόβεται στη μέση σε κάθε επανάληψη, η συνάρτηση τρέχει σε $O(\log w)$ χρόνο, όπου $w = \text{abs}(b_0 - a_0)$. Όταν f είναι διαμετρική και όχι συνεκτική, η συνάρτηση BISECTION είναι ισοδύναμη με την συνάρτηση BINARY SEARCH. Η συμπεριφορά της συνάρτησης φαίνεται στο βλ. 8.1 για κάποια συνάρτηση f . Μετά από 4 επαναλήψεις, είναι γνωστό ότι η ρίζα βρίσκεται στο διάστημα (m_2, m_3) .

(*) Η παράλληλη εκδοχή της είναι η CREW SEARCH.

Με ανάλογο τρόπο, όπως και στην BINARY SEARCH μπορούμε να υλοποιήσουμε τη συνάρτηση BISECTION σε παράλληλο υπολογιστή. Έχοντας N επεξεργαστές, η ιδέα είναι να γίνει αναζήτηση $N+1$ σημείων b' έναν CREW SM SIMD υπολογιστή. Το αρχικό διάστημα, το οποίο είναι γνωστό ότι περιέχει τη ρίζα της συνάρτησης f , χωρίζεται σε $N+1$ υποδιαστήματα ίσου μήκους. Κάθε επεξεργαστής αξιολογεί τη συνάρτηση σε κάθε ένα από τα σημεία διαίρεσης, και βασισμένος σε αυτά τις υπολογίσεις, ένα από τα υποδιαστήματα επιλέγεται για επημέλυν υποδιαίρεση. Όπως και στην ακολουθιακή περίπτωση, αυτή η διαδικασία συνεχίζεται μέχρι το διάστημα που περιέχει τη ρίζα να στενεύει στο επιθυμητό πλάτος. Ο αλγόριθμος λέγεται SIMD ROOT SEARCH. Παιρνει τη συνάρτηση f , το αρχικό διάστημα (a, b) και την ανεπιθύμητα c σαν είσοδο και επιστρέφει το διάστημα στο οποίο βρίσκεται η ρίζα της f και του οποίου το πλάτος είναι μικρότερο από το c . Η συνάρτηση σχεδιάστηκε για να τρέξει σε έναν CREW SM SIMD υπολογιστή αφού στο τέλος κάθε επανάληψης όλοι οι επεξεργαστές πρέπει να γνωρίζουν τα άκρα (a, b) του νέου διαστήματος. Χωρίς απώλεια γενικότητας,

υποθέτουμε ότι ισχύει πάντα ότι $a < b$.



Σχ. 8.2 : Εύρεση ρίζα με χρήση της συνάρτησης SIMD ROOT SEARCH.

procedure SIMD ROOT SEARCH (f, a, b, c)

while $(b-a) \geq c$ do

(1) $s \leftarrow (b-a)/(N+1)$

(2) $y_0 \leftarrow f(a)$

(3) $y_{N+1} \leftarrow f(b)$

(4) for $k=1$ to N do in parallel.

(4.1) $y_k \leftarrow f(a+ks)$

(4.2) if $y_{k-1}y_k < 0$ then (i) $a \leftarrow a+(k-1)s$
(ii) $b \leftarrow a+ks$

end if

(5) if $y_N y_{N+1} < 0$ then $a \leftarrow a+Ns$

end if

end while

Ανάλυση: Τα βήματα 1, 2, 3 και 5 μπορούν να εκτελεστούν από έναν επεξεργαστή, έστω P_N , σε σταθερό χρόνο. Στο βήμα 4, το οποίο θέλει επίσης σταθερό χρόνο, το πολύ ένας επεξεργαστής P_k θα βρει ότι $y_{k-1} y_k < 0$ και έτσι θα ευφραίνει τα a και b . Αν κανένας επεξεργαστής δεν ευφραίνει τα a και b στο βήμα 4, τότε η ρίζα πρέπει να είναι στο $(N+1)^{\text{ο}}$ διάστημα και μόνο το a ευφραίνεται στο βήμα 5. Ο αριθμός των επαναλήψεων λαμβάνεται ως εξής. Έστω w το πλάτος του αρχικού διαστήματος, δηλ. $w = b - a$. Μετά από j επαναλήψεις, το πλάτος του διαστήματος είναι $w / (N+1)^j$. Η συνάρτηση τερματίζει μόλις $w / (N+1)^j < \epsilon$. Ο αριθμός των επαναλήψεων, και επομένως ο χρόνος εκτέλεσης, της συνάρτησης SIMD ROOT SEARCH είναι $O(\log_{N+1} w)$. Το κόστος είναι $O(N \log_{N+1} w)$, το οποίο δεν είναι βέλτιστο.

Παράδειγμα:

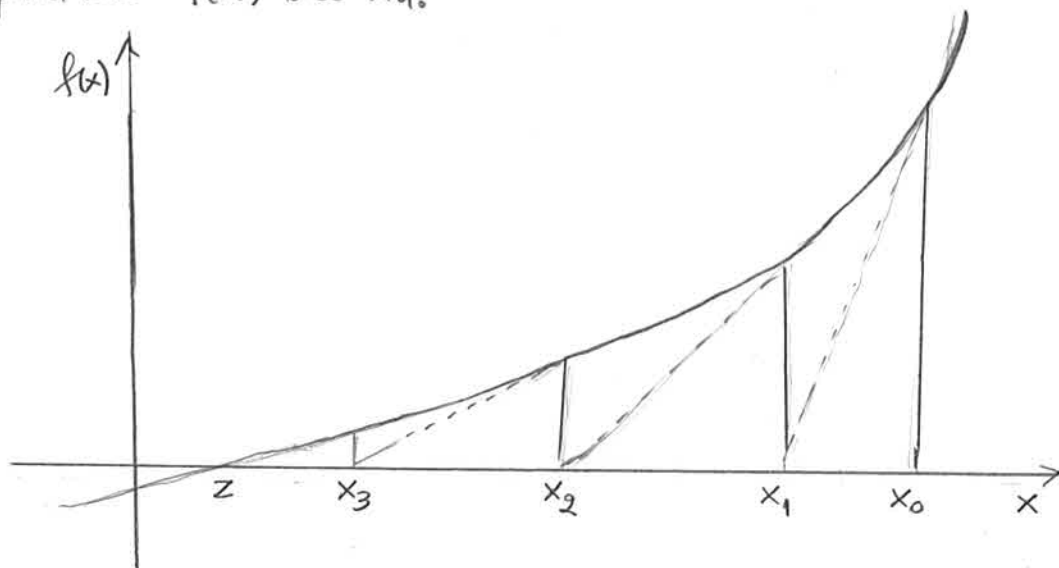
Η συμπεριφορά της συνάρτησης SIMD ROOT SEARCH της συνάρτησης του βχ. 8.1 όταν χρησιμοποιούνται 3 επεξεργαστές, φαίνεται στο βχ. 8.2. Μετά την 1η επανάληψη, το διάστημα που περιέχει τη ρίζα είναι το (x_1, x_2) όπως φαίνεται στο βχ. 8.2(a). Μετά τη 2η επανάληψη το διάστημα είναι το (x_2, x_3) , όπως φαίνεται στο βχ. 8.2(b).

3.2. Ένας ΜΙΜΔ Αλγόριθμος (μέθοδος Newton)

Ένας άλλος αυτολειτουργικός αλγόριθμος εύρεσης ρίζας ο οποίος είναι κοινά χρησιμοποιούμενος είναι η μέθοδος Newton. Δίνεται μια συνεχής και διαφορίσιμη συνάρτηση $f(x)$ μαζί με μια αρχική προσέγγιση x_0 για μια από τις ρίζες της z . Η μέθοδος υπολογίζει

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \text{ για } n=0, 1, 2, \dots$$

μέχρι $\text{abs}(x_{n+1} - x_n) < c$. Εδώ η $f'(x)$ είναι η παράγωγος της $f(x)$ και το c είναι η επιθυμητή ακριβεία. Μια γεωμετρική αναπαράσταση της μεθόδου Newton φαίνεται στο βλ. 8.3. Να σημειωθεί ότι η επόμενη προσέγγιση x_{n+1} είναι το σημείο κομής του x άξονα με την εφαπτομένη της καμπύλης $f(x)$ στο x_n .



Σχ 8.3 : Μέθοδος Newton για την εύρεση ρίζας.

Ο κυρίως λόγος για τη δυσκολότητα της μεθόδου είναι ότι συζητείται γρήγορα στο x_0 που είναι κοντά στο z . Για την απρίβεια, αν

(i) η $f(x)$ και η πρώτη και δεύτερη παραγώγος, $f'(x)$ και $f''(x)$, αντίστοιχα είναι συνεχείς σε διάστημα που περιέχει τη ρίζα z , με $f'(x) \neq 0$ και

(ii) $\text{abs}(x_0 - z) < 1$,

τότε για μεγάλο n , το $\text{abs}(x_{n+1} - z) = k(x_n - z)^2$, όπου το k είναι μια σταθερά αναλογίας η οποία εξαρτάται από το $f'(z)$ και το $f''(z)$. Με άλλα λόγια, το σφάλμα στο x_{n+1} είναι ανάλογο του τετραγώνου του σφάλματος στο x_n .

Η μέθοδος έχει τετραγωνική τάξη σύγκλισης μάκω από τις υποθέσεις που ναφέρθηκαν. Στην πράξη αυτό σημαίνει ότι ο αριθμός των σωστών ψηφίων της απάντησης διπλασιάζεται σε κάθε επανάληψη. Άρα, αν η απάντηση είναι ότι είναι απρίβιος σε n ψηφία, τότε η μέθοδος συζητίνει σε $O(\log n)$ βήματα.

Για δυσκολία με τη μέθοδο Newton είναι στην εύρεση της αρχικής προσέγγισης, η οποία είναι αρκετά κοντά στην επιθυμητή ρίζα. Αυτή η δυσκολία κεδόν απαλείφεται με την υλοποίηση της μεθόδου σ'έναν CRCW SM MIMD πολυμετή ως εξής: Αρχίζουμε με το διάστημα (a, b) , όπου $a < b$, στο οποίο είναι γνωστό ότι υπάρχει μια ρίζα z της $f(x)$. Το διάστημα χωρίζεται σε $N+1$ υποδιαστήματα ίσου μεγέθους, για κάποιο $N \geq 2$ και τα σημεία διαιρέσεως λαμβάνονται σαν αρχικές προσεγγίσεις του z . Ο υπολογιστής απαιτεί N επεξεργαστές. Κάθε διεργασία εφαρμόζει τη μέθοδο Newton ξεκινώντας με ένα από τα σημεία διαιρέσεως. Οι διεργασίες εκτελούνται

ταυτόχρονα, ωστόσο ασύγχρονα, εξαρτώμενες από τη διαδεξιμότητα των επεξεργαστών. Μόλις η διεργασία συζητήσει, το δείχνει γραφοντας την τιμή που βρήκε στις τοποθεσίες ROOT της κοινής μνήμης. Αρχικά, η τιμή της ROOT είναι ∞ . Μόλις η τιμή αυτή αλλάξει από μία διεργασία, όλες οι διεργασίες σταματούμε. Αν δύο (ή περισσότερες) διεργασίες συζητουν την ίδια χρονική στιγμή και προηαδούν να γραφουν στη ROOT ταυτόχρονα, τότε μόνον η διεργασία με τη μικρότερη αριθμική επιρρέπεια πρόβαση ενώ στις άλλες όχι. Στην περίπτωση που η διεργασία δεν συζητίνει μετά τον προκαθορισμένο αριθμό βημάτων, τότε ανατέλλεται. Ο αλγόριθμος δίνεται από τη συνάρτηση MIND ROOT SEARCH. Παίρνει σαν είσοδο τη συνάρτηση f , την παράγωγο f' , το διάστημα (a,b) , την ανεκτικότητα c και τον μέγστο επιτρεπτό αριθμό επαναλήψεων r . Επιστρέφει την απόκριση στη ROOT.

procedure MIND ROOT SEARCH ($f, f', a, b, c, r, \text{ROOT}$)

Step 1: $s \leftarrow (b-a)/(N+1)$

Step 2: for $k=1$ to N do

create process k

end for

Step 3: $\text{ROOT} \leftarrow \infty$

Step 4: Process k

(4.1) $x_{\text{old}} \leftarrow a + ks$

(4.2) iteration $\leftarrow 0$

(4.3) while (iteration $< r$) and ($\text{ROOT} = \infty$) do

(i) iteration $\leftarrow$ iteration + 1

(ii) $x_{\text{new}} \leftarrow x_{\text{old}} - f(x_{\text{old}})/f'(x_{\text{old}})$

(iii) if $\text{abs}(x_{\text{new}} - x_{\text{old}}) < c$ then $\text{ROOT} \leftarrow x_{\text{new}}$

end if

(iv) $x_{\text{old}} \leftarrow x_{\text{new}}$

end while ■

Να σημειωθεί ότι οι μεταβλητές a, s, r, c και $ROOT$ που χρησιμοποιούνται στη διεργασία k είναι καθολικές. Από την άλλη πλευρά, οι μεταβλητές επαναλήψεις, τα x_{old} και x_{new} είναι τοπικές.

Ανάλυση: Έστω ότι έχουμε διαδέξιμα N επεξεργασίες. Αν το N είναι μεγάλο, ένα από τα βήματα επιλογής θα είναι αρκετά κοντά στο z . Αν είδους οι $f(x)$, $f'(x)$ και $f''(x)$ είναι συνεχείς και οριοθετημένες στο διάστημα $[a, b]$, τότε μια από τις N διεργασίες θα συγκλίνει σε $O(\log m)$ χρόνο, που m είναι ο επιθυμητός αριθμός ακριβών γυρίων στην απόλυση.

Παράδειγμα

Έστω $f(x) = x^3 - 4x - 5$. Τότε $f'(x) = 3x^2 - 4$. Υπάρχει μια ρίζα της $f(x)$ στο διάστημα $(-3, 3)$. Έστω $N=5$, το διάστημα χωρίζεται σε 6 υποδιαστήματα με οπτικά διαμερίσματα τα $x = -2, -1, 0, 1, 2$ και οι αντίστοιχες 5 διεργασίες που διηγουρεύονται. Έστω $c = 10^{-10}$ και υποθέτουμε ότι 5 επεξεργασίες είναι διαδέξιμοι να εκτελέσουν τις 5 διεργασίες ταυτόχρονα. Σε αυτή την περίπτωση, η διεργασία 5 είναι η γρηγορότερη που συγκλίνει στη ρίζα στο 2.456678. ■