ΠΑΡΑΛΛΗΛΟΙ ΑΛΓΟΡΙΘΜΟΙ

Περιεχόμενα

Κεφάλαιο 1

Δίκτυα Επεξεργαστών

1.1 Δίκτυα σταθερών συνδέσεων

Τα προβλήματα που παρουσιάζονται στη συνέχεια αποτελούν ενδεικτικά παραδείγματα παράλληλου υπολογισμού σε δίκτυα σταθερών συνδέσεων (fixed connection networks). Σε τέτοια δίκτυα, κάθε επεξεργαστής διαθέτει τοπικό πρόγραμμα ελέγχου (local control) σχετικά απλό, χαμηλής πολυπλοκότητας και μπορεί να αποθηκεύσει τοπικά, μικρή ποσότητα δεδομένων (local storage).

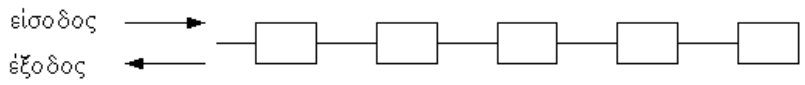
Σε κάθε χρονική μονάδα, ένας επεξεργαστής:

- δέχεται δεδομένα εισόδου,
- επιτελεί λειτουργίες που ορίζονται από το τοπικό του πρόγραμμα πάνω σε δεδομένα εισόδου και δεδομένα που έχει αποθηκευμένα,
- εξάγει αποτελέσματα προς τους γειτονικούς του επεξεργαστές και
- ανανεώνει ό,τι υπάρχει αποθηκευμένο στην τοπική του μνήμη.

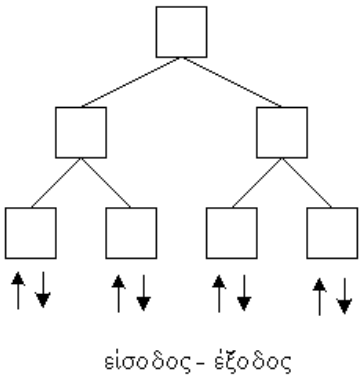
Η ύπαρξη καθολικού ρολογιού (global clock) εξασφαλίζει το συγχρονισμό, οπότε όλοι οι επεξεργαστές του δικτύου εκτελούν την παραπάνω ακολουθία λειτουργιών ταυτόχρονα. Τα γραμμικά πλέγματα επεξεργαστών (Σχήμα 1.1) και τα δέντρα (Σχήμα 1.2) αποτελούν τα απλούστερα παραδείγματα τέτοιων δικτύων.

1.2 Απόδοση παράλληλων αλγορίθμων

Η απόδοση ενός αλγορίθμου μετριέται με τους ακόλουθους τρόπους:



Σχήμα 1.1: Γραμμικό πλέγμα 5 επεξεργαστών.



Σχήμα 1.2: Δένδρο 7 επεξεργαστών.

1. Υπολογίζουμε το πόσο πιο γρήγορα λειτουργεί ο παράλληλος αλγόριθμος σε σχέση με το γρηγορότερο ακολουθιακό ή, σε κάποιες περιπτώσεις, σε σχέση με κάποιο συγκεκριμένο σειριακό αλγόριθμο. Ο αντίστοιχος όρος που μας ενδιαφέρει, λέγεται επιτάχυνση (speed up) S. Είναι:

$$S = \frac{\Gamma}{T}$$

όπου Γ ο χρόνος εκτέλεσης του γρηγορότερου ακολουθιακού αλγορίθμου και T ο χρόνος εκτέλεσης του παράλληλου αλγορίθμου.

Οι παράλληλοι αλγόριθμοι που αναπτύσσονται πρέπει να έχουν όσο το δυνατό μεγαλύτερη επιτάχυνση. Το βέλτιστο είναι ένας παράλληλος αλγόριθμος να πετυχαίνει γραμμική επιτάχυνση (linear speedup), δηλαδή αν χρησιμοποιούνται P επεξεργαστές, ο παράλληλος αλγόριθμος να τρέχει P φορές πιο γρήγορα από τον ακολουθιακό. Πρέπει να αναφερθεί ότι κανένας παράλληλος αλγόριθμος δε μπορεί να πετύχει επιτάχυνση μεγαλύτερη από τον αριθμό των επεξεργαστών ($S \leq P$). Η απόδειξη αφήνεται σαν άσκηση.

2. Μετράμε το έργο W που παράχθηκε από τον παράλληλο αλγόριθμο, δηλαδή τη συνολική επεξεργαστική ισχύ που χρειάσθηκε ο αλγόριθμος ή

απλούστερα, πόσοι επεξεργαστές είτε δεν έκαναν τίποτα είτε έκαναν μη χρήσιμο έργο κατά τη διάρκεια του υπολογισμού. Ο ορισμός του έργου W είναι

$$W = TP$$

όπου T ο χρόνος εκτέλεσης του παράλληλου αλγορίθμου και P ο αριθμός των επεξεργαστών. Επίσης, το έργο μπορεί να εκφραστεί και ως εξής:

$$W = N_1 + N_2 + ... + N_t$$

όπου N_i ο αριθμός των επεξεργαστών που είναι ενεργοί, δηλαδή εκτελούν κάποια χρήσιμη λειτουργία, κατά τη διάρκεια του i–οστού βήματος ενός αλγορίθμου που εκτελείται για t βήματα. Η έννοια του έργου μπορεί να χρησιμοποιηθεί για τον καθορισμό της αποδοτικότητας (efficiency) με την οποία χρησιμοποιούνται οι επεξεργαστές από τον παράλληλο αλγόριθμο. Συγκεκριμένα, ως αποδοτικότητα ενός παράλληλου αλγόριθμου ορίζεται ο λόγος του χρόνου που χρειάζεται για να τρέξει ο γρηγορότερος ακολουθιακός αλγόριθμος προς το έργο του παράλληλου αλγόριθμου, δηλ.

$$E = \frac{\Gamma}{W}$$

ή με βάση τον ορισμό του έργου

$$E = \frac{\Gamma}{W} = \frac{\Gamma}{TP} = \frac{S}{P}.$$

Οι καλύτεροι παράλληλοι αλγόριθμοι είναι αυτοί που είναι γρήγοροι και αποδοτικοί. Δηλαδή, επιθυμητά χαρακτηριστικά είναι:

- T όσο το δυνατό μικρότερο και
- E όσο το δυνατό πιο κοντά στο 1.

Αυτό, όμως, είναι σχετικά δύσκολο να επιτευχθεί σε πραγματικά προβλήματα μιας και διαφορετικοί παράγοντες ανάλογα με την εφαρμογή καθιστούν είτε την ταχύτητα είτε την απόδοτικότητα μείζονος σημασίας.

1.3 Περιγραφή παράλληλων αλγόριθμων

Εχοντας γνωστά τα παραπάνω γενικά στοιχεία σχετικά με παράλληλο υπολογισμό και παράλληλους αλγόριθμους, παραθέτουμε συγκεκριμένα παραδείγματα που λύνουν προβλήματα ταξινόμησης (sorting) και σύγκρισης (comparison).

1.3.1 Ταξινόμηση σε γραμμικό πλέγμα

Θεωρούμε ένα δίκτυο σταθερών συνδέσεων στην απλούστερή του μορφή, δηλ. ένα γραμμικό πλέγμα. Κάθε εσωτερικός επεξεργαστής επικοινωνεί με κάθε γείτονά του με συνδέσεις δυο κατευθύνσεων, ενώ ο πρώτος χρησιμοποιείται σαν σημείο εισόδου/εξόδου.

Το πρόβλημα: Να ταξινομηθεί μια λίστα N αριθμών με χρήση ενός γραμμικού πλέγματος με N επεξεργαστές.

Αλγόριθμος: Παραθέτουμε τη σειρά λειτουργιών που εκτελούνται σε κάθε επεξεργαστή ανεξάρτητα από τη θέση του στο πλέγμα. Ο αλγόριθμος λειτουργεί στις ακόλουθες δυο φάσεις:

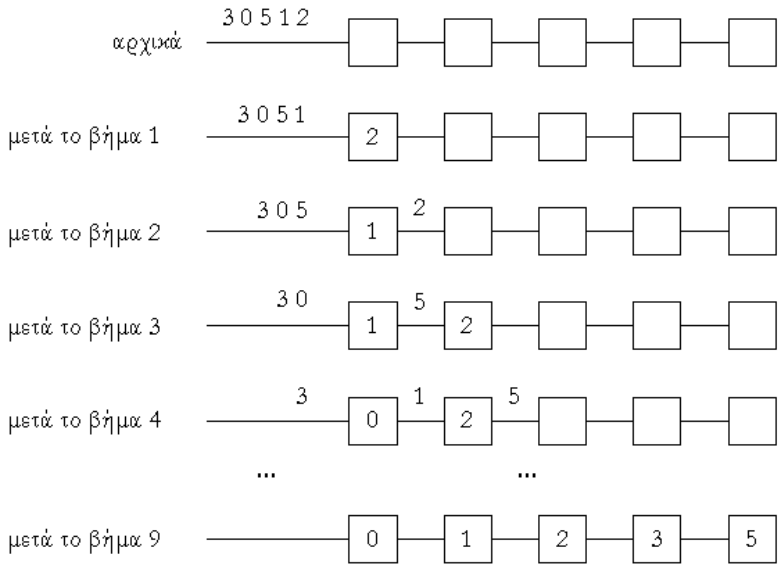
Στην πρώτη φάση (Σχήμα 1.3), κάθε επεξεργαστής:

1. παίρνει δεδομένα, (έναν αριθμό) από τον αριστερό του γείτονα,
2. συγκρίνει τα δεδομένα που δέχεται με ό,τι έχει αποθηκευμένο στην τοπική μνήμη του,
3. στέλνει στο δεξιό γείτονα τη μεγαλύτερη τιμή που προέκυψε από τη σύγκριση,
4. αποθηκεύει τοπικά τη μικρότερη τιμή.

Εύκολα μπορούμε να δείξουμε ότι μετά από $2N-1$ βήματα ο i–οστός πιο μικρός αριθμός βρίσκεται στον i–οστό επεξεργαστή του πλέγματος (μετρώντας από αριστερά) $(1 \leq i \leq N)$. Σκεφτόμαστε ως εξής: Ο αριστερότερος επεξεργαστής εξετάζει μια ακολουθία N αριθμών, κρατάει τη μικρότερη τιμή και περνάει την υπόλοιπη ακολουθία στα δεξιά. Οπότε, ο επόμενος επεξεργαστής εκτελεί τις ίδιες λειτουργίες, σε $N-1$ αριθμούς. Επαγωγικά, ο i–οστός επεξεργαστής κρατάει τη i–οστή μικρότερη τιμή και περνάει στο δεξιό του γείτονα $N-i$ μικρότερους αριθμούς. Η μεγαλύτερη τιμή φθάνει στο δεξιότερο επεξεργαστή στο βήμα $2N-1$ που είναι ο συνολικός χρόνος της φάσης.

Στη δεύτερη φάση, εξάγεται η ταξινομημένη ακολουθία των αριθμών με συγκεκριμένη σειρά. Για να γίνει αυτό υπάρχουν διάφορες ιδέες/μέθοδοι μόνο που δεν είναι δυνατό να υλοποιηθούν όλες στο πλέγμα ή ακόμα και αν υλοποιηθούν δεν είναι δυνατόν να λειτουργήσουν αποδοτικά. Έτσι:

- Μόλις ταξινομηθούν οι αριθμοί, κάθε επεξεργαστής στέλνει στον αριστερό του γείτονα ό,τι έχει αποθηκευμένο στην τοπική του μνήμη. Ομως δεν είναι δυνατό να γνωρίζει ένας εσωτερικός επεξεργαστής, εκ των



Σχήμα 1.3: Η πρώτη φάση του παλού αλγόριθμου ταξινόμησης σε γραμμικό πλέγμα.

προτέρων, πότε έχει ολοκληρωθεί η εισαγωγή της ακολουθίας των N αριθμών στο πλέγμα. Για το σκοπό αυτό απαιτείται επιπλέον υλικό (hardware), πχ. ένας μετρητής.

- Κάθε επεξεργαστής ξέρει τη θέση του χ_1 στο πλέγμα και μετράει τον αριθμό χ_2 των αντικειμένων που έχει δεχτεί σαν εισόδους. Οταν το άθροισμά τους γίνει $\chi_1 + \chi_2 = N + 1$, ο επεξεργαστής αρχίζει να στέλνει δεδομένα αριστερά. Εδώ αν και δεν απαιτείται παραπάνω υλικό, δαπανάται πολύς χρόνος: $4N - 1$ βήματα.

- Ο τελευταίος επεξεργαστής (στα δεξιά) έχει την ιδιαιτερότητα να αρχίσει να στέλνει δεδομένα αριστερά μόλις δεχτεί κάτι σαν είσοδο. Οι υπόλοιποι επεξεργαστές όταν παίρνουν κάτι από δεξιά αρχίζουν να στέλνουν αριστερά.

- Κάθε επεξεργαστής στέλνει δεδομένα αριστερά όταν δεν πάιρνει άλλα δεδομένα από τον αριστερό του γείτονα. Με τον τρόπο αυτό, χωρίς επιπλέον υλικό, απαιτούνται μόνο $3N - 1$ βήματα.

Η επιτάχυνση του παραπάνω αλγορίθμου είναι $S = \Theta(logN)$, το έργο $W = \Theta(N^2)$ και η αποδοτικότητά του $\Theta\left(\frac{\log N}{N}\right)$. Εύκολα παρατηρούμε ότι

για μεγάλες τιμές του N, δηλαδή όταν θέλουμε να συγκρίνουμε πολλούς αριθμούς, ο παραπάνω αλγόριθμος είναι μη αποδοτικός αφού το E είναι μικρό.

Είναι πάντα εύκολο να μετατραπεί ένας αλγόριθμος που έχει σχεδιαστεί για μεγάλο δίκτυο επεξεργαστών, σε έναν εξίσου αποδοτικό αλγόριθμο για ένα μικρότερο δίκτυο του οποίου οι επεξεργαστές έχουν μεγαλύτερη αποθηκευτική ικανότητα.

Παράδειγμα: Εύκολα μετατρέπεται ένας παράλληλος αλγόριθμος σε ισοδύναμο (ίδιας απόδοσης) ακολουθιακό.

Προσοχή: Το αντίστροφο δεν ισχύει πάντα. Δεν υπάρχει γενικευμένη μέθοδος μετατροπής ενός ακολουθιακού αλγορίθμου σε παράλληλο (σε ορισμένες περιπτώσεις αυτό είναι αδύνατο).

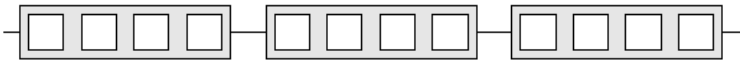
Συνήθως ο αριθμός των επεξεργαστών που χρησιμοποιούνται για τη λύση ενός προβλήματος είναι τουλάχιστο όσο το μέγεθος του προβλήματος. Στη συνέχεια θα περιγραφεί η ταξινόμηση N αριθμών με χρήση ενός γραμμικού πλέγματος με λιγότερους από N επεξεργαστές. Εννοια-κλειδί είναι η αποθηκευτική δυνατότητα των επεξεργαστών στο γραμμικό πλέγμα (granularity). Η χρησιμοποίηση λιγότερων επεξεργαστών συνεπάγεται την αύξηση της αποθηκευτικής τους ισχύος, όχι όμως απαραίτητα και τη βελτίωση της απόδοσης ενός παράλληλου αλγόριθμου που τρέχει σε αυτό το δίκτυο. Οπότε, αν έχουμε να ταξινομήσουμε N αριθμούς σε ένα γραμμικό πλέγμα P επεξεργαστών με $P < N$, θα πρέπει κάθε επεξεργαστής να μπορεί να αποθηκεύσει τουλάχιστο N/P αντικείμενα.

Η γενική μέθοδος για μια τέτοια μετατροπή είναι η εξής. Εστω ένας αλγόριθμος σχεδιασμένος για ένα δίκτυο G_1 που αποτελείται από P_1 επεξεργαστές. Θέλουμε να μετατρέψουμε αυτόν τον αλγόριθμο ώστε να τρέχει σε ένα δίκτυο G_2 με λιγότερους επεξεργαστές P_2. Μοναδική απαίτηση είναι η εξής: οι επεξεργαστές του δικτύου G_2 να έχουν μεγαλύτερη αποθηκευτική δυνατότητα από αυτούς του G_1.

Προκειμένου να επιτευχθεί η παραπάνω μετατροπή, κάθε επεξεργαστής του δικτύου G_2 εξομοιώνει $\lceil P_1/P_2 \rceil$ επεξεργαστές του δικτύου G_1. Με τον τρόπο αυτό εισάγεται μια καθυστέρηση $\lceil P_1/P_2 \rceil$ εξαιτίας της μείωσης του αριθμού των χρησιμοποιούμενων επεξεργαστών, γεγονός όμως που δεν επηρεάζει την αποδοτικότητα του τελικού αλγορίθμου για το δίκτυο G_2 σε σχέση με τον αρχικό αλγόριθμο.

Για να συγκεκριμενοποιηθεί η παραπάνω διαδικασία: θεωρούμε ένα γραμμικό πλέγμα G_1 με 12 επεξεργαστές και θέλουμε να το εξομοιώσουμε με ένα νέο G_2 με 3 επεξεργαστές (Σχήμα 1.4). Τότε, κάθε επεξεργαστής του G_2

εξομοιώνει 4 διαδοχικούς επεξεγαστές του G_1 και κατά συνέπεια, κάθε βήμα στο G_1 ισοδυναμεί με 4 βήματα στο G_2. Δηλαδή, κάθε αλγόριθμος που τρέχει σε χρόνο T στο G_1, απαιτεί χρόνο $4T$ για να τρέξει στο G_2. Γενικά, λοιπόν, μπορούμε να ταξινομήσουμε N αριθμούς σε γραμμικό πλέγμα με P επεξεργαστές σε $O(N/P)$ βήματα ($P < N$).



Σχήμα 1.4: Εξομοίωση γραμμικού πλέγματος 12 επεξεργαστών G_1 μέσω πλέγματος 3 επεξεργαστών G_2. Κάθε βήμα του G_1 ισοδυναμεί με τέσσερα βήματα του G_2 και κάθε αλγόριθμος που εκτελείται σε T βήματα στο πλέγμα G_1 εκτελείται σε $4T$ βήματα στπ πλέγμα G_2.

1.3.2 Ενα απλό πρόβλημα ταξινόμησης

Μέχρι τώρα περιγράψαμε και αναλύσαμε αλγόριθμους βασιζόμενοι στο μοντέλο λέξης (word model), δηλαδή θεωρήσαμε ότι πραγματοποιείται σύγκριση και μεταφορά, σε κάθε βήμα, ολόκληρων λέξεων (ακολουθιών από ψηφία–bits). Αυτή η πρακτική, αν και είναι αρκετά συνηθισμένη, δεν ενδείκνυται στην περίπτωση που οι λέξεις είναι μεγάλες (αποτελούνται από πολλά bits) και απαιτείται να είναι γνωστός ο αριθμός των ηλεκτρονικών στοιχείων (transistors/gates) που χρειάζονται για να κατασκευαστούν τα κυκλώματα.

Σε αυτή την παράγραφο, θα χρησιμοποιήσουμε την έννοια του bit. Ολες οι λειτουργίες ανάγονται σε λειτουργίες που πραγματοποιούνται σε bits (μοντέλο bit). Έτσι, για την ανάλυση του αλγορίθμου ταξινόμησης που προαναφέρθηκε, αναλύεται ο τρόπος που γίνονται οι συγκρίσεις ανάμεσα στα bits και όχι ανάμεσα σε λέξεις. Στη συνέχεια, θα παρουσιαστούν δύο μέθοδοι πραγματοποίησης συγκρίσεων αριθμών bit–by–bit, που διαφέρουν στον τύπο του δικτύου που χρησιμοποιείται.

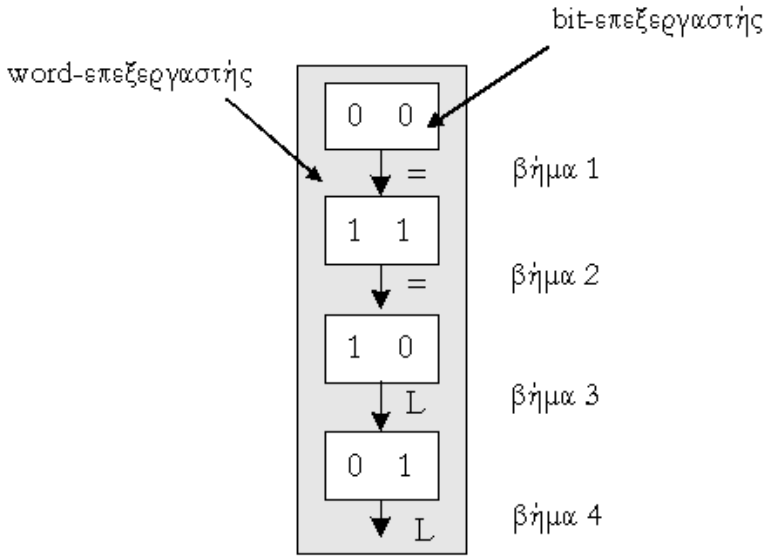
Με χρήση γραμμικού πλέγματος: Θεωρούμε δύο αριθμούς a και b των k bits ο καθένας, με δυαδικές ακολουθίες $a_1a_2...a_k$ και $b_1b_2...b_k$. Οι αριθμοί συγκρίνονται bit–by–bit ξεκινώντας από τα περισσότερο σημαντικά bits. Τα i–οστά bits των αριθμών, a_i και b_i αντίστοιχα, αποθηκεύονται και συγκρίνονται στο i–οστό επεξεργαστή του γραμμικού πλέγματος. Κατά το i–οστό βήμα, ο i–οστός επεξεργαστής:

- δέχεται από τον προηγούμενό του (τον $i-1$) μια είσοδο v, που δείχνει ποιος από τους $a_1a_2...a_{i-1}$ και $b_1b_2...b_{i-1}$ είναι μεγαλύτερος ή αν αυτοί

είναι ίσοι,

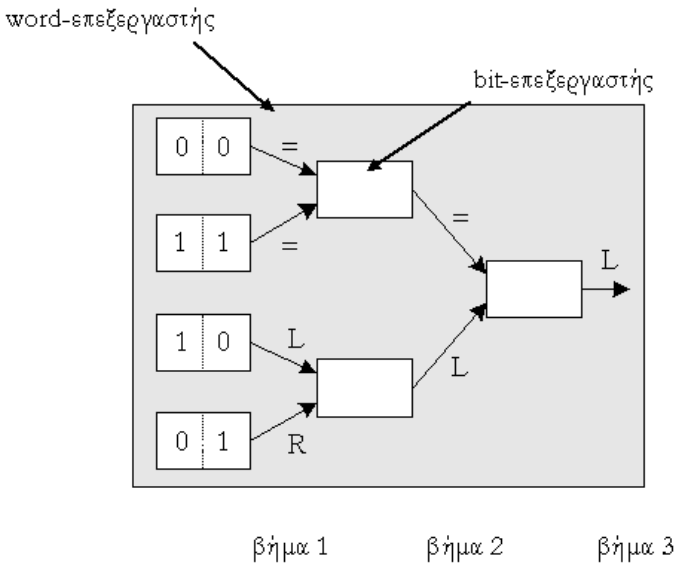
- συγκρίνει τα a_i και b_i, ανανεώνει την τιμή v με βάση το αποτέλεσμα της σύγκρισης και στέλνει στον επόμενό επεξεργαστή (τον $i+1$) την τιμή v, δείχνοντας ποιος από τους $a_1a_2...a_i$ και $b_1b_2...b_i$ είναι μεγαλύτερος (ενδέχεται να είναι και ίσοι).

Η προηγούμενη διαδικασία επαναλαμβάνεται μέχρι να συγκριθούν όλα τα αντίστοιχα bits των αριθμών. Στο σχήμα 1.5 παρουσιάζεται η Μέθοδος 1 για τους αριθμούς 0110 και 0101.



Σχήμα 1.5: Σύγκριση των αριθμών 0110 και 0101 σε γραμμικό πλέγμα με 4 επεξεργαστές.

Με χρήση Πλήρους Δυαδικού Δένδρου: Θεωρούμε, πάλι, δύο αριθμούς a και b των k bits ο καθένας, με δυαδικές ακολουθίες $a_1a_2...a_k$ και $b_1b_2...b_k$. Τα bits των αριθμών αποθηκεύονται στα φύλλα ενός πλήρους δυαδικού δένδρου και τα i–οστά bits a_i και b_i συγκρίνονται στον i–οστό επεξεργαστή του δικτύου (φύλλο του δένδρου). Το αποτέλεσμα της σύγκρισης διαδίδεται προς τα πάνω στους εσωτερικούς κόμβους. Οι λεπτομέρειες της λειτουργίας των εσωτερικών κόμβων αφήνονται στον αναγνώστη. Οταν φτάσουμε στη ρίζα του δένδρου, η διαδικασία της σύγκρισης έχει ολοκληρωθεί. Ενδεικτικό είναι το παράδειγμα στο Σχήμα 1.6, για τους αριθμούς 0110 και 0101.



Σχήμα 1.6: Σύγκριση των αριθμών 0110 και 0101 σε δυαδικό δένδρο με 4 επεξεργαστέσ-φύλλα.

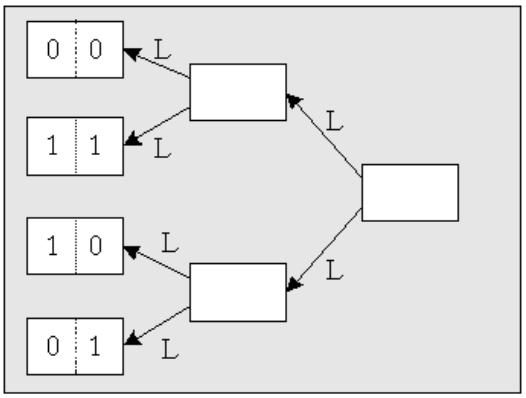
Για τη σύγκριση 2 αριθμών μεγέθους k bits, η χρησιμοποίηση του πλήρους δυαδικού δένδρου πλεονεκτεί σε σύγκριση με το γραμμικό πλέγμα για τους ακόλουθους δύο λόγους:

- Με τη δενδρική δομή απαιτούνται $\log k + 1$ βήματα, ενώ με το γραμμικό πλέγμα απαιτούνται k βήματα.

- Το πλήρες δυαδικό δένδρο είναι καταλληλότερο για την ταξινόμηση bit-by-bit μιας και συνδυαζόμενο με τη διαδικασία ταξινόμησης αριθμών (μοντέλο λέξης) την επιταχύνει.

Κατά τη bit–by–bit σύγκριση δύο αριθμών, η διαδικασία της γνωστοποίησης (Σχήμα 1.7)απαιτεί $\log k$ βήματα για ένα δυαδικό δένρο με k φύλλα. Οπότε, μετά από $2 \log k$ βήματα, η σύγκριση δύο k–bit αριθμών έχει ολοκληρωθεί και τα bits του μεγαλύτερου αριθμού μετακινούνται, σε ένα βήμα, στο δεξιότερο επεξεργαστή.

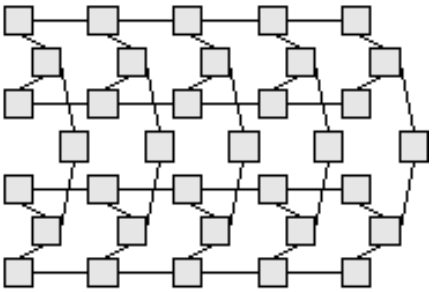
Θεωρούμε τον αλγόριθμο ταξινόμησης με χρήση γραμμικού πλέγματος που περιγράψαμε παραπάνω. Κάθε επεξεργαστής του γραμμικού πλέγματος αντικαθίσταται με ένα πλήρες δυαδικό δένδρο k–φύλλων. Οι επεξεργαστές εκτελούν λειτουργίες μεταξύ bits. Το δίκτυο που προκύπτει έχει $(2k-1)N$ επεξερ-



Σχήμα 1.7: Διαδικασία γνωστοποίησης μετά τη σύγκριση των αριθμών 0110 και 0101.

γαστές. Η εκτέλεση της πρώτης φάσης του αλγορίθμου ταξινόμησης απαιτεί $2(2N-1)\log k$ βήματα.

Τότε μπορούμε, εύκολα, να κατασκευάσουμε ένα δίκτυο με $(2k-1)N$ επεξεργαστές που απαιτεί $2(2N-1)\log k$ βήματα για να ολοκληρώσει τη Φάση 1 του αλγορίθμου ταξινόμησης. Αυτό φαίνεται και στο Σχήμα 1.8 όπου παρουσιάζεται τη δικτυακή δομή.



Σχήμα 1.8: Γραμμικό πλέγμα που αποτελείται από πλήρη δυαδικά δένδρα.

1.3.3 Βελτίωση της απόδοσης

Πέρα από την χρήση δυαδικού δένδρου, ο αλγόριθμος μπορεί να επιταχυνθεί με χρήση της τεχνικής pipeline έτσι ώστε κάθε επεξεργαστής να δουλεύει

σε παραπάνω από μια συγκρίσεις, ταυτόχρονα. Μάλιστα, αυτό επιτυγχάνεται χρησιμοποιώντας τον αλγόριθμο σύγκρισης στο γραμμικό πλέγμα και όχι στο δέντρο. Διατάσσουμε k γραμμικά πλέγματα ώστε να σχηματιστεί ένα $k \times N$ πλέγμα στο οποίο κάθε επεξεργαστής εκτελεί λειτουργίες μεταξύ bits. Με τον τρόπο αυτό, ο αλγόριθμος για ταξινόμηση N k–bit αριθμών σε γραμμικό πλέγμα επεξεργαστών ολοκληρώνει την πρώτη φάση σε $(k-1)+(2N-1) = 2N+k-2$ βήματα. Με δεδομένο ότι η δεύτερη φάση πραγματοποιείται όπως ακριβώς και στο μοντέλο λέξης, καταλήγουμε στο συμπέρασμα ότι με χρήση pipeline, η σύγκριση N k–bit αριθμών σε γραμμικό πλέγμα ολοκληρώνεται σε $3N+k-4$ βήματα.

1.4 Κάτω φράγματα

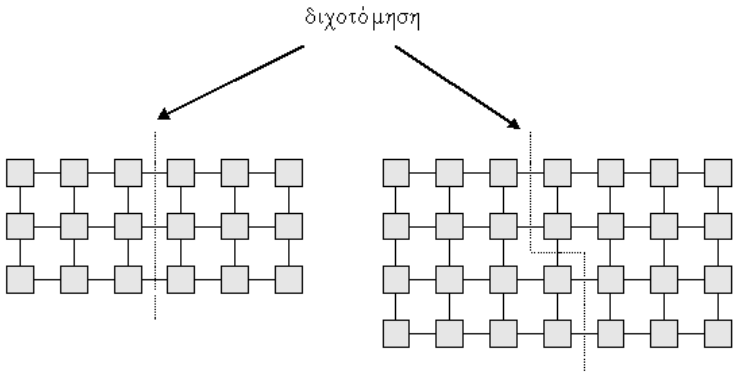
Ο αλγόριθμος για ταξινόμηση με το bit μοντέλο αποτελεί χαρακτηριστική περίπτωση όπου απλοί επεξεργαστές συνδέονται κατάλληλα προκειμένου να εκτελέσουν μια πολύπλοκη εργασία. Στόχος, βέβαια, είναι η γρήγορη, απλή και χαμηλού κόστους εκτέλεση πολύπλοκων εργασιών. Τρεις παράγοντες επηρεάζουν την απόδοση ενός δικτύου σταθερών συνδέσεων:

- Το ποσό των δεδομένων που μπορεί να εισάγεται/εξάγεται στο δίκτυο σε κάθε βήμα (I/O bandwidth).

- Η διάμετρος του δικτύου, δηλαδή, η μέγιστη απόσταση μεταξύ 2 επεξεργαστών του δικτύου. Η διάμετρος αποτελεί συνήθως και το κάτω φράγμα στο χρόνο που χρειάζεται για να εκτελεστεί ένας χρήσιμος υπολογισμός. Ο λόγος είναι ότι πληροφορία που υπολογίζεται από έναν επεξεργαστή μπορεί να χρειάζεται να χρησιμοποιηθεί από ένα μακρινό επεξεργαστή.

- Το εύρος διχοτόμησης (bisection width) του δικτύου, δηλαδή ο ελάχιστος αριθμός συνδέσεων που πρέπει να απομακρυνθούν ώστε να χωριστεί το δίκτυο σε δύο ξένα σύνολα επεξεργαστών ίσου μεγέθους (Σχήμα 1.9). Το εύρος διχοτόμησης αποτελεί καθοριστικό παράγοντα για την ταχύτητα με την οποία ένα δίκτυο μπορεί να εκτελέσει έναν υπολογισμό, αφού δεδομένα που υπάρχουν ή παράγονται στο ένα μισό ενός δικτύου μπορεί να χρειαστούν στο άλλο μισό για να ολοκληρωθεί ο υπολογισμός.

Οι παραπάνω παράγοντες αποτελούν μια βάση για το χαρακτηρισμό ενός αλγορίθμου ως βέλτιστου (τουλάχιστο για συγκεκριμένες κατηγορίες δικτύων) και είναι πολύ χρήσιμοι για τον καθορισμό οποιουδήποτε κάτω φράγματος στο χρόνο εκτέλεσης οποιουδήποτε αλγορίθμου για ένα συγκεκριμένο πρόβλημα.



Σχήμα 1.9: Ελάχιστες διχοτομήσεις για γραμμικά πλέγματα διαστάσεων 3×6 και 4×7 που αποτελείται από πλήρη δυαδικά δένδρα.

Μερικές φορές τα επιχειρήματα αυτά δεν μπορούν να χρησιμοποιηθούν (δώστε παραδείγματα).

1.5 Ασκήσεις

1. Θεωρήστε δυο αλγορίθμους που λύνουν ένα πρόβλημα μεγέθους M. Ο πρώτος εκτελείται σε M βήματα σε υπολογιστή με M επεξεργαστές και ο δεύτερος σε $\sqrt{M}$ βήματα σε υπολογιστή με M^2 επεξεργαστές. Ποιος αλγόριθμος είναι πιο αποδοτικός· Υποθέστε ότι το κόστος λειτουργίας ενός υπολογιστή με P επεξεργαστές για T βήματα είναι $T^a P^b$, όπου a και b σταθερές και $P = M$ ή $P = M^2$. Για ποιές τιμές των a,b συμφέρει να χρησιμοποιήσουμε τον πρώτο αλγόριθμο και για ποιές τον δεύτερο·

2. Θεωρήστε δυο αλγορίθμους που λύνουν ένα πρόβλημα μεγέθους M. Ο πρώτος εκτελείται σε M βήματα σε υπολογιστή με M επεξεργαστές και ο δεύτερος σε $\sqrt{M}$ βήματα σε υπολογιστή με M^2 επεξεργαστές. Ποιος αλγόριθμος θα εκτελεστεί γρηγορότερα σε έναν υπολογιστή με N επεξεργαστές· (*Υπόδειξη*: Η απάντηση εξαρτάται από τη σχέση των N και M. Χρησιμοποιήστε το γεγονός ότι ένας υπολογιστής με N επεξεργαστές μπορεί να εξομοιώσει έναν υπολογιστή με P επεξεργαστές με καθυστέρηση (slowdown) P/N.)

3. Θεωρήστε δυο αλγορίθμους που λύνουν ένα πρόβλημα μεγέθους M. Ο πρώτος εκτελείται σε M βήματα σε υπολογιστή με M επεξεργαστές και ο δεύτερος σε $\sqrt{M}$ βήματα σε υπολογιστή με M^2 επεξεργαστές. Θέλουμε

να εκτελέσουμε έναν από τους δυο αλγορίθμους Q φορές σε έναν υπολογιστή με N επεξεργαστές. Δώστε συνθήκες για τα Q, M και N για να αποφασίσουμε ποιος αλγόριθμος είναι καλύτερος.

Κεφάλαιο 2

Ακέραια Αριθμητική

2.1 Πρόσθεση με διάδοση κρατουμένου

Οπως είναι γνωστό, με έναν επεξεργαστή μπορούμε να προσθέσουμε 2 αριθμούς μεγέθους N bits σε $N+1$ βήματα. Το ερώτημα που τίθεται είναι αν μπορούμε καλύτερα με πολλούς επεξεργαστές. Μια πρώτη απάντηση είναι όχι, αφού για να προστεθούν τα δυο περισσότερα σημαντικά ψηφία θα πρέπει να γνωρίζουμε τα κρατούμενα που έχουν προκύψει από τις προσθέσεις των λιγότερο σημαντικών ψηφίων. Αυτό σημαίνει απλά ότι ο συγκεκριμένος σειριακός αλγόριθμος δεν παραλληλοποιείται.

Σε αυτή την παράγραφο θα δείξουμε πως μπορούμε να προσθέσουμε 2 αριθμούς μεγέθους N bit σε $2\log N + 1$ βήματα, παραλληλοποιώντας έναν άλλο αλγόριθμο. Ο αλγόριθμος που θα περιγράψουμε λέγεται 'πρόσθεση με διάδοση κρατουμένου' (carry lookahead addition) και είναι σχετικά απλός αλλά όχι τετριμμένος.

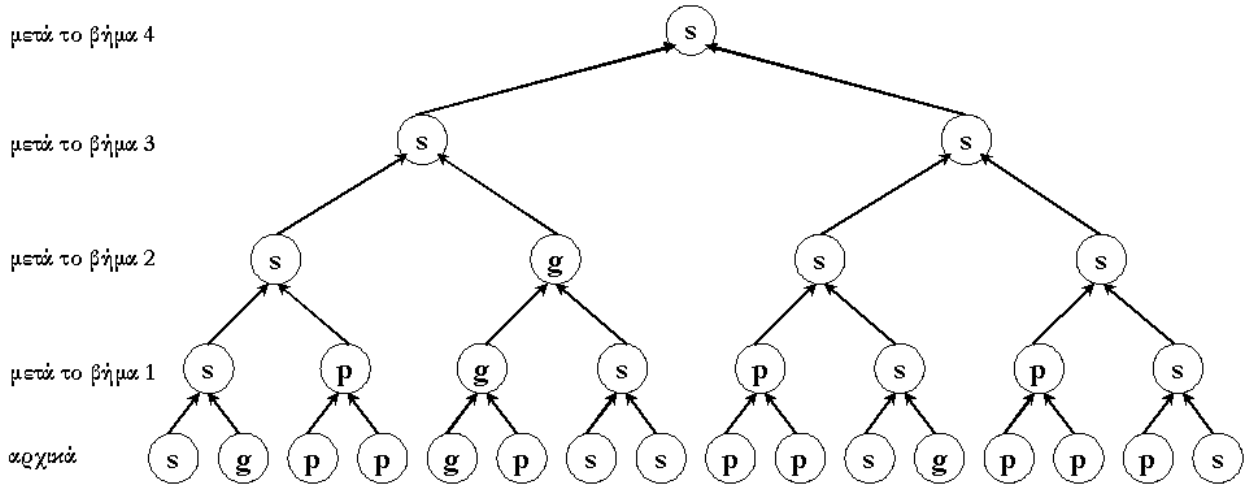
Η ιδέα του αλγορίθμου είναι να κρατάμε για κάθε πρόσθεση αντίστοιχων bit την πληροφορία αν η συγκεκριμένη πρόσθεση: σταματά (s), διαδίδει (p) ή γεννά (g) κρατούμενο. Παρατηρούμε ότι η τιμή του i–οστού κρατουμένου είναι 1 αν και μόνο αν το αριστερότερο μη p bit στα δεξιά του i–οστού bit είναι g (Σχήμα 2.1).

Χρησιμοποιούμε ένα 'δέντρο διάδοσης κρατουμένου'. Οι τιμές s,p,g αποθηκεύονται στα φύλλα (Σχήμα 2.2). Η τιμή κάθε εσωτερικού κόμβου τίθεται στην τιμή του αριστερότερου μη p παιδιού. Αν και τα δυο παιδιά έχουν τιμή p, η τιμή του πατέρα τίθεται στην τιμή p. Το δέντρο διάδοσης κρατουμένου δημιουργείται σε $\log N$ βήματα και κάθε κόμβος περιέχει την πληροφορία αν το υποάθροισμα των φύλλων που είναι απόγονοί του σταματά, διαδίδει ή γεννά κρατούμενο.

```
0 1 0 1 1 1 0 0 1 0 0 1 0 0 1 0
0 1 1 0 1 0 0 0 0 1 0 1 1 1 0 0
-----------------------------------------------
1 1 0 0 0 1 0 0 1 1 1 0 1 1 1 0  Άθροισμα
0 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0  Κρατούμενο

s g p p g p s s p p s  g p p p s
```

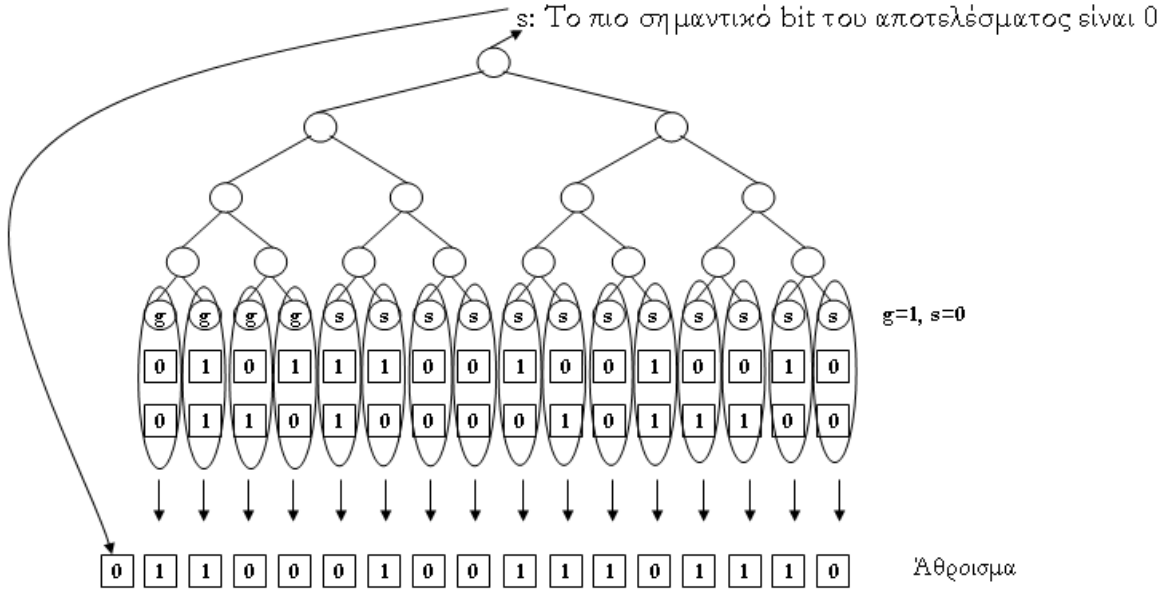
Σχήμα 2.1: Προσδιορισμός των υποαθροισμάτων που σταματούν (s), διαδίδουν (p) ή δημιουργούν (g) κρατούμενο.



Σχήμα 2.2: Δημιουργία των τιμών σε ένα carry-lookahead δένδρο.

Η τιμή της ρίζας είναι g αν όλο το άθροισμα γεννά κρατούμενο. Καθώς ένας εσωτερικός κόμβος παίρνει τις τιμές των παιδιών του, αντικαθιστά την τιμή του αριστερού παιδιού με την τιμή του δεξιού σβήνοντας τις προγούμενες τιμές τους. Ειδικά για τη ρίζα, η τιμή βγαίνει στην έξοδο και αντικαθίσταται από s. Στα επόμενα βήματα κάθε μη–φύλλο δίνει την τιμή του και στους δυο γιους. Κάθε φύλλο περιμένει μέχρι να λάβει την πρώτη μη p τιμή. Την αποθηκεύει και αγνοεί όλες τις υπόλοιπες. Μετά από $2\log N+1$ κάθε φύλλο θα έχει λάβει και αποθηκεύσει ένα g ή ένα s (εφόσον υπάρχει s στη ρίζα). Το i–οστό bit θα λάβει ένα g πριν ένα s αν και μόνο αν το i–οστό κρατούμενο του αθροίσματος είναι 1.

Στο βήμα $2\log N+1$ κάθε φύλλο γνωρίζει τους i–οστούς προσθετέους και το i–οστό κρατούμενο. Η πρόσθεση (χωρίς κρατούμενο) δίνει το i–οστό bit του αποτελέσματος. Το $N+1$ περισσότερο σημαντικό bit έχει βγει από τη ρίζα.



Σχήμα 2.3: Το τελευταίο βήμα του αλγορίθμου κατά το οποίο παράγεται το άθροισμα όταν δίνονται οι προσθετέοι και οι - υπολογισμένες απο πριν - τιμές των bits κρατουμένων.

Η αποδοτικότητα του αλγορίθμου είναι $\Theta\left(\frac{1}{\log N}\right)$. Η αποδοτικότητα του

αλγορίθμου μπορεί να βελτιωθεί σε $\Theta(1)$ λύνοντας πολλά προβλήματα ταυτόχρονα.

2.2 Υπολογισμοί προθεμάτων

Το πρόβλημα που αντιμετωπίζουμε σε αυτή την παράγραφο είναι το εξής: Εστω συμβολοσειρά με N στοιχεία $x_1, x_2, ..., x_N$ και ένας προσεταιριστικός τελεστής $\otimes$. Να υπολογιστούν τα $y_i = x_1 \otimes x_2 \otimes ... \otimes x_i$, για $1 \leq i \leq N$.

Το πρόβλημα του υπολογισμού κρατουμένων είναι ένα τέτοιο πρόβλημα. Το x_i είναι η τιμή s,p ή g στο i–οστό bit (από δεξιά). Οπως είδαμε στην προηγούμενη παράγραφο, η τιμή του αποτελέσματος y_i είναι η τιμή του αριστερότερου μη p στα δεξιά του i–οστού bit. Αυτό μπορεί να γραφτεί ως:

$$y_i = s \otimes x_1 \otimes ... \otimes x_{i-1}.$$

Ο αναγνώστης μπορεί εύκολα να καθορίσει τον πίνακα τιμών του τελεστή $\otimes$. Εύκολα αποδεικνύεται ότι ο τελεστής είναι προσεταιριστικός.

Το γενικό πρόβλημα μπορεί να λυθεί σε $2D + 1$ βήματα χρησιμοποιώντας ένα δίκτυο με τοπολογία δυαδικού δέντρου βάθους D. Ο αλγόριθμος που ακολουθούμε έχει δυο φάσεις:

1. Κάθε εσωτερικός κόμβος υπολογίζει το γινόμενο των φύλλων που είναι απόγονοί του.

2. Τα γινόμενα διαδίδονται προς τα κάτω ώστε το i–οστό φύλλο να φτιάξει το i–οστό πρόθεμα.

Πιο συγκεκριμένα, στο πρώτο βήμα το x_i είναι είσοδος στο i–οστό φύλλο. Η τιμή αποθηκεύεται και διαδίδεται στον πατέρα. Στα επόμενα βήματα, κάθε εσωτερικός κόμβος υπολογίζει το γινόμενο των εισόδων του και περνά το αποτέλεσμα στον πατέρα. Μετά από $D + 1$ βήματα, κάθε κόμβος θα έχει υπολογίσει το γινόμενο των φύλλων που είναι απόγονοί του. Καθώς ένας εσωτερικός κόμβος δέχεται τις εισόδους του, περνά την τιμή του αριστερού γιου στο δεξιό (προσέξτε ότι η διαδικασία αυτή είναι η αντίθετη με την αντίστοιχη του αλγορίθμου διάδοσης κρατουμένου). Μετά το πρώτο βήμα, τα φύλλα πολλαπλασιάζουν την τιμή που τους έρχεται από πάνω με την τιμή που έχουν αποθηκευμένη. Τα μη φύλλα περνούν την τιμή που έρχεται από πάνω στους δυο γιους. Προφανώς ο αλγόριθμος που περιγράφτηκε απαιτεί $2D + 1$ βήματα, όπου D το βάθος του δέντρου.

Θεώρημα 1. *Ο αλγόριθμος είναι σωστός, δηλ. το i–οστό φύλλο υπολογίζει το $y_i = x_1 \otimes x_2 \otimes ... \otimes x_i$ για $1 \leq i \leq N$.*

Απόδειξη. Χρησιμοποιούμε επαγωγή στο βάθος D του δέντρου. Η περίπτωση $D = 1$ είναι τετριμμένη. Υποθέτουμε ότι ο αλγόριθμος δουλεύει σωστά για όλα τα δέντρα βάθους το πολύ $D-1$. Θεωρούμε το πρόβλημα με δέντρο βάθους D. Εστω k τέτοιο ώστε τα $x_1, ..., x_k$ είναι είσοδοι στο αριστερό υποδέντρο και τα $x_{k+1}, ...x_N$ είναι είσοδοι στο δεξιό. Αν αγνοήσουμε τη ρίζα, το i–οστό φύλλο υπολογίζει το γινόμενο $x_1 \otimes ... \otimes x_N$ αν $i \leq k$ ή το γινόμενο $x_{k+1} \otimes ... \otimes x_N$ αν $k + 1 \leq i \leq N$.

Αυτό που κάνει η ρίζα είναι να περνά την τιμή $x_1 \otimes ... \otimes x_k$ που υπολογίζει το αριστερό υποδέντρο στο δεξιό. Αυτή είναι η τελευταία είσοδος που θα πάρει ένα φύλλο του δεξιού υποδέντρου και θα ανανεώσει την τιμή του σε $x_1 \otimes ... \otimes x_i$. Προσέξτε ότι η ρίζα δεν επηρρεάζει τις τιμές που υπολογίζονται από το αριστερό υποδέντρο. □

Ο παράλληλος αλγόριθμος υπολογισμού προθεμάτων μπορεί να υλοποιηθεί σε κάθε δίκτυο με πεπερασμένο βαθμό με σταθερή επιβράδυνση σε χρόνο ανάλογο της διαμέτρου του δικτύου. Η ιδέα είναι να φτιάξουμε ένα δέντρο με διαπέραση του δικτύου κατά πλάτος. Το βάθος του δέντρου που παράγεται με αυτό τον τρόπο είναι το πολύ ίσο με τη διάμετρο του δικτύου. Ο αλγόριθμος έχει πολλές εφαρμογές. Για το λόγο αυτό είναι συχνά ενσωματωμένος στο υλικό των παράλληλων υπολογιστών.

2.3 Πρόσθεση με αποθήκευση κρατουμένου

Ο αλγόριθμος πρόσθεσης με διάδοση κρατουμένου δουλεύει για 2 αριθμούς μεγέθους k bit και τους προσθέτει σε $2 \log k + 1$ βήματα. Στην παράγραφο αυτή θα δείξουμε πως μπορούμε να προσθέσουμε γρήγορα N αριθμούς μεγέθους k bits.

Η πρώτη ιδέα είναι να εκμεταλλευτούμε τον αλγόριθμο διάδοσης κρατουμένου. Χρησιμοποιώ N δυαδικά δέντρα με $k + \log N$ φύλλα το καθένα. Προσθέτω τα $N/2$ ζευγάρια στα πρώτα $2 \log k + 1$ βήματα, μετά προσθέτω τα $N/4$ ζευγάρια των αθροισμάτων μεγέθους $k + 1$ bit σε $2 \log (k + 1) + 1$ βήματα, κ.ο.κ, τέλος προσθέτω ένα ζευγάρι αριθμών μεγέθους $k + \log N$ bit σε $2 \log (k + \log N) + 1$ βήματα. Ο συνολικός χρόνος φράσσεται από άνω από την ποσότητα:

$$(2 \log (k + \log N) + 1) \log N = \Theta(\log k \log N + \log \log N \log N).$$

Στην παράγραφο αυτή, θα παρουσιάσουμε τον αλγόριθμο 'πρόσθεσης με αποθήκευση κρατουμένου' (carry save addition) που βελτιώνει συνολικά τον απαιτούμενο χρόνο για την πρόσθεση N αριθμών μεγέθους k bit. Η βασική ιδέα του αλγορίθμου είναι, σε κάθε βήμα, η πρόσθεση 3 αριθμών μεγέθους k bit

να μειωθεί στο άθροισμα 2 αριθμών μεγέθους $k+1$ bit. Αυτό μπορεί να γίνει γράφοντας το άθροισμα τριών bit σε κάθε θέση, σαν ένα αριθμό μεγέθους 2 bit που αποτελείται από το bit χαμηλής σημαντικότητας και το bit κρατουμένου. Συνολικά, με αυτό τον τρόπο, το άθροισμα των τριών αριθμών μετασχηματίζεται στο άθροισμα του αριθμού που αποτελείται από τα bit χαμηλής σημαντικότητας και του αριθμού που αποτελείται από τα bit κρατουμένων.

Πρόταση 2. *Εστω $a_k...a_1$, $a'_k...a'_1$ και $a''_k...a''_1$ τριών αριθμών a, a' και a'' που προστίθενται. Για $1 \leq i \leq N$, έστω $c_i d_i$ η δυαδική αναπαράσταση (μεγέθους 2 bit) του αθροίσματος των $a_i + a'_i + a''_i$. Τότε $a + a' + a'' = c + d$ όπου $c_k....c_1 0$ η δυαδική αναπαράσταση του c και $d_k...d_1$ η δυαδική αναπαράσταση του d.*

Απόδειξη.

$$a + a' + a'' = \sum_{i=1}^{k} (a_i + a'_i + a''_i) 2^{i-1} = \sum_{i=1}^{k} (2c_i + d_i) 2^{i-1} =$$

$$= \sum_{i=1}^{k} c_i 2^i + \sum_{i=1}^{k} d_i 2^{i-1} = c + d.$$

□

Πρόταση 3. *Επαναλαμβάνοντας τη μείωση του αθροίσματος 3 αριθμών στο άθροισμα των 2, μπορούμε να μειώσουμε το άθροισμα των N αριθμών μεγέθους k bit στο άθροισμα 2 αριθμών μεγέθους $k + \log N$ bit σε $\log_{3/2} N + 1$ βήματα.*

Απόδειξη. Στο πρώτο βήμα, χωρίζουμε σε τριάδες και μετατρέπουμε το άθροισμα των N αριθμών μεγέθους k bit σε άθροισμα το πολύ $\frac{2N}{3} + \frac{2}{3}$ αριθμών μεγέθους $k+1$ bit. Στο επόμενο βήμα θα μείνουμε με το πολύ

$$\frac{2}{3}\left(\frac{2N}{3} + \frac{2}{3}\right) + \frac{2}{3} = \frac{4N}{9} + \frac{4}{9} + \frac{2}{3}$$

αριθμούς μεγέθους $k+2$ bit. Μετά το j–οστό βήμα θα μείνουμε με το πολύ

$$\left(\frac{2}{3}\right)^j N + \left(\frac{2}{3}\right)^j + \left(\frac{2}{3}\right)^{j-1} + ... + \frac{2}{3} < \left(\frac{2}{3}\right)^j N + \frac{\frac{2}{3}}{1 - \frac{2}{3}} = \left(\frac{2}{3}\right)^j N + 2$$

αριθμούς μεγέθους $k + j$ bit. Εφόσον το άθροισμα των N αριθμών έχει μέγεθος το πολύ $k + \log N$ bit, το συνολικό άθροισμα μειώνεται σε άθροισμα $\left(\frac{2}{3}\right)^{\log_{3/2} N} N + 2 = 3$ αριθμών έπειτα από $\log_{3/2} N$ βήματα. Συνεχίζοντας για ένα ακόμα βήμα, έχουμε ότι το συνολικό άθροισμα μειώνεται σε άθροισμα 2 αριθμών μεγέθους $k + logN$ bit έπειτα από $\log_{3/2} N + 1$ βήματα. □

Για την ολοκλήρωση της πρόσθεσης, χρησιμοποιούμε τον αλγόριθμο διάδοσης κρατουμένου για $2\log(k+\log N)+1$ επιπλέον βήματα. Συνολικά, η πρόσθεση N αριθμών μεγέθους k bit απαιτεί $2\log(k+\log N)+\log_{3/2} N+2 = O(\log k+\log N)$ βήματα.

2.4 Ασκήσεις

1. Δίνεται ένα πλήρες δυαδικό δέντρο με N φύλλα, κάθε ένα από τα οποία έχει μνήμη μεγέθους $O(\log N)$ bit. Δείξτε πως μπορούμε να προσθέσουμε δυο αριθμούς μεγέθους $N\log N$ bit σε χρόνο $O(\log N)$ βημάτων. Ποια είναι η αποδοτικότητα του αλγορίθμου·

2. Δώστε ένα αλγόριθμο υπολογισμού προθέματος σε τριαδικό δέντρο. Πως γενικεύεται ο αλγόριθμος για d–αδικά δέντρα·

3. Πόσο χρόνο απαιτεί η πρόσθεση δυο αριθμών μεγέθους N bit σε ένα δισδιάστατο πλέγμα $\sqrt{N}\times\sqrt{N}$·

4. (πρόβλημα κατανομής δεδομένων) Δίνεται ένα δυαδικό δέντρο βάθους D και m τιμές $v(1),...,v(m)$. Κάθε φύλλο του δέντρου περιέχει είτε μια αίτηση $r(i)$ για την i–οστή τιμή, είτε την τιμή $v(i)$ για κάποιο i. Επιπλέον, οι αιτήσεις $r(i)$ για την i–οστή τιμή βρίσκονται στα αμέσως επόμενα φύλλα από αυτό που περιέχει την τιμή $v(i)$. Το πρόβλημα είναι να κατανεμηθούν οι τιμές $v(i)$ σε κάθε φύλλο που περιέχει αντίστοιχες αιτήσεις $r(i)$, για $1\le i\le m$. Χρησιμοποιώντας αλγόριθμο υπολογισμού προθέματος, δείξτε πως λύνεται το πρόβλημα σε $O(N)$ βήματα.

5. Δείξτε πως μπορούν να λυθούν γραμμικές σχέσεις αναδρομής χρησιμοποιώντας τον αλγόριθμο προθέματος. Συγκεκριμένα, δείξτε πως μπορούν να υπολογιστούν τα $z_1,...,z_n$ σε $O(\log N)$ βήματα χρησιμοποιώντας ένα πλήρες δυαδικό δέντρο N κόμβων, όπου

$$z_i = a_i z_{i-1} + b_i z_{i-2}$$

για $2\le i\le N$, όταν τα $a_2,...,a_N,b_2,...,b_N,z_0$ και z_1 δίνονται σαν είσοδοι.

6. Δίνεται ένα δυαδικό δέντρο βάθους D, στο οποίο κάποια από τα φύλλα περιέχουν πακέτα. Ορίζουμε σαν δείκτη ενός πακέτου p τον αριθμό των πακέτων που βρίσκονται αποθηκευμένα σε φύλλα αριστερά του φύλλου που αποθηκεύει το p. Δώστε αλγόριθμο που να υπολογίζει το δείκτη κάθε πακέτου σε $2D+1$ βήματα.

7. Δείξτε πως μπορούμε να προσθέσουμε N αριθμούς μεγέθους N bit σε $O(N)$ βήματα χρησιμοποιώντας ένα γραμμικό πλέγμα με $N + \log N$ επεξεργαστές. Υποθέστε ότι κάθε επεξεργαστής του γραμμικού πλέγματος έχει είσοδο.

Κεφάλαιο 3

Αλγόριθμοι Ταξινόμησης

3.1 Γενικά

Οι αλγόριθμοι που προαναφέρθηκαν για ταξινόμηση σε γραμμικό πλέγμα επεξεργαστών και σε πλήρες δυαδικό δένδρο απαιτούν $J(N)$ βήματα λέξεων για να ταξινομήσουν N αριθμούς μεγέθους $\log N$ βιτς ο καθένας μιας και τα παραπάνω δίκτυα έχουν μικρό εύρος διχοτόμησης. Στη συνέχεια θα περιγράψουμε τρεις αλγορίθμους για ταξινόμηση N αριθμών (υποθέτουμε ότι το $\sqrt{N}$ είναι δύναμη του 2) σε διδιάστατο πλέγμα επεξεργαστών διαστάσεων $\sqrt{N} \times \sqrt{N}$:

- Αλγόριθμος ταξινόμησης με εναλλαγή άρτιων – περιττών αντιμεταθέσεων σε γραμμικό πλέγμα επεξεργαστών (odd – even transposition sort): Ταξινομεί οποιαδήποτε ακολουθία N αριθμών σε N βήματα.
- Απλός αλγόριθμος ταξινόμησης σε $\sqrt{N}(\log N + 1)$ βήματα.
- Αλγόριθμος ταξινόμησης σε $3\sqrt{N} + o(N)$ βήματα που αποτελούν σχεδόν το βέλτιστο χρόνο για ταξινόμηση σε διδιάστατα πλέγματα.

Όλοι αυτοί οι αλγόριθμοι υλοποιούνται με πεπερασμένο αριθμό απλών τοπικών λειτουργιών και υποθέτουμε ότι οι αριθμοί βρίσκονται αρχικά στους επεξεργαστές του πλέγματος κι εκεί παραμένουν μετά το τέλος της διαδικασίας ταξινόμησης.

3.2 Ενας αλγόριθμος ταξινόμησης για το γραμμικό πλέγμα

Για να πραγματοποιήσουμε ταξινόμηση αριθμών σε mesh πρέπει ουσιαστικά να ταξινομήσουμε αριθμούς σε στήλες και γραμμές δηλαδή σε γραμμικά πλέγ-

ματα με

τη διαφορά ότι δε λαμβάνουμε υπ'οψη το χρόνο που απαιτείται για είσοδο και έξοδο των αριθμών στο πλέγμα (όπως προηγουμένως). Στη συνέχεια περιγράφουμε έναν αλγόριθμο που ταξινομεί N αριθμούς σε ένα γραμμικό πλέγμα N επεξεργαστών και απαιτεί N βήματα λέξεων.

Βασική Ιδέα: Στα περιττά βήματα, συγκρίνουμε τα περιεχόμενα των θέσεων $(1,2)$ $(3,4)$ κ.ο.κ. με τελικό σκοπό η μικρότερη τιμή να βρεθεί στην αριστερότερη θέση του ζευγαριού. Στα άρτια βήματα κάνουμε την ίδια διαδικασία για τα ζεύγη θέσεων $(2,3)$ $(4,5)$ κ.ο.κ. Γιάυτό και ο αλγόριθμος αναφέρεται σαν αλγόριθμος με εναλλαγή των άρτιων – περιττών αντιμεταθέσεων. Προκειμένου να αποδείξουμε την ορθότητα του αλγορίθμου ταξινόμησης με εναλλαγή άρτιων – περιττών αντιμεταθέσεων, λαμβάνουμε υπ'οψη μας ένα απλό και σημαντικό λήμμα: το $0-1$ Λήμμα.

Λήμμα 4. *Αν ένας αλγόριθμος σύγκρισης – ταξινόμησης, στον οποίο η παρούσα κατάσταση δεν εξαρτάται από προηγούμενες καταστάσεις, ταξινομεί κάθε σύνολο εισόδου που αποτελείται μόνο από* 0 *και* 1*, τότε ταξινομεί και όλα τα σύνολα εισόδου που περιέχουν αυθαίρετες τιμές.*

Απόδειξη. Έστω ότι υπάρχει αλγόριθμος με τα παραπάνω χαρακτηριστικά, δηλαδή

- η παρούσα κατάστασή του δεν εξαρτάται από προηγούμενες
- ταξινομεί κάθε σύνολο εισόδου που αποτελείται μόνο από 0 και 1

ο οποίος δεν ταξινομεί σωστά κάποιο σύνολο εισόδου με τιμές x_1, x_2, ..., x_N . Έστω επίσης:

- π μια διάταξη τέτοια ώστε: $x_\pi(1) \leq x_\pi(2) \leq ... \leq x_\pi(N)$
- s μια διάταξη τέτοια ώστε η ακολουθία εξόδου (αποτέλεσμα) του αλγορίθμου ταξινόμησης να είναι $x_{\sigma(1)}$, $x_{\sigma(2)}$, ... $x_{\sigma(N)}$.

Θεωρούμε k τη μικρότερη τιμή για την οποία είναι: $x_{\sigma(k)} \neq x_{\pi(k)}$ και αφού υποθέσαμε ότι ο αλγόριθμος κάνει λάθος στην ταξινόμηση της ακολουθίας x_1, x_2, ..., x_N, πράγματι υπάρχει τέτοια τιμή k. Τότε, όμως, από τον ορισμό είναι $x_{\sigma(i)} = x_{\pi(i)}$ για $1 \leq i \leq k$ και κατά συνέπεια $x_{\sigma(k)} > x_{\pi(k)}$. Επομένως, θα πρέπει να υπάρχει τιμή r, με $r > k$ τέτοια ώστε $x_{s(r)} = x_{\pi(k)}$. Ορίζουμε $x_i^* = 0$ αν $x_i \leq x_{\pi(k)}$ και $x_i^* = 1$ αν $x_i > x_{\pi(k)}$. Αντικαθιστούμε τα x_i με τα x_i^* για κάθε i, $1 \leq i \leq N$ και εξετάζουμε πώς ο αλγόριθμος ταξινομεί την καινούρια

ακολουθία. Αν είναι τότε είναι και $x_i \geq x_j$ για κάθε i, j και η διαδικασία είναι η ίδια που εφαρμόστηκε και για την αρχική ακολουθία. Άρα, το αποτέλεσμα του αλγορίθμου για τις τιμές $0-1$ θα είναι η ακολουθία:

$$x^*_{\sigma(1)}, x^*_{\sigma(2)}, x^*_{\sigma(3)}..., x^*_{\sigma(k-1)}, x^*_{\sigma(k)}, ...x^*_{\sigma(r)}... = 0, 0, ..., 0, 1, ..., 0, ...$$

που είναι μεν σωστή, αλλά έρχεται σε αντίθεση με την υπόθεση ότι ο αλγόριθμος ταξινομεί σωστά οποιαδήποτε ακολουθία από 0 και 1. Καταλήγουμε, επομένως, σε άτοπο και κατά συνέπεια αποδεικνύεται η ορθότητα του $0-1$ λήμματος. □

Με βάση το παραπάνω απλό αλλά αρκετά ισχυρό λήμμα απλοποιείται πολύ η απόδειξη του ότι ο αλγόριθμος ταξινόμησης με εναλλαγή άρτιων – περιττών αντιμεταθέσεων λειτουργεί σωστά αφού αρκεί να αποδείξουμε ότι ο αλγόριθμος ταξινομεί σωστά μια τυχαία ακολουθία μεγέθους N από '0' και '1'. Έστω, λοιπόν, ότι εισάγουμε την ακολουθία σε ένα γραμμικό πλέγμα N επεξεργαστών ώστε κάθε όρος της να αποτελεί είσοδο για ένα επεξεργαστή του πλέγματος και έστω k ο αριθμός των '1' της ακολουθίας. Αυτό που ουσιαστικά πρέπει να δείξουμε είναι ότι ο αλγόριθμος ταξινόμησης με εναλλαγή άρτιων – περιττών αντιμεταθέσεων μετακινεί τα k '1' στους επεξεργαστές $N-k+1$, $N-k+2$, ..., N σε N βήματα. Ονομάζουμε j_i τον επεξεργαστή που αρχικά περιέχει το δεξιότερο '1'. Τότε:

- αν το j_i είναι σε άρτια θέση το δεξιότερο '1' δεν μετακινείται κατά το πρώτο βήμα του αλγορίθμου. Αυτό συμβαίνει γιατί το περιεχόμενο του επεξεργαστή j_i συγκρίνεται με το περιεχόμενο του επεξεργαστή $j_i - 1$, κατά το πρώτο βήμα του αλγορίθμου και το '1' δε μπορεί να πάει πιο αριστερά σε καμιά περίπτωση (είτε ο επεξεργαστής $j_i - 1$ περιέχει '0' έιτε '1'). Κατά το δεύτερο βήμα, όμως, το δεξιότερο '1' θα κινηθεί προς τα δεξιά με την προϋπόθεση ότι $j_i < N$.

- αν το j_i είναι σε περιττή θέση το δεξιότερο '1' θα μετακινηθεί προς τα δεξιά κατά το πρώτο βήμα του αλγορίθμου. Και ανεξάρτητα από το αν το j_i είναι σε άρτια ή περιττή θέση, αφού το δεξιότερο '1' κινήθηκε προς τα δεξιά, θα συνεχίσει να κινείται προς την ίδια κατεύθυνση σε όλα τα επόμενα βήματα του αλγορίθμου εως ότου φτάσει στον επεξεργαστή N.

Εξετάζουμε, τώρα, την κίνηση του δεύτερου κατά σειρά δεξιότερου '1' της ακολουθίας και έστω j_2 η αρχική θέση του στο πλέγμα. Από τη στιγμή που το δεξιότερο '1' κινείται προς τα δεξιά σε κάθε βήμα μετά το πρώτο μέχρι να φτάσει στον τελευταίο επεξεργαστή, δε μπορεί ποτέ να εμποδίσει το δεύτερο δεξιότερο '1' να κινηθεί προς τα δεξιά μέχρι αυτό να φτάσει τον $N-1$ επεξεργαστή. Γενικά,

το i–οστό δεξιότερο `1′ θα κινηθεί προς τα δεξιά στο $i+1$ βήμα καθώς και σε όλα τα επόμενα μέχρι να φτάσει στον $N-i+1$ επεξεργαστή (αποδεικνύεται εύκολα με επαγωγή στο $i$). Οπότε, το κατά σειρά $k$ δεξιότερο `1′ (δηλαδή το αριστερότερο `1′) θα μετακινηθεί προς τα δεξιά στα βήματα $k+1$, $k+2$, ..., $N$ εκτός κι αν έχει φτάσει στη τελική του θέση, δηλ. στον επεξεργαστή $N-k+1$ πριν το βήμα $N$. Άρα το κατά σειρά $k$ δεξιότερο `1′ θα φτάσει στην τελική του θέση μέχρι να συμπληρωθούν N βήματα. Αν λάβουμε υπόψιν και το 0–1 Λήμμα, ότι, δηλαδή, αφού ο αλγόριθμος ταξινομεί σωστά οποιαδήποτε ακολουθία $0-1$, θα λειτουργεί σωστά και οποιαδήποτε ακολουθία από οποιουσδήποτε αριθμούς, καταλήγουμε στο συμπέρασμα ότι ο αλγόριθμος εναλλαγής άρτιων – περιττών αντιμεταθέσεων ταξινομεί σωστά οποιαδήποτε ακολουθία N αριθμών σε N βήματα.

3.3 Ενας αλγόριθμος ταξινόμησης στο διδιάστατο πλέγμα

Η ιδέα του αλγορίθμου είναι η εξής: οι γραμμές ταξινομούνται στις φάσεις 1, 3, ..., $\log\sqrt{N}+1$ ώστε ο μικρότερος αριθμός να πάει αριστερότερα στις περιττές γραμμές και δεξιότερα στις άρτιες. Οι στήλες ταξινομούνται στις φάσεις 2, 4, ..., $\log\sqrt{N}$ ώστε ο μικρότερος αριθμός να πάει προς τα πάνω. Η ολοκλήρωση της παραπάνω διαδικασίας έχει σαν αποτέλεσμα την εμφάνιση των αριθμών στο πλέγμα βουστροφηδόν (snakelike) μετά από $2\log\sqrt{N}+1=\log N+1$ φάσεις. Ο αλγόριθμος ολοκληρώνει την ταξινόμηση σε $\log N+1$ φάσεις. Οι υποθέσεις του 0–1 ικανοποιούνται, οπότε αν δείξουμε ότι ο αλγόριθμος ταξινομεί σωστά οποιαδήποτε ακολουθία από 0 και 1, έχουμε εξασφαλίσει την ορθότητα της λειτουργίας του. Εστω ένα πλέγμα που τα στοιχεία του, που είναι αρχικά αποθηκευμένα στους επεξεργαστές είναι μόνο 0 και 1. Αναφέρουμε, εδώ, ότι οι γραμμές που περιέχουν και 0 και 1 λέγονται βρώμικες (dirty rows). Ενώ, αυτές που περιέχουν μόνο 0 ή μόνο 1 λέγονται καθαρές (clean rows). Τότε, αν ταξινομηθούν οι γραμμές και μετά οι στήλες, τουλάχιστο οι μισές από τις βρώμικες γραμμές θα καταλήξουν να είναι καθαρές. Προκειμένου να εξηγήσουμε το συμπέρασμα αυτό, σκεφτόμαστε με τον ακόλουθο τρόπο.Χωρίζουμε τις γραμμές σε τρεις περιοχές:

- πάνω περιοχή όπου ανήκουν γραμμές που περιέχουν μόνο 0,
- κάτω περιοχή όπου ανήκουν γραμμές που περιέχουν μόνο 1,
- μεσαία περιοχή όπου υπάρχουν βρώμικες γραμμές.

Στη συνέχεια, οι βρώμικες γραμμές ομαδοποιούνται ακολουθιακά σε ζευγάρια και ταξινομούνται. Τρία είναι τα πιθανά αποτελέσματα:

1. το ταξινομημένο ζευγάρι γραμμών να έχει περισσότερα 0,

0.....01.....1

1.....10.....0

2. το ταξινομημένο ζευγάρι γραμμών να έχει περισσότερα 1,

0.....01.....1

1.....10.....0

3. το ταξινομημένο ζευγάρι γραμμών να έχει ίσο αριθμό από 0 και 1.

0.....01.....1

1.....10.....0

Στη συνέχεια, ταξινομούμε κατά στήλες. Υποθέτουμε, εδώ, ότι κατά το πρώτο βήμα της ταξινόμησης, συγκρίνονται και ταξινομούνται στοιχεία της ίδιας στήλης σε κάθε ζεύγος γραμμών. Μετά τη ολοκλήρωση αυτής της διαδικασίας, τα ζεύγη γραμμών εμφανίζονται ως εξής:

0.....0

1....10...01....1

διαδοχικά ζεύγη βρώμικων γραμμών που περιέχουν περισσότερα 0

0....01...10....0

1.....1

διαδοχικά ζεύγη βρώμικων γραμμών που περιέχουν περισσότερα 1

0.....0

1.....1

διαδοχικά ζεύγη βρώμικων γραμμών που περιέχουν τον ίδιο αριθμό 0 και 1

Μετά, όλες οι γραμμές που περιέχουν μόνο 0 μετακινούνται προς τα πάνω και όλες οι γραμμές που περιέχουν μόνο 1 μετακινούνται προς τα κάτω. Συνεχίζουμε, ομοιοτρόπως, μέχρι να ταξινομηθούν όλες οι στήλες, δηλαδή να

υπάρχει μόνο μια βρώμικη γραμμή. Σε κάθε φάση (ταξινόμηση κατά γραμμές και στήλες), τουλάχιστο μια από τις γραμμές κάθε ζεύγους γίνεται καθαρή, οπότε η μεσαία περιοχή των βρώμικων γραμμών μειώνεται, τουλάχιστο, στο μισό μετά από $2 \log \sqrt{N} = \log N$ φάσεις, όλοι οι αριθμοί εκτός από μια γραμμή έχουν ταξινομηθεί. Μόλις ταξινομηθούν και τα στοιχεία αυτής της γραμμής, ο αλγόριθμος τερματίζει. Τα παραπάνω ισχύουν ανεξάρτητα από τη μέθοδο που υιοθετείται γιατις επιμέρους ταξινομήσεις, αφού η ουσία του αλγορίθμου βρίσκεται στο ρυθμό συρρίκνωσης της μεσαίας περιχής των βρώμικων γραμμών. Αν, επομένως, επιλεχθεί ο αλγόριθμος εναλλαγής άρτιων – πεεριττών θέσεων για τις επιμέρους ταξινομήσεις ($\sqrt{N}$ βήματα), θα χρειαστούν $\sqrt{N}(\log N + 1)$ βήματα για να τερματίσει ο αλγόριθμος.

Προσοχή: Στην πράξη, τα βήματα μπορούν να γίνουν και λιγότερα, μιας και μπορεί να μην είναι απαραίτητο να εκτελεστούν όλα.

Το πέρασμα από φάση σε φάση Ένα άλλο βασικό ζήτημα είναι το ακόλουθο: 'πώς καταλαβαίνει κάθε επεξεργαστής πότε να σταματήσει μια φάση και να προχωρήσει στην επόμενη'. Υπάρχουν τρεις εκδοχές:

- Να υπάρχει τοπικό ρολόι σε κάθε επεξεργαστή μόνο που αυτό θα απιτούσε επιπλέον κόστος στην υλοποίηση και πιο πολύπλοκες λειτουργίες ανά επεξεργαστή.
- Να εξομοιώνεται κάθε διάδοση με επαναχρονισμό, να μηδενίζεται δηλαδή κάποιο γενικό ρολόι και να αρχίζει η μέτρηση του χρόνου από την αρχή. Κάτι τέτοιο, όμως, θα επέφερε περίπου διπλάσια καθυστέρηση.
- Να υπάρχουν δύο ειδών ακμών (ticks) ρολογιού που θα καθορίζουν τη λειτουργία κάθε επεξεργαστή: ένα για κανονική λειτουργία και ένα για να σηματοδοτεί αλλαγή φάσης. Αυτή η εκδοχή είναι ευκολότερη στην υλοποίηση αν και δεν είναι η πλέον κατάλληλη για δίκτυα σταθερών συνδέσεων.

3.4 Ενας καλύτερος αλγόριθμος

Για μεγάλες τιμές του N, δηλαδή όταν έχουμε να ταξινομήσουμε πολλούς αριθμούς, ο προηγούμενος αλγόριθμος είναι εξαιρετικά αργός, ακόμα και αν υποστεί βελτιώσεις. Στη συνέχεια, παρατίθεται ένας πιο πολύπλοκος αλγόριθμος ο οποίος απαιτεί μόνο $3\sqrt{N} - o(N)$ βήματα που είναι και ο βέλτιστος χρόνος για ταξινόμηση σε διδιάστατο πλέγμα. Ο αλγόριθμος ταξινομεί N αριθμούς βουστροφηδόν ως εξής:

Φάση 1: Χώρισε το πλέγμα σε $N^{1/4}$ blocks μεγέθους $N^{3/8} \times N^{3/8}$ και ταυτόχρονα ταξινόμησε κάθε block βουστροφηδόν.

Φάση 2: Εκτέλεσε μια διάταξη των στηλών έτσι ώστε $N^{3/8}$ στήλες σε κάθε block να κατανεμηθούν ομοιόμορφα ανάμεσα στις $N^{1/8}$ στήλες από blocks.

Φάση 3: Ταξινόμησε τα blocks βουστροφηδόν.

Φάση 4: Ταξινόμησε τις στήλες γραμμικά.

Φάση 5: Ταξινόμησε τα blocks 1 και 2, 3 και 4, κ.ο.κ. κάθε στήλης από blocks βουστροφηδόν.

Φάση 6: Ταξινόμησε τα blocks 2 και 3, 4 και 5, κ.ο.κ. κάθε στήλης από blocks βουστροφηδόν.

Φάση 7: Ταξινόμησε γραμμικά τις γραμμές ακολουθώντας την κατεύθυνση του συνολικού σχήματος (snake).

Φάση 8: Εκτέλεσε 2 $N^{3/8}$ βήματα του αλγορίθμου εναλλαγής άρτιων περιττών θέσεων στο συνολικό σχήμα (snake) μεγέθους N. Οι φάσεις 1, 3, 5 και 6 μπορούν να εκτελεστούν με χρήση του αλγορίθμου που αναφέρθηκε στην προηγούμενη παράγραφο σε το πολύ $O(N^{3/8} \log N)$ βήματα. Οι φάσεις 4 και 7 μπορούν να εκτελεστούν με τον αλγόριθμο εναλλαγής άρτιων – περιττών αντιμεταθέσεων σε $2\sqrt{N}$ βήματα. Η φάση 2 μπορεί να εκτελεστεί με διάφορες μεθόδους σε όχι παραπάνω από $\sqrt{N} + O(N^{3/8})$ βήματα. Η φάση 8 μπορεί να εκτελεστεί με τον αλγόριθμο εναλλαγής άρτιων – περιττών αντιμεταθέσεων σε μόνο $2N^{3/8}$ βήματα. Επομένως, ο συνολικός χρόνος που απαιτείται για να τρέξει ο αλγόριθμος είναι $3\sqrt{N} + O(N^{3/8} \log N) = 3\sqrt{N} - o(N)$ βήματα. Ο χρόνος αυτός θεωρείται βέλτιστος αφού για λογικές τιμές του N μπορεί να μειωθεί ελάχιστα. Παρατηρώντας ότι σε κάθε παρούσα κατάσταση του αλγορίθμου δεν εξαρτάται από τις προηγούμενες, μπορούμε να χρησιμοιήσουμε το 0 – 1 Λήμμα για να αποδείξουμε την ορθότητά του. Ετσι, θεωρούμε ότι έχουμε να ταξινομήσουμε μια ακολουθία από 0 και 1, μεγέθους N. Ακολουθώντας τα βήματα που προαναφέρθηκαν, έχουμε:

Μετά τη Φάση 1: Το πολύ μια σειρά σε κάθε block είναι βρώμικη.

Μετά τη Φάση 2: Σε κάθε γραμμή από blocks, ο αριθμός των 1 σε ένα block μπορεί να διαφέρει το πολύ κατά $N^{1/8}$ από τον αριθμό των 1 σε κάθε άλλο block της γραμμής.

Μετά τη Φάση 3: Το πολύ δύο γραμμές σε κάθε γραμμή από blocks είναι βρώμικες.

Μετά τη Φάση 4: Υπάρχουν το πολύ $N^{1/8}$ βρώμικες γραμμές σε καάθε στήλη από blocks.

Μετά τις Φάσεις 5, 6: Ταξινομούνται βουστροφηδόν οι στήλες από blocks (κάθε μία) και μένει μια βρώμικη γραμμή σε κάθε στήλη από blocks. Με βάση το συμπέρασμα της Φάσης 2, καταλήγουμε στο ότι: ο αριθμός των 1 σε δύο στήλες από blocks διαφέρει το πολύ κατά $N^{1/4}$ μετά τη Φάση 2. Οι Φάσεις 3 – 6 καθώς και αυτές που ακολουθούν δε μεταβάλουν το πλήθος των 1 σε κάθε στήλη από blocks. Αρα, μετά από τη Φάση 6 υπάρχουν το πολύ δύο βρώμικες γραμμές συνολικά. Αν μετά τη Φάση 6 υπάρχει μόνο μια βρώμικη γραμμή, στη Φάση 7 τερματίζει ο αλγόριθμος. Αν υπάρχουν δύο βρώμικες γραμμές τότε η πιο πάνω από αυτές περιέχει το πολύ $N^{3/8}$ 1 και όλα τα υπόλοιπα στοιχεία της είναι 0. Ενώ η πιο κάτω περιέχει το πολύ $N^{3/8}$ 0 και όλα τα υπόλοιπα στοιχεία της είναι 1. Οπότε η ταξινόμηση ολοκληρώνεται μετά τις Φάσεις 7 και 8. Αρα, η τυχαία ακολουθία από 0 και 1 ταξινομείται σωστά από τον αλγόριθμο και με βάση το 0 – 1 Λήμμα, κάθε ακολουθία αριθμών θα ταξινομηθεί σωστά.

3.5 Κάτω φράγματα για αλγορίθμους ταξινόμησης

Ενα προφανές κάτω φράγμα στον ελάχιστο χρόνο που χρειάζεται ένας αλγόριθμος ταξινόμησης σε δισδιάστατο πλέγμα $\sqrt{N} \times \sqrt{N}$ είναι $2\sqrt{N}-2$ βήματα. Στην παράγραφο αυτή θα δείξουμε ότι υπάρχει χειρότερο κάτω φράγμα στην απόδοση των αλγορίθμων ταξινόμησης όταν η τελική επιθυμητή διάταξη είναι βουστροφηδόν.

Θεώρημα 5. *Ο ελάχιστος χρόνος που απαιτεί οποιοσδήποτε αλγόριθμος ταξινόμησης σε δισδιάστατο πλέγμα διαστάσεων $\sqrt{N} \times \sqrt{N}$ για να διατάξει N στοιχεία σε βουστροφηδόν σειρά είναι τουλάχιστον $3\sqrt{N} - o(\sqrt{N})$ βήματα.*

Απόδειξη. Έστω $2\sqrt{N}$ άγνωστες τιμές αποθηκευμένες στο πάνω αριστερό $2N^{1/4} \times 2N^{1/4}$ τρίγωνο του πλέγματος. Οι αριθμοί $1, ..., N - 2\sqrt{N}$ είναι αποθηκευμένοι στο υπόλοιπο πλέγμα. Εστω x ο αριθμός που κρατά ο επεξεργαστής $(\sqrt{N} \times \sqrt{N})$ στο βήμα $2\sqrt{N}-2N^{1/4}-3$. Παρατηρήστε ότι η τιμή αυτή

δεν εξαρτάται από τις τιμές που υπάρχουν αρχικά στο πάνω αριστερά τρίγωνο. Εστω $C(m,x)$ η σωστή στήλη για το x όταν m από τους άγνωστους αριθμούς στο τρίγωνο έχουν την τιμή 0 και οι υπόλοιποι την τιμή N. Καθώς το m κυμαίνεται μεταξύ του 0 και του $2\sqrt{N}$, το $C(m,x)$ κυμαίνεται από το 1 ως το $\sqrt{N}$. Οπότε, το $C(m,x)$ παίρνει κάθε πιθανή τιμή (1 ως $\sqrt{N}$) τουλάχιστον 2 φορές. Επιλέγουμε m' έτσι ώστε $C(m',x)=1$, θέτοντας m' αριθμούς του τριγώνου στην τιμή 0 και τις υπόλοιπες στην τιμή N. Για να ολοκληρωθεί ο αλγόριθμος, το x που βρίσκεται στον επεξεργαστή $(\sqrt{N}\times\sqrt{N})$ πρέπει να μεταφερθεί σε κάποιον επεξεργαστή της πρώτης στήλης. Δηλ. απαιτούνται επιπλέον $\sqrt{N}-1$ βήματα, οπότε συνολικά έχουμε

$$3\sqrt{N}-2N^{1/4}-4=3\sqrt{N}-o(\sqrt{N})$$

βήματα. □

Το κάτω φράγμα ισχύει για κάθε τελική διάταξη στην οποία αλλαγή σε $2\sqrt{N}$ εισόδους μπορεί να προκαλέσει αλλαγές σε $\sqrt{N}$ στήλες ή σειρές στην τελική θέση ενός στοιχείου που αποθηκεύεται στον επεξεργαστή $(\sqrt{N}\times\sqrt{N})$. Βέβαια, υπάρχουν τελικές διατάξεις για τις οποίες το κάτω φράγμα δεν ισχύει.

3.6 Ασκήσεις

1. Αν κάθε κόμβος μπορεί να κρατά 4 στοιχεία, δείξτε πως μπορούμε να λύσουμε 4 προβλήματα ταξινόμησης N στοιχείων σε $3\sqrt(N)+o(N)$ βήματα σε ένα πλέγμα $\sqrt{N}\times\sqrt{N}$. (Υποθέστε ότι κάθε σύνδεση μπορεί να χρησιμοποιείται σε κάθε βήμα.)

2. Δείξτε ότι ο παρακάτω αναδρομικός αλγόριθμος ταξινομεί N αριθμούς σε $O(\sqrt{N})$ βήματα σε ένα πλέγμα $\sqrt{N}\times\sqrt{N}$. Αρχικά ταξινόμησε αναδρομικά τα τεταρτημόρια βουστροφηδόν. Στη συνέχεια, ταξινόμησε τις σειρές σε έναλλάξ΄ σειρά και τις στήλες από πάνω προς τα κάτω. Τέλος, εκτέλεσε $4\sqrt{N}$ βήματα όδδ–εεν τρανσποσιτιον σορτ΄ σε όλη την γραμμική διάταξη (σνακε).

3. Δώστε έναν αλγόριθμο για ταξινόμηση N στοιχείων που εκτελείται σε $O(d)$ σε δισδιάστατο πλέγμα αν κάθε στοιχείο αρχικά είναι αποθηκευμένο σε απόσταση το πολύ d από τον τελικό προορισμό του.

4. Δείξτε πως μπορούν να ταξινομηθούν N αριθμοί σε $\sqrt{8N}$ βήματα σε πλέγμα $\sqrt{2N}\times\sqrt{N/2}$.

Κεφάλαιο 4

Αλγόριθμοι Δρομολόγησης

4.1 Το πρόβλημα

Ενα πρόβλημα δρομολόγησης πακέτων ορίζεται από M πακέτα, καθένα από τα οποία έχει ένα κόμβο προορισμού p_i. Αρχικά, τα πακέτα είναι αυθαίρετα αποθηκευμένα στους N κόμβους του δικτύου. Το πρόβλημα είναι να δρομολογηθούν τα πακέτα στους προορισμούς τους χρησιμοποιώντας τοπικό έλεγχο σε κάθε επεξεργαστή και όσο το δυνατόν λιγότερα βήματα. Συνήθως, υποθέτουμε ότι κάθε επεξεργαστής κρατά αρχικά ένα πακέτο και ότι οι προορισμοί των πακέτων είναι διαφορετικοί ($p_i \neq p_j$ για $1 \leq j \leq M$). Αυτό το πρόβλημα ονομάζεται 1–1 πρόβλημα δρομολόγησης ή πρόβλημα δρομολόγησης μεταθέσεων. Στη γενική μορφή ένα μ–ν πρόβλημα δρομολόγησης απαιτεί κάθε επεξεργαστής να έχει αρχικά το πολύ m πακέτα και να είναι ποορισμός το πολύ n πακέτων.

4.2 Απληστοι αλγόριθμοι

Ορισμός 6. *Ενας αλγόριθμος δρομολόγησης σε ένα δίκτυο επεξεργαστών ονομάζεται άπληστος αν οδηγεί κάθε πακέτο από τον αρχικό κόμβο στον κόμβο προορισμού μέσω του ελάχιστου μονοπατιού μεταξύ των δυο κόμβων.*

Ο άπληστος αλγόριθμος δρομολόγησης στο γραμμικό πλέγμα είναι βέλτιστος τόσο ως προς το χρόνο εκτέλεσης όσο και ως προς το μέγιστο μέγεθος των ουρών. Τα πακέτα κινούνται προς την επιθυμητή κατεύθυνση μέχρι να φθάσουν στον κόμβο προορισμό. Παρατηρούμε ότι τα πακέτα δεν συγκρούονται μεταξύ τους και η δρομολόγηση ολοκληρώνεται το πολύ σε $N - 1$ βήματα, όπου N το μέγεθος του γραμμικού πλέγματος. Ενα σημαντικό πρόβλημα με

τους άπληστους αλγορίθμους όταν εφαρμόζονται σε πιο πολύπλοκα δίκτυα επεξεργαστών εμφανίζεται όταν δυο η περισσότερα πακέτα θέλουν ταυτόχρονα να χρησιμοποιήσουν την ίδια ακμή προς την ίδια κατεύθυνση. Η λύση δίνεται σε αυτή την περίπτωση από πρωτόκολλα λύσης ανταγωνισμού (contention resolution protocols). Το σενάριο που μας ενδιαφέρει σε αυτή την παράγραφο είναι το εξής:

- οι ουρές μπορεί να μεγαλώσουν αυθαίρετα και
- προτεραιότητα δίνεται στο πακέτο που πρόκειται να ταξιδέψει μακρύτερα (farthest–first contention resolution protocol).

Στη συνέχεια παρουσιάζουμε το βασικό άπληστο αλγόριθμο στο δισδιάστατο πλέγμα.

Φάση 1. Δρομολόγηση στις γραμμές χρησιμοποιώντας τον άπληστο αλγόριθμο στο γραμμικό πλέγμα. Κοστίζει $\sqrt{N}-1$ βήματα.

Φάση 2. Δρομολόγηση στις στήλες χρησιμοποιώντας πάλι τον άπληστο αλγόριθμο στο γραμμικό πλέγμα. Κοστίζει άλλα $\sqrt{N}-1$ βήματα. Πριν μπούμε στην ανάλυση της απόδοσης του αλγορίθμου, θα κάνουμε κάποιες σημαντικές παρατηρήσεις. Κάποια πακέτα μπορεί να έχουν μπει ήδη στη δεύτερη φάση πριν ολοκληρωθεί η πρώτη φάση από όλα τα πακέτα. Κάτι τέτοιο το αγνοούμε στην ανάλυση που ακολουθεί. Επίσης, σημειώστε ότι χρησιμοποιώντας λάθος πρωτόκολλο λύσης ανταγωνισμού, ο συνολικός αριθμός βημάτων μπορεί να αυξηθεί.

Λήμμα 7. *Θεωρήστε γραμμικό πλέγμα με N επεξεργαστές στο οποίο κάθε κόμβος έχει αυθαίρετο αριθμό πακέτων κάθε ένα από τα οποία κατευθύνεται σε διαφορετικό κόμβο. Αν σε κάθε κόμβο, δίνεται προτεραιότητα στο πακέτο που πρόκειται να ταξιδέψει μακρύτερα, τότε ο άπληστος αλγόριθμος απαιτεί $N-1$ βήματα.*

Απόδειξη. Χωρίς βλάβη της γενικότητας, επικεντρώνουμε την προσοχή μας στα πακέτα που κινούνται προς τα δεξιά. Θα δείξουμε ότι για κάθε i, κάθε πακέτο που κατευθύνεται σε κάποιον από τους i δεξιότερους κόμβους φθάνει στον προορισμό του σε $N-1$ βήματα.

Εστω i. Θεωρούμε τα πακέτα που κατευθύνονται στους i δεξιότερους κόμβους. Τα πακέτα αυτά τα ονομάζουμε πακέτα προτεραιότητας και έχουν την ιδιότητα ότι δεν καθυστερούνται από τα υπόλοιπα. Θεωρήστε το δεξιότερο πακέτο προτεραιότητας. Δεν θα καθυστερηθεί από κανένα άλλο πακέτο και θα

φτάσει στον προορισμό του σε $N-i$ βήματα. Θεωρήστε το δεύτερο δεξιότερο πακέτο προτεραιότητας. Εκτός από το πρώτο βήμα, δεν καθυστερείται άλλο και φθάνει στον προορισμό του σε $N-i+1$ βήματα. Συνεχίζοντας το επιχείρημα, έχουμε ότι το i–οστό δεξιότερο πακέτο προτεραιότητας, μετά το βήμα $i-1$ (αν υπάρχει) δεν καθεστερείται άλλο και θα φθάσει στους i δεξιότερους κόμβους σε $N-1$ βήματα. □

Συνδυάζοντας το αποτέλεσμα για την δρομολόγηση στο γραμμικό πλέγμα και το λήμμα 2, έχουμε ότι όλα τα πακέτα δρομολογούνται σε $2\sqrt{N}-2$ βήματα. Αυτός ο χρόνος είναι βέλτιστος μιας και κάποιο πακέτο μπορεί να χρειαστεί να ταξιδέψει όλη τη διάμετρο του δικτύου για να φθάσει στον κόμβο προορισμού. Το μειονέκτημα του αλγορίθμου είναι ότι απαιτούνται ουρές μεγάλου μεγέθους σε κάθε επεξεργαστή για την προσωρινή αποθήκευση των πακέτων όπως φαίνεται από το παρακάτω λήμμα.

Λήμμα 8. *Το μέγιστο μέγεθος της ουράς ενός κόμβου είναι τουλάχιστον* $O(\sqrt{N})$.

Απόδειξη. Έστω ότι τα πακέτα που είναι αρχικά αποθηκευμένα στους κόμβους $(1,2),...,\left(1,\frac{\sqrt{N}}{3}\right)$ και $(2,1),...,\left(2,\frac{2\sqrt{N}}{3}-1\right)$ κατευθύνονται στους κόμβους $\left(3,\frac{\sqrt{N}}{3}\right),...,\left(\sqrt{N},\frac{\sqrt{N}}{3}\right)$. Στο βήμα $\frac{\sqrt{N}}{3}-1$, όλα αυτά τα $\sqrt{N}-2$ πακέτα έχουν φθάσει στον επεξεργαστή $\left(2,\frac{\sqrt{N}}{3}\right)$ και μόνο $\frac{\sqrt{N}}{3}-1$ έχουν περάσει. Οπότε το μέγεθος της ουράς στον επεξεργαστή $\left(2,\frac{\sqrt{N}}{3}\right)$ είναι $\frac{2\sqrt{N}}{3}-1$. □

4.3 Αποδοτικοί ντετερμινιστικοί αλγόριθμοι

Στην παράγραφο αυτή, εξετάζουμε αλγορίθμους που απαιτούν ουρές μικρού (σταθερού) μεγέθους.

Φάση 1. Ταξινομούμε τα πακέτα κατά στήλες ως προς τη στήλη προορισμού των πακέτων. Η διαδικασία αυτή κοστίζει $4\sqrt{N}+o(\sqrt{N})$ βήματα: $3\sqrt{N}+o(\sqrt{N})$ για να ταξινομήσουμε σε σειρά βουστροφηδόν και $\sqrt{N}-1$ επιπλέον βήματα για να πάρουμε ταξινόμηση κατά στήλες. Αν αρχικά έχουμε ακριβώς N πακέτα, τότε ο αλγόριθμος τερματίζει. Αλλιώς συνεχίζει στις επόμενες φάσεις.

Φάση 2. Στέλνουμε τα πακέτα στη σωστή στήλη χρησιμοποιώντας τον άπληστο αλγόριθμο στο γραμμικό πλέγμα. Η διαδικασία αυτή κοστίζει άλλα $\sqrt{N}-1$ βήματα.

Φάση 3. Στέλνουμε τα πακέτα στη σωστή γραμμή χρησιμοποιώντας πάλι τον άπληστο αλγόριθμο για $\sqrt{N}-1$ επιπλέον βήματα. Συνολικά ο αλγόριθμος απαιτεί $6\sqrt{N}+o(\sqrt{N})$ βήματα. Θα εξηγήσουμε γιατί δεν δημιουργούνται ουρές. Στη φάση 1 (ταξινόμηση), τα πακέτα διατάσσονται έτσι ώστε το πολύ ένα πακέτο σε κάθε σειρά να έχει σαν τελικό προορισμό μια συγκεκριμένη στήλη. Οπότε, μετά τη φάση 2, θα υπάρχει το πολύ ένα πακέτο σε κάθε κόμβο. Αυτό υποννοεί ότι δεν υπάρχει ανταγωνισμός για τις γραμμές στη φάση 3. Το τίμημα για τη μείωση του μεγέθους των ουρών είναι 300% απώλεια σε ταχύτητα. Στη συνέχεια, παρουσιάζουμε έναν αλγόριθμο με καλύτερη συμπεριφορά.

Φάση 1. Χωρίζουμε το πλέγμα σε q^2 υποπλέγματα μεγέθους $\frac{\sqrt{N}}{q} \times \frac{\sqrt{N}}{q}$ και διατάσσουμε τα πακέτα σε κάθε υποπλέγμα κατά στήλες ως προς τη στήλη προορισμού. Χρησιμοποιώντας τον αλγόριθμο ταξινόμησης, η φάση 1 απαιτεί $4\frac{\sqrt{N}}{q}+o\left(\frac{\sqrt{N}}{q}\right)$ βήματα.

Φάση 2. Χρησιμοποιούμε τον άπληστο αλγόριθμο, δηλ. $2\sqrt{N}-2$ επιπλέον βήματα. Για να απαντήσουμε στο ερώτημα ποιο είναι το μέγιστο μέγεθος των ουρών, παρατηρούμε ότι είναι το πολύ ο μέγιστος αριθμός πακέτων σε κάθε γραμμή στο τέλος της πρώτης φάσης που κατευθύνονται στην ίδια στήλη. Εστω ένας κόμβος (i,j) και $B_1, B_2, ..., B_q$ τα q υποπλέγματα που περιέχουν τη σειρά i. Αν το B_k περιέχει r_k πακέτα που κατευθύνονται στην i–οστή στήλη, τότε το πολύ $\lceil r_k\frac{q}{\sqrt{N}}\rceil \leq 1+(r_k-1)\frac{q}{\sqrt{N}}$ πακέτα βρίσκονται στη γραμμή i μετά το τέλος της φάσης 1. Επειδή $r_k \leq \sqrt{N}$, ο αριθμός των πακέτων που κατευθύνονται στη στήλη j και είναι στη γραμμή i μετά την πρώτη φάση, είναι το πολύ

$$\sum_{k=1}^{q}\left(1+(r_k-1)\frac{q}{\sqrt{N}}\right) \leq q-\frac{q^2}{\sqrt{N}}+\frac{q}{\sqrt{N}}\sum_{k=1}^{q} r_k < 2q$$

οπότε το μέγιστο μέγεθος ουράς είναι το πολύ $2q-1$.

4.4 Μια διαφορετική λύση

Στην παράγραφο αυτή, παρουσιάζουμε έναν off–line αλγόριθμο δρομολόγησης. Οι off–line αλγόριθμοι είναι χρήσιμοι όταν το πρόβλημα δρομολόγησης είναι γνωστό από την αρχή (κάτι που συνήθως δεν ισχύει στην πράξη).

Φάση 1. Εναλλάσουμε τα πακέτα κάθε στήλης έτσι ώστε σε κάθε γραμμή, το πολύ ένα πακέτο να έχει προορισμό κάποια στήλη.

Φάση 2. Στέλνουμε τα πακέτα στη σωστή στήλη.

Φάση 3. Στέλνουμε τα πακέτα στη σωστή γραμμή. Οι δυο τελευταίες φάσεις ολοκληρώνονται σε $\sqrt{N}-1$ βήματα η κάθε μια, χρησιμοποιώντας τον άπληστο αλγόριθμο στο γραμμικό πλέγμα. Στη συνέχεια, Θα δείξουμε ότι η πρώτη φάση μπορεί να ολοκληρωθεί σε $\sqrt{N}-1$ βήματα, καταλήγοντας σε συνολικό χρόνο $3\sqrt{N}-3$ βημάτων για τον αλγόριθμο. Αρχικά διατυπώνουμε χωρίς απόδειξη ένα σημαντικό θεώρημα της θεωρίας γράφων, το θεώρημα του Hall.

Θεώρημα 9. *Ενας διμερής γράφος έχει ένα πλήρες ταίριασμα αν και μόνο αν για κάθε σύνολο κορυφών $S \subseteq U$, ισχύει $|N(S)| \geq |S|$, όπου $N(S) \subseteq V$ η γειτονιά του S.*

Συμπέρασμα 10. *Κάθε r–κανονικός διμερής γράφος έχει ένα πλήρες ταίριασμα.*

Συμπέρασμα 11. *Εστω G ένας r–κανονικός διμερής γράφος. Είναι δυνατό να αριθμήσουμε κάθε ακμή με έναν ακέραιο από το διάστημα $[1,r]$ έτσι ώστε ακμές γειτονικές στον ίδιο κόμβο να αριθμούνται με διαφορετικό ακέραιο.*

Υποθέστε ότι έχουμε ένα 1–1 πρόβλημα δρομολόγησης σε ένα δισδιάστατο πλέγμα διαστάσεων $r \times s$. Ορίζω τον διμερή γράφο $G=(U,V,E)$ με $2s$ κορυφές $U=\{u_1,...,u_s\}$ και $V=\{v_1,...,v_s\}$ και rs ακμές. Για κάθε πακέτο p_k με αρχική στήλη i_k και στήλη προορισμού j_k, ο G περιέχει μια ακμή $e_k=(u_{i_k},v_{j_k})$. Από το συμπέρασμα 11, εφόσον ο G είναι διμερής και r–κανονικός, μπορούμε να δώσουμε σε κάθε ακμή αριθμούς από το διάστημα $[1,r]$ έτσι ώστε δυο ακμές που μοιράζονται τον ίδιο κόμβο να μην έχουν τον ίδιο αριθμό. Στη φάση 1, κάθε πακέτο p στέλνεται στην σειρά l_p όπου l_p ο αριθμός της αντίστοιχης ακμής του διμερούς γράφου. Παρατηρούμε ότι τα πακέτα που ξεκινούν από την i–οστή στήλη, θα σταλούν σε διαφορετικές σειρές στο τέλος της πρώτης φάσης. Επίσης, τα πακέτα που κατευθύνονται στην j–οστή στήλη, θα σταλούν σε διαφορετικές σειρές.

4.5 Ασκήσεις

1. Δείξτε πως μπορεί να δρομολογηθεί οποιοδήποτε ζεύγος μεταθέσεων σε N βήματα σε γραμμικό πλέγμα με N επεξεργαστές. Επίσης, δείξτε πως μπορούν να δρομολογηθούν k μεταθέσεις σε $\lceil \frac{k}{2} \rceil N$ βήματα σε γραμμικό πλέγμα με N επεξεργαστές. Υπόδειξη: Χρησιμοποιήστε τον αλγόριθμο ταξινόμησης στο γραμμικό πλέγμα.

2. Δείξτε ότι ο βασικός αλγόριθμος μπορεί να απαιτήσει περισσότερα από $2\sqrt{N}$ βήματα αν χρησιμοποιηθεί λάθος πρωτόκολλο λύσης ανταγωνισμού, αλλά ποτέ μεγαλύτερο από $O(\sqrt{N})$ αν το μέγεθος των ουρών μπορεί να μεγαλώσει απεριόριστα.

3. Δείξτε ότι το μέγιστο μέγεθος ουράς του βασικού άπληστου αλγορίθμου είναι $\frac{2\sqrt{N}}{3}$.

4. Δείξτε ότι ο βασικός άπληστος αλγόριθμος εκτελείται σε το πολύ $2\lceil\frac{\sqrt{N}}{2}\rceil$ βήματα σε διασδιάστατο δίκτυο τορους $\sqrt{N}\times\sqrt{N}$. Σημειώνουμε ότι, στο δίκτυο τορους, κάθε πακέτο ακολουθεί το ελάχιστο μονοπάτι πρώτα προς τη σωστή στήλη και έπειτα προς τη σωστή γραμμή.

5. Σχεδιάστε ένα ντετερμινιστικό αλγόριθμο με σταθερό μέγεθος ουρών, που δρομολογεί κάθε πακέτο σε $O(d)$ βήματα σε δισδιάστατο πλέγμα, όπου d η μέγιστη απόσταση που πρέπει να ταξιδέψει ένα πακέτο.

6. Αν σε κάθε κόμβο ενός δικτύου επεξεργαστών, το πολύ k πακέτα έχουν αυτόν το κόμβο σαν αρχή ή προορισμό, τότε το πρόβλημα μπορεί να διασπαστεί σε k 1–1 προβλήματα δρομολόγησης. Υπόδειξη: Χρησιμοποιήστε το θεώρημα του Hall.

7. Δείξτε ότι αν το πολύ k πακέτα αρχίζουν ή προορίζονται σε κάθε κόμβο ενός πλέγματος διαστάσεων $\sqrt{N}\times\sqrt{N}$, τότε το πρόβλημα δρομολόγησης μπορεί να λυθεί σε $3\lceil\frac{k}{4}\rceil\sqrt{N}$ βήματα.

8. Δείξτε πως μπορεί να λυθεί ένα πρόβλημα δρομολόγησης μεταθέσεων σε πλέγμα διαστάσεων $\sqrt{N}\times\sqrt{N}/2$ σε $2\sqrt{N}-3$ βήματα με ουρές μεγέθους 1. Χρησιμοποιήστε το αποτέλεσμα αυτό για να λύσετε το ίδιο πρόβλημα σε πλέγμα διαστάσεων $\sqrt{N}\times\sqrt{N}$ σε $2.5\sqrt{N}$ βήματα χρησιμοποιώντας ουρές μεγέθους 2.

9. Δείξτε ότι αν κάθε κόμβος έχει αρχικά ένα πακέτο και αν το πολύ m πακέτα έχουν τον ίδιο προορισμό, τότε ο βασικός άπληστος αλγόριθμος μπορεί να απαιτήσει $O(\min\{N, m\sqrt{N}\})$ βήματα.

10. Δείξτε ότι οποιοσδήποτε αλγόριθμος δρομολόγησης μπορεί να απαιτήσει $\Omega(m\sqrt{N})$ βήματα για ένα m–1 πρόβλημα δρομολόγησης N πακέτων σε πλέγμα διαστάσεων $\sqrt{N}\times\sqrt{N}$.

11. Σχεδιάστε αλγόριθμο που χρησιμοποιεί ουρές σταθερού μεγέθους για τη λύση του m–1 προβλήματος δρομολόγησης N πακέτων σε πλέγμα διαστάσεων $\sqrt{N}\times\sqrt{N}$.

Κεφάλαιο 5

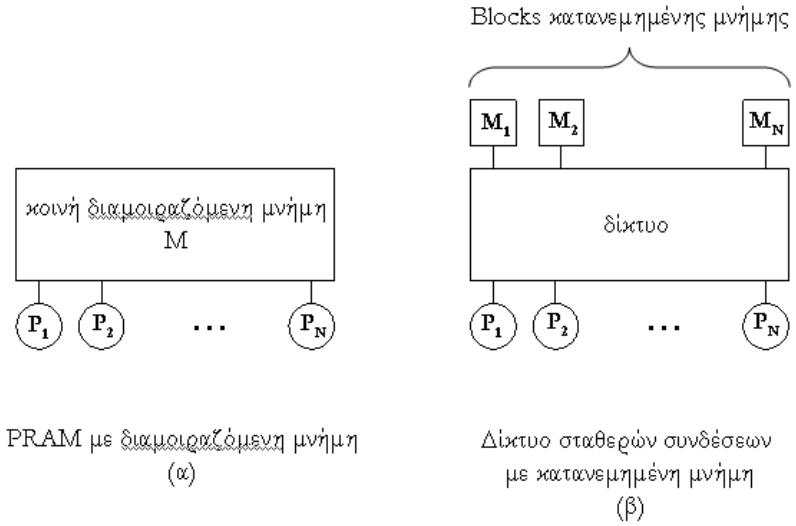
Το μοντέλο PRAM

5.1 Παράλληλης Μηχανής Τυχαίας Προσπέλασης

Το μοντέλο Παράλληλης Μηχανής Τυχαίας Προσπέλασης (Parallel Random Access Machine - PRAM) είναι σίγουρα το πιο δημοφιλές μοντέλο ενός παράλληλου υπολογιστή. Παρά το την ύπραξη αρκετών διαφορετικών τύπων PRAM μοντέλων, το βασικό τους χαρακτηριστικό είναι το ότι αποτελούνται από πολλούς ανεξάρτητους επεξεργαστές που διαμοιράζονται μια κοινή μνήμη. Η μία κοινή διαμοιραζόμενη μνήμη των PRAM είναι το χαρακτηριστικό που ουσιαστικά τις διαφοροποιεί σε πολύ μεγάλο βαθμό από τις παράλληλες μηχανές που ακολουθούν το μοντέλο των δικτύων σταθερών συνδέσεων (Fixed Connection Network Model) όπου - όπως και σε παράλληλους υπολογιστές μεγάλης κλίμακας - η μνήμη είναι κατανεμημένη σε διάφορες τοποθεσίες που διασυνδέονται μεταξύ τους μέσω του δικτύου.

Στο σχήμα 5.1παρουσιάζεται τη βασική αρχιτεκτονική μιας PRAM. Υπάρχουν N επεξεραγαστές $P_0, P_1, P_2 \cdots P_N$ που χρησιμοποιούν σαν αποθηκευτικό χώρο μια διαμοιραζόμενη μνήμη. Όλοι οι επεξεργαστές μπορούν να διαβάσουν από τη μνήμη ή να γράψουν σ΄ αυτή ταυτόχρονα. Επίσης, οι επεξεργαστές μπορούν να πραγματοποιήσουν ποικίλες αριθμητικές και λογικές λειτουργίες παράλληλα.

Στο PRAM μοντέλο, η βασική υπόθεση, όσον αφορά στην απόδοση των αλγορίθμων, είναι το γεγονός ότι ο χρόνος εκτέλεσης μπορεί να θεωρηθεί σαν τον αριθμό των προσπελάσεων του αλγορίθμου στην παράλληλη μνήμη. Αυτή η παραδοχή αποτελεί μια γενίκευση του κανονικού PRAM μοντέλου, στο οποίο ο αριθμός των προσπελάσεων στη μνήμη είναι ένα καλό μέτρο του χρόνου εκτέλεσης. Επιπλέον, αυτή η υπόθεση σρησιμοποιείται γενικότερα στη μελέτη παράλληλων αλγορίθμων, παρά το γεγονός ότι οι παράλληλοι υπολογιστές δε



Σχήμα 5.1: PRAM με διαμοιραζόμενη μνήμη (α) και δίκτυο σταθερών συνδέσεων με κατανεμημένη μνήμη (β). Στο μοντέλο PRAM, κάθε επεξεργαστής μπορεί σε κάθε βήμα να προσπελάσει κάθε θέση της διαμοιραζόμενης μνήμης. Στο πιο ρεαλιστικό μοντέλο του δικτύου σταθερών συνδέσεων, υπάρχουν καθυστερήσεις που σχετίζονται με τις απομακρυσμένες προσπελάσεις στη μνήμη και υπάρχουν όρια στον αριθμό των προσπελάσεων στη μνήμη που μπορούν να πραγματοποιηθούν σε κάθε block μνήμης σε κάθε βήμα.

μπορούν να προσπελάσουν παράλληλα τη μνήμη σε μοναδιαίο χρόνο αφού ο χρόνος που απαιτείται για προσπέλαση στη μνήμη αυξάνεται ανάλογα με τον αριθμό των επεξεργαστών σ΄ ένα παράλληλο υπολογιστή.

Παρολαυτά, για παράλληλους αλγόριθμους που πραγματοποιούν προσπελάσεις σε δεδομένα με τυχαίο τρόπο, η υπόθεση λειτουργιών μνήμης σταθερού χρόνου μπορεί να τεκμηριωθεί. Οι πραγματικές παράλληλες μηχανές διαθέτουν τυπικά ένα δίκτυο επικοινωνίας που μπορεί να υποστηρίξει μια μορφή καθολικής μνήμης. Η προσπέλαση δεδομένων μέσω του δικτύου είναι μια σχετικά αργή λειτουργία σε σύγκριση με άλλες λειτουργίες (π.χ. αριθμητικές πράξεις). Επομένως, η μέτρηση των προσπελάσεων στην παράλληλη μνήμη από δυο παράλληλους αλγορίθμους παρέχει με αρκετή ακρίβεια εκτίμηση της απόδοσής τους. Ο βασικός τρόπος με τον οποίο οι πραγματικές μηχανές δεν πετυχαίνουν το μοναδιαίο χρόνο προσπέλασης της PRAM είναι ότι κάποιες τεχνικές προσπέλασης είναι γρηγορότερες από άλλες.

Ο χρόνος εκτέλεσης ενός παράλληλου αλγόριθμου εξαρτάται από τον αριθμό των επεξεργαστών που εκτελούν τον αλγόριθμο καθώς επίσης και από το μέγεθος του προβλήματος που καλείται να επιλύσει ο αλγόριθμος. Κατά συνέπεια, στη γενική περίπτωση πρέπει να λαμβάνονται υπ΄ όψη οι μετρήσεις και του χρόνου και των επεξεργαστών κατά την ανάλυση του PRAM αλγόριθμου. Αυτό έρχεται σε αντίθεση με την ανάλυση ακολουθιακών αλγορίθμων κατά την οποία δίνεται έμφαση στο χρόνο. Τυπικά υπάρχει σχέση αντιστάθμισης ανάμεσα στον αριθμό των επεξεργαστών που χρησιμοποιούνται από έναν αλγόριθμο και στο χρόνο εκτέλεσής του.

5.2 Μοντέλα PRAM και διαμοιραζόμενη μνήμη

Μια PRAM με N επεξεργαστές, $P_1, P_2, \cdots, P_N$, διαθέτει μια κοινή διαμοιραζόμενη μνήμη και κάθε επεξεργαστής διαθέτει το δικό του τοπικό έλεγχο και τη δική του τοπική μνήμη. Η διαμοιραζόμενη μνήμη αποτελείται από M περιοχές με ίδιο μέγεθος, κάθε μία από τις οποίες έχει δική της ανεξάρτητη διεύθυνση. Σε κάθε βήμα του υπολογισμού μιας PRAM, κάθε επεξεργαστής μπορεί να προσπελάσει - δηλαδή, μπορεί να διαβάσει ή να γράψει - οποιαδήποτε αυθαίρετη θέση της διαμοιραζόμενης μνήμης και να εκετλέσει κάποιον υπολογισμό με βάση τον τοπικό του έλεγχο και την τοπική του μνήμη.

Σε μια PRAM αποκλειστικής ανάγνωσης και εγγραφής (exclusive-read exclusive-write - EREW PRAM), οι προσπελάσεις στη μνήμη που μπορούν να πραγματοποιήσουν οι N επεξεργαστές είναι περιορισμένες έτσι ώστε το πολύ ένας επεξεργαστής να μπορεί να διαβάσει ή να γράψει σε οποιαδήποτε θέση της κοινής διαμοιραζόμενης μνήμης σε κάθε χρονικό βήμα. Σε μια PRAM ταυτό-

χρονης ανάγνωσης και αποκλειστικής εγγραφής (concurrent-read exclusive-write - CREW PRAM), πολλοί επεξεργαστές μπορούν να διαβάζουν ταυτόχρονα από την ίδια θέση της κοινής διαμοιραζόμενης μνήμης, αλλά μόνο ένας επεξεργαστής μπορεί να γράψει σε κάποια θέση σε κάθε χρονικό βήμα. Σε μια PRAM ταυτόχρονης εγγραφής (concurrent-write - CR PRAM), πολλοί επεξεργαστές μπορούν να γράψουν σε μια συγκεκριμένη θέση της κοινής διαμοιραζόμενης μνήμης ταυτόχρονα, αλλά πρέπει να υπάρχει κάποια μέθοδος για να διαχειρίζεται με αυθαίρετο τρόπο της συγκρούσεις. Για τη διαχείριση των συγκρούσεων κατά την ταυτόχρονη εγγραφή, έχει αναπτυχθεί μια σειρά σχετικών πρωτοκόλλων. Στο ασθενές CR PRAM μοντέλο, επιτρέπονται ταυτόχρονες εγγραφές μόνον αν όλοι οι επεξεργαστές επιθυμούν να γράψουν 0 στη συγκεκριμένη θέση της διαμοιραζόμενης μνήμης. Στο κοινό CR PRAM μοντέλο, επιτρέπονται ταυτόχρονες εγγραφές μόνον αν όλοι οι επεξεργαστές γράφουν την ίδια τιμή στη συγκεκριμένη θέση της διαμοιραζόμενης μνήμης. Στο αυθαίρετο CR PRAM μοντέλο, μια σύγκρουση κατά τη διάρκεια προσπάθειας ταυτόχρονης εγγραφής επιλύεται με αυθαίρετη επιλογή μιας από τις τιμές που πρόκειται να εγγραφούν στη συγκεκριμένη θέση της διαμοιραζόμενης μνήμης. Στο CR PRAM μοντέλο που υιοθετεί προτεραιότητες, στην περίπτωση σύγκρουσης ο επεξεργαστής που γράφει τελικά την τιμή του στη συγκεκριμένη θέση της διαμοιραζόμενης μνήμης είναι αυτός με το μικρότερο δείκτη ταξινόμησης. Τέλος, στο συνδυαστικό μοντέλο CR PRAM, οι τιμές που οι επεξεργαστές προσπαθούν να γράψουν στη συγκεκριμένη θέση της διαμοιραζόμενης μνήμης συνδυάζονται με χρήση μιας συσχετιστικής και μεταβατικής πράξης.

Κατάντιστοιχία με τα παραπάνω, ένας αλγόριθμος ταυτόχρονης ανάγνωσης (concurrent read) είναι ένας PRAM αλγόριθμος κατά την εκτέλεση του οποίου πολλοί επεξεργαστές μπορούν να διαβάσουν από την ίδια θέση της κοινής μνήμης την ίδια χρονική στιγμή. Ένας αλγόριθμος αποκλειστικής ανάγνωσης (exclusive read) είναι ένας PRAM αλγόριθμος που δεν επιτρέπει σε δύο επεξεργαστές να διαβάσουν ταυτόχρονα από την ίδια θέση της μνήμης. Η ίδια διαφοροποίηση πραγματοποιείται και για την εγγραφή δεδομένων στη μνήμη οπότε προκύπτουν αντίστοιχα PRAM αλγόριθμοι concurrent write - CW και exclusive write - EW. Συχνά χρησιμοποιούμενα ακρωνύμια για τους παραπάνω τύπους αλγορίθμων είναι τα EREW και CRCW. Μια PRAM που υποστηρίζει μόνο EREW αλγορίθμους καλείται EREW PRAM και μια που υποστηρίζει CRCW αλγορίθμους καλείται CRCW PRAM. Μια CRCW PRAM μπορεί φυσικά να εκτελεί EREW αλγορίθμους αλλά μια EREW PRAM δεν δίνει τη δυνατότητα ταυτόχρονης προσπέλασης της μνήμης που απαιτούν οι CRCW αλγόριθμοι.

Όσον αφορά στους δύο άλλους τύπους αλγορίθμων (CREW, ERCW) περισσότερη έμφαση έχει δοθεί στον CREW. Όμως, πρακτικά, η υποστήριξ-

η ταυτόχρονων εγγραφών δεν είναι δυσκολότερη από την υποστήριξη ταυτόχρονων αναγνώσεων. Γενικά, αλγόριθμοι που υποστηρίζουν ταυτόχρονες είτε εγγραφές είτε αναγνώσεις θεωρούνται CRCW. Όταν πολλοί επεξεργαστές πραγματοποιούν ταυτόχρονες εγγραφές στην ίδια θέση μνήμης σ΄ έναν CRCW αλγόριθμο το αποτέλεσμα δεν είναι εξ΄ ορισμού πλήρως ορισμένο. Συνήθως, υιοθετείται η χρήση του common (κοινού) CRCW μοντέλου, όπου όταν πολλοί επεξεργαστές γράφουν στην ίδια θέση της μνήμης πρέπει όλοι να γράφουν την ίδια τιμή. Στη βιβλιογραφία αναφέρονται αρκετοί διαφορετικοί τύποι PRAM μηχανών που χειρίζονται το πρόβλημα με διαφορετικές παραδοχές: - τυχαιότητα: μια τυχαία τιμή από όσες γράφονται τελικά αποθηκεύεται, - προτεραιότητα: αποθηκεύεται η τιμή την οποία έγραψε ο επεηεργαστής με τη μικρότερη σειρά, - συνδυασμός: αποθηκεύεται κάποις συγκεκριμένος συνδυασμός των τιμών που γράφτηκαν από κάθε επεξεργαστή. Αυτός ο συνδυασμός είναι συνήθως μια συνάρτηση όπως άθροισμα (άθροισε όλες τις τιμές που αποθήκευσες) ή max (εύρεση μέγιστων τιμών - αποθήκευσε τη μέγιστη τιμή από όσες γράφτηκαν).

Οι PRAM αποτελούν ένα πολύ καλό αφηρημένο μοντέλο για τη μελέτη του παραλληλισμού. Δεν υπάρχουν καλώδια/συνδέσεις και κατά συνέπεια δεν τίθεται θέμα για το πώς τα σωστά δεδομένα θα είναι στο σωστό σημείο τη σωστή χρονική στιγμή. Με λίγα λόγια, το μοντέλο PRAM παρακάμπτει όλες τις λεπτομέρειες σχετικά με την υλοποίηση ενός παράλληλου αλγορίθμου σε μια παράλληλη μηχανή που πολλές φορές προκαλούν σύγχυση και κατά συνέπεια αποτελεί ένα ιδανικό μοντέλο για παράλληλο προγραμματισμό.

Όμως, αυτά τα χαρακτηριστικά που κάνουν το μοντέλο PRAM ελκυστικό από προγραμματιστική άποψη, το καθιστούν δύσχρηστο από κατασκευαστική πλευρά. Συγκεκριμένα, μια κοινή διαμοιραζόμενη μνήμη αποτελεί μια αφαίρεση που δύσκολα υλοποιείται με υλικό. Για παράδειγμα, το υλικό (hardware) που απαιτείται την υλοποίηση μιας EREW PRAM είναι σχετικά απλό και κατά συνέπεια γρήγορο μιας και δεν υπάρχει ανάγκη χειρισμού συγκρουόμενων αναγνώσεων και εγγραφών από και προς τη μνήμη. Μια CREW PRAM απαιτεί πολυπλοκότερο υλικό (hardware) προκειμένου η υπόθεση μοναδιαίου χρόνου να οδηγήσει σε αξιόπιστες μετρήσεις της απόδοσης, αλλά παρέχει ένα προγραμματστικό μοντέλο αρκετά ισχυρότερο από ό,τι μια EREW PRAM. Το γεγονός αυτό έχει σα συνέπεια μεγάλης κλίμακας PRAM να υλοποιούνται στην πράξη με χρήση δικτύων σταθερών συνδέσεων (όπως είναι για παράδειγμα τα δίκτυα πεταλούδας) και εξοιμοίωσης μέσω αυτών των μοντέλων PRAM. Η κοινή διαμοιραζόμενη μνήμη κατανέμεται σε ισοεμεγέθη block στους κόμβους του δικτύου σταθερών συνδέσεων και για να προσπελάσει ένας επεξεργαστής κάποια θέση της μνήμης πρέπει να αποστείλει μέσω του δικτύου ένα πακέτο στον επεξεργαστή που διαθέτει τη συγκεκριμένη θέση μνήμης. Τότε, στην περίπτωση ανάγνωσης, επιστρέφεται μέσω του δικτύου στον επεξεργαστή που

πραγματοποίησε το αίτημα ανάγνωσης ένα πακέτο που περιέχει τα δεδομένα. Η δρομολόγηση του πακέτου επιτυγχάνεται με χρήση γνωστών αλγορίθμων δρομολόγησης.

Θεωρώντας ότι οι M μονάδες μιας κοινής διαμοιραζόμενης μνήμης είναι κατανεμημένες σε N επεξεργαστές του δικτύου, αν $M \gg N$, η περιεκτικότητα της μνήμης μπορεί να αποτελέσει σημαντικό πρόβλημα ακόμα και στην περίπτωση μόνο αποκλειστικών αναγνώσεων και εγγραφών. Για παράδειγμα, αν $M \geq N^2$, τότε καθένας από τους N επεξεργαστές μπορεί να επιθυμεί να προσπελάσει μια διαφορετική θέση μνήμης στο ίδιο block. Τέτοιου είδους προβλήματα μπορούν να επιλυθούν μέσω πραγματοποίησης συνδυασμών οι οποίοι όμως μπορούν να είναι αποδοτικοί μόνον αν ο χρόνος που απαιτείται για τη μετάδοση ενός μεγάλου πακέτου που προκύπτει από συνδυασμό δε σχετίζεται με το μέγεθός του. Αντίθετα, αν το μέγεθος του πακέτου έχει ουσιώδη σημασία (όπως συμβαίνει συνήθως) ή αν δεν επιτρέπεται συνδυασμός, θα πρέπει να ακολουθηθεί άλλη εναλλακτική προσέγγιση.

5.3 Συγχρονισμός και Έλεγχος

Οι αλγόριθμοι προκειμένου να δουλεύουν σωστά πρέπει να έχουν σε μεγάλο βαθμό συγχρονισμό. Επίσης, οι επεξεργαστές πρέπει συχνά να ανιχνεύουν σε αλγόριθμους συνθήκες τερματισμού επανάληψης που εξαρτώνται από την κατάσταση όλων των επεξεργαστών.

Πολλοί πραγματικοί παράλληλοι επεξεργαστές συνδέονται ώστε να δημιουργούν ένα δίκτυο ελέγχου που συμβάλει στο συγχρονισμό και στην ανίχνευση συνθηκών τερματισμού. Το δίκτυο ελέγχου μπορεί να ανιχνεύει συνθήκες τερματισμού τόσο γρήγορα όσο ένα δίκτυο δρομολόγησης μπορεί να υλοποιήσει αναφορές στη μνήμη (global memory).

Θεωρούμε ότι υπάρχει εξ΄ αρχής συγχρονισμός μεταξύ των επεξεργαστών, δηλαδή οι επεξεργαστές εκτελούν τις ίδιες εντολές την ίδια χρονική στιγμή και κανένας δεν προηγείται ή έπεται κάποιου άλλου.

Για την ανίχνευση του τερματισμού παράλληλων βρόγχων που εξαρτώνται από την κατάσταση όλων των επεξεργαστών, υποθέτουμε ότι το δίκτυο ελέγχου μπορεί να ελέγξει μια τέτοια συνθήκη σε σταθερό χρόνο. Στη βιβλιογραφία, κάποια EREW PRAM μοντέλα δεν υιοθετούν αυτή την υπόθεση και ο χρόνος (λογαριθμικός) που απαιτείται για τον έλεγχο των συνθηκών ανακύκλωσης πρέπει να συμπεριληφθεί στο συνολικό χρόνο εκτέλεσης. Οι CRCW PRAM δε χρειάζονται δίκτυο ελέγχου για τον έλεγχο τερματισμού μιας και μπορούν να ανιχνεύσουν τον τερματισμό παράλληλων ανακυκλώσεων σε σταθερό χρόνο μέσω των ταυτόχρονων εγγραφών.

Θεώρημα 12. *Αν ένας αλγόριθμος A τρέχει σε PRAM με p επεξεργαστές σε χρόνο t, τότε για κάθε $p' < p$ υπάρχει αλγόριθμος A' για το ίδιο πρόβλημα που τρέχει σε PRAM με p' επεξεργαστές και απαιτεί χρόνο $O(pt/p')$.*

Απόδειξη. Έστω μια αρίθμηση των χρονικών βημάτων του A, $1, 2, \cdots, t$. Ο αλγόριθμος A' εξομοιώνει την εκτέλεση κάθε χρονικού βήματος i, με $1 \leq i \leq t$ σε χρόνο $O([p/p'])$. Αφού υπάρχουν t βήματα, η συνολική εξομοίωση στοιχίζει $O([p/p']t) = O(pt/p')$ αφού $p' < p$.

Το έργο W που εκτελείται από τον αλγόριθμο A είναι pt ενώ αυτό που εκτελείται από τον A' είναι $(pt/p')p' = pt$. Κατά συνέπεια, η εξομοίωση δουλεύει αποτελεσματικά και επομένως αν ο αλγόριθμος A είναι ''καλός '' τότε και ο αλγόριθμος A' είναι ''καλός ''. □