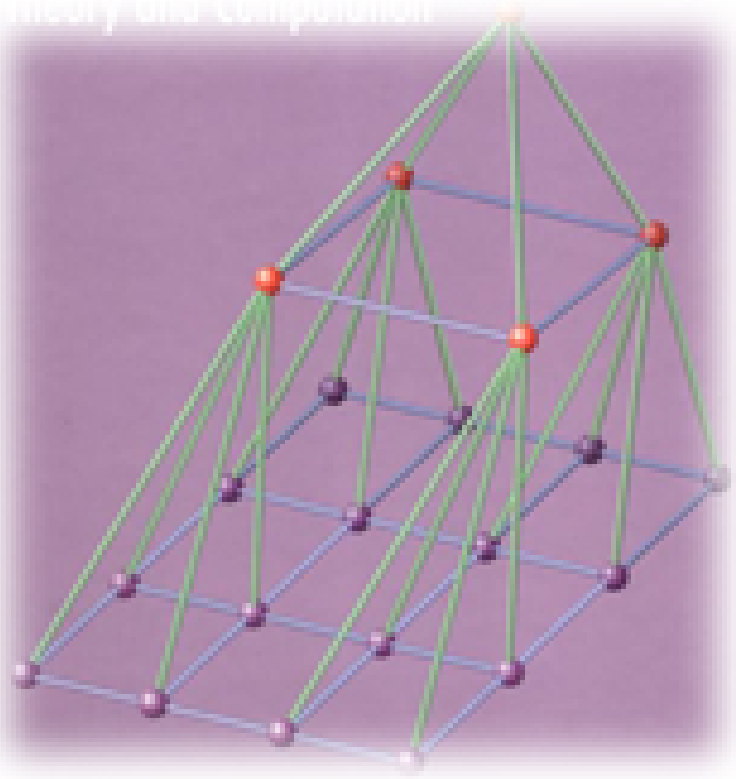
Εισαγωγή στους

ΠΑΡΑΛΛΗΛΟΥΣ ΥΠΟΛΟΓΙΣΜΟΥΣ

Ν. Μισυρλής



ΑΘΗΝΑ 2020

Ν. Μισυρλής

Τμήμα Πληροφορικής
και Τηλεπικοινωνιών

Εθνικό και Καποδιστριακό
Πανεπιστήμιο Αθηνών

Εισαγωγή στους
ΠΑΡΑΛΛΗΛΟΥΣ ΥΠΟΛΟΓΙΣΜΟΥΣ

- Θεωρία
- Αλγόριθμοι
- Παραδείγματα

ΑΘΗΝΑ 2020

Περιεχόμενα

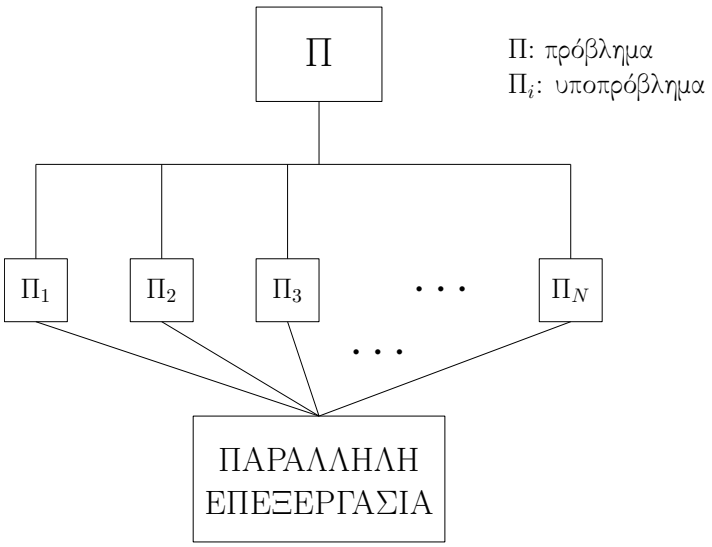
Κεφάλαιο 1

ΕΙΣΑΓΩΓΗ

1.1 Η ανάγκη για παράλληλους υπολογιστές

- Οι εφαρμογές απαιτούν χρονοβόρους υπολογισμούς:
  - σχεδιασμός αεροπλάνων
  - αριθμητική προσομοίωση σε προβλήματα Υπολογιστικής Δυναμικής των Ρευστών
- Η αύξηση των υπολογισμών θα κορεστεί.

Λύση: Χρήση της παραλληλίας



Σχήμα 1.1: Διάγραμμα block παράλληλης επεξεργασίας

Παράλληλος Αλγόριθμος

Μια μέθοδος λύσης για ένα δεδομένο πρόβλημα που πρόκειται να υλοποιηθεί σε ένα παράλληλο υπολογιστή

Παράλληλος Υπολογιστής

Ένας υπολογιστής με πολλές μονάδες επεξεργασίας ή επεξεργαστές

Μοντέλα Υπολογισμού

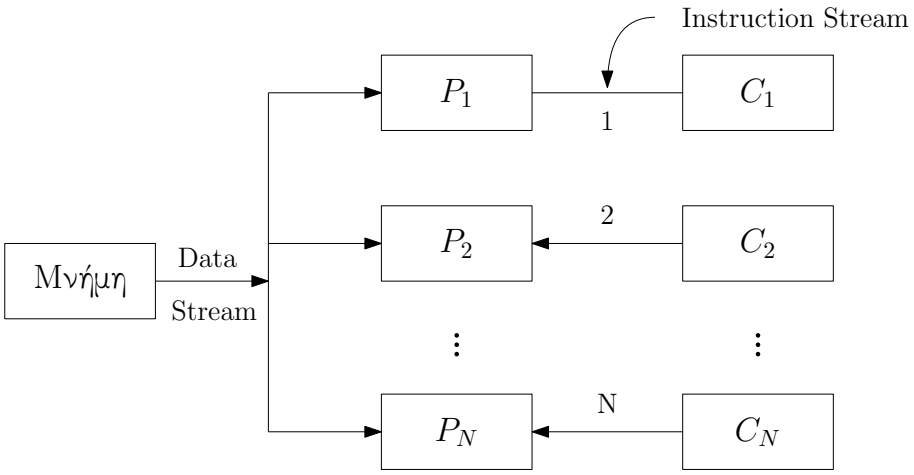
- SISD S: Single
- MISD I: Instruction
- SIMD D: Data
- MIMD M: Multiple

SISD

- John von Neumann 1940
- Ακολουθιακός (σειριακός) Αλγόριθμος

Σχήμα 1.2: Single Instruction Single Data

1.1.1 MISD



Σχήμα 1.3: Multiple Instruction Single Data

Παράδειγμα: Έλεγχος εάν ένας αριθμός z είναι πρώτος

Παράλληλος Αλγόριθμος

- Έστω ότι υπάρχουν τόσοι επεξεργαστές όσοι είναι και οι υποψήφιοι διαιρέτες του z.
- Έλεγχος από κάθε επεξεργαστή αν ο υποψήφιος διαιρέτης διαιρεί το z.

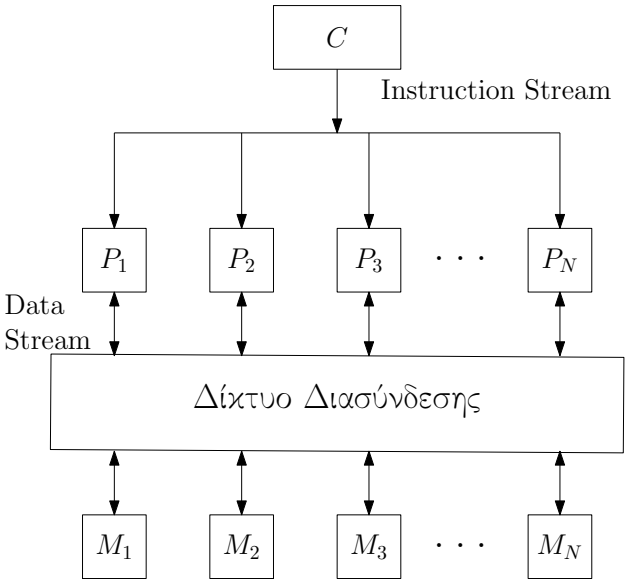
Παράδειγμα: Εύρεση του συνόλου (κατηγορίας) στην οποία ανήκει ένα αντικείμενο

Παράλληλος Αλγόριθμος

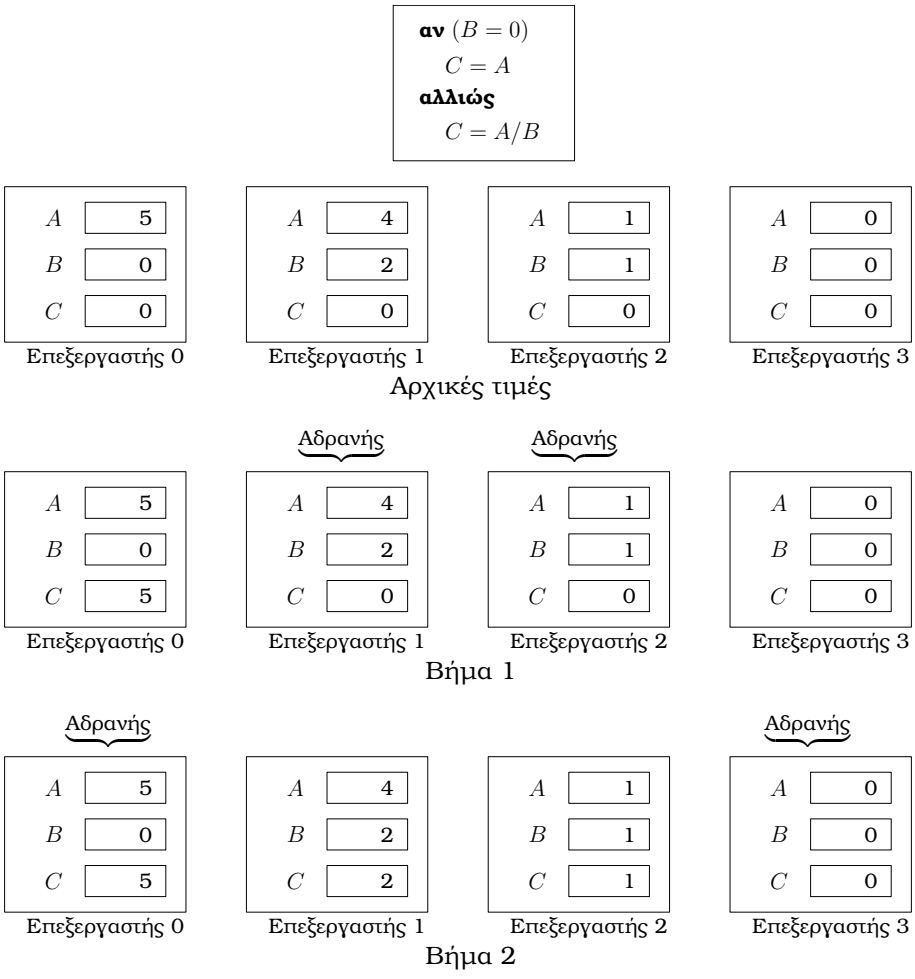
- Υποθέτουμε ότι υπάρχουν τόσοι επεξεργαστές όσες είναι οι κατηγορίες.
- Στέλνεται το αντικείμενο σε κάθε επεξεργαστή, ο οποίος ελέγχει αν ανήκει στην κατηγορία του.

Μειονέκτημα
• Ειδικής μορφής υπολογισμοί

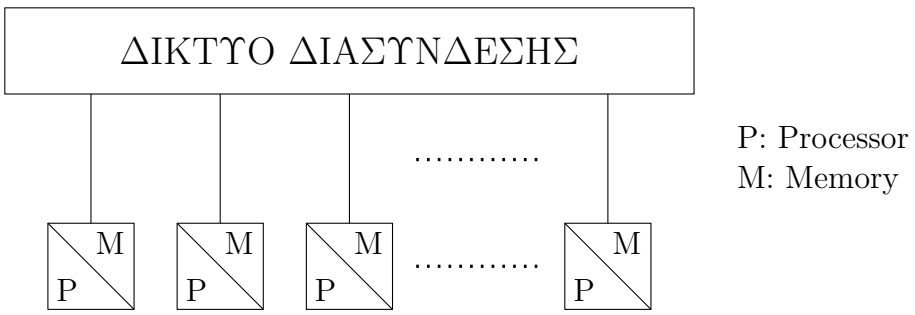
1.1.2 SIMD



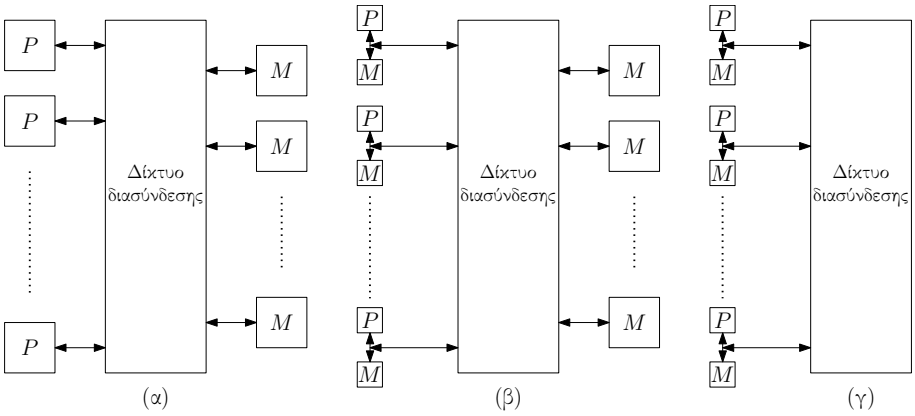
Σχήμα 1.4: Single Instruction Multiple Data



Σχήμα 1.5



Σχήμα 1.6: Μία τυπική αρχιτεκτονική διαβίβασης μηνυμάτων (message passing)



Σχήμα 1.7: Κοινή Μνήμη: (α) UMA, (β) NUMA με τοπική και ολική μνήμη, (γ) NUMA με τοπική μνήμη μόνο

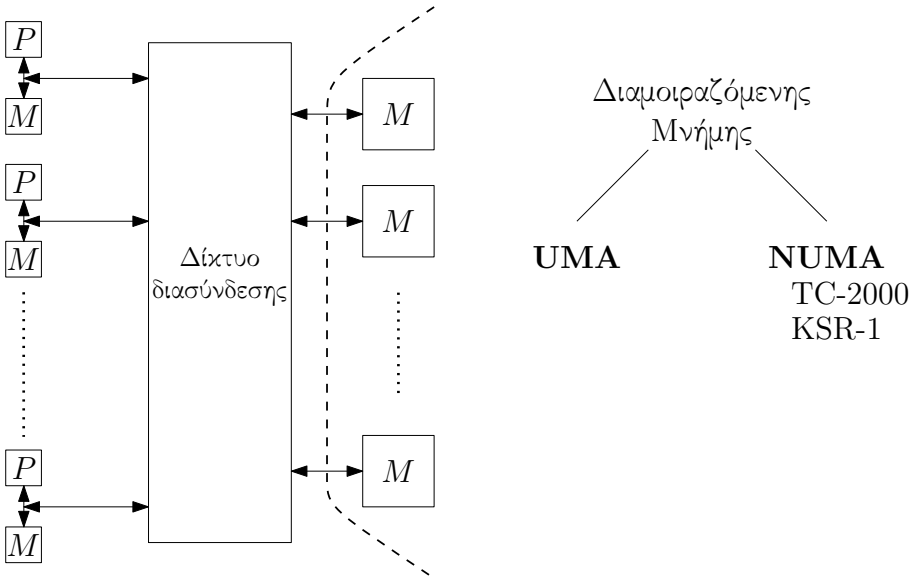
Διαμοιραζόμενης Μνήμης Υπολογιστές

- C. mmp
- NYU Ultracomputer

Μειονεκτήματα

- μεγάλο εύρος δικτύου διασύνδεσης
- αργή προσπέλαση μνήμης

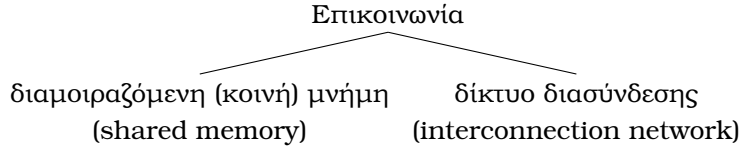
Θεραπεία $\rightarrow$ τοπική μνήμη



Σχήμα 1.8: Υπολογιστής Διαμοιραζόμενης Μνήμης

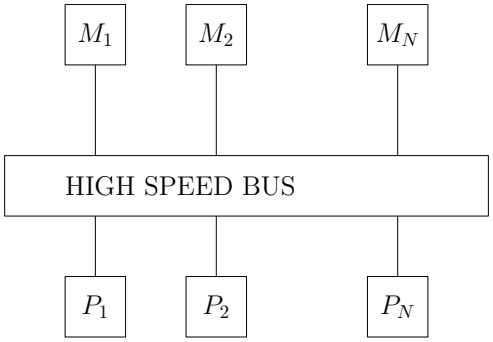
- Κάθε επεξεργαστής έχει τη δική του τοπική μνήμη (πρόγραμμα και δεδομένα)
- Οι επεξεργαστές λειτουργούν σύγχρονα: σε κάθε βήμα, όλοι οι επεξεργαστές εκτελούν την ίδια εντολή, ο κάθε ένας σε διαφορετικά δεδομένα
- Κατάλληλοι για παράλληλα-δεδομένα (data-parallel)

distributed memory ή/και message-passing



1.2 Διαμοιραζόμενης Μνήμης (PRAM) SIMD

- EREW
- CREW
- ERCW
- CRCW



Παράδειγμα

Να προσδιοριστεί αν ένα δεδομένο στοιχείο x βρίσκεται σε ένα αρχείο με n θέσεις.

Ακολουθιακός αλγόριθμος $\longrightarrow$ απαιτεί n βήματα.

Παράλληλος αλγόριθμος:
EREW SM SIMD με $N \leqslant n$ επεξεργαστές $P_1, P_2, \ldots P_N$.

- Μετάδοση (Broadcasting) του x στους επεξεργαστές
  1. P_1 διαβάζει το x και το μεταδίδει στον P_2
  2. Οι P_1 και P_2 μεταδίδουν ταυτόχρονα το x στους P_3 και P_4 αντίστοιχα, κ.ο.κ.
  3. Οι P_1, P_2, P_3 και P_4 μεταδίδουν ταυτόχρονα το x στους P_5, P_6, P_7 και P_8 αντίστοιχα, κ.ο.κ.

  Η μετάδοση του x σε όλους τους επεξεργαστές απαιτεί $\log_2 N$ βήματα

- Χωρισμός του αρχείου σε υποαρχεία με n/N στοιχεία

  P_i αναζητεί σε n/N στοιχεία άρα n/N βήματα

Σύνολο: $\log N + n/N$ βήματα

Αν χρησιμοποιηθεί μια λογική μεταβλητή F για τη δήλωση ότι βρέθηκε το x, τότε

$$\log N + (n/N) \log N$$

γιατί σε κάθε βήμα της αναζήτησης απαιτείται η γνώση της F σε <u>όλους</u> τους επεξεργαστές (broadcasting $\log N$).

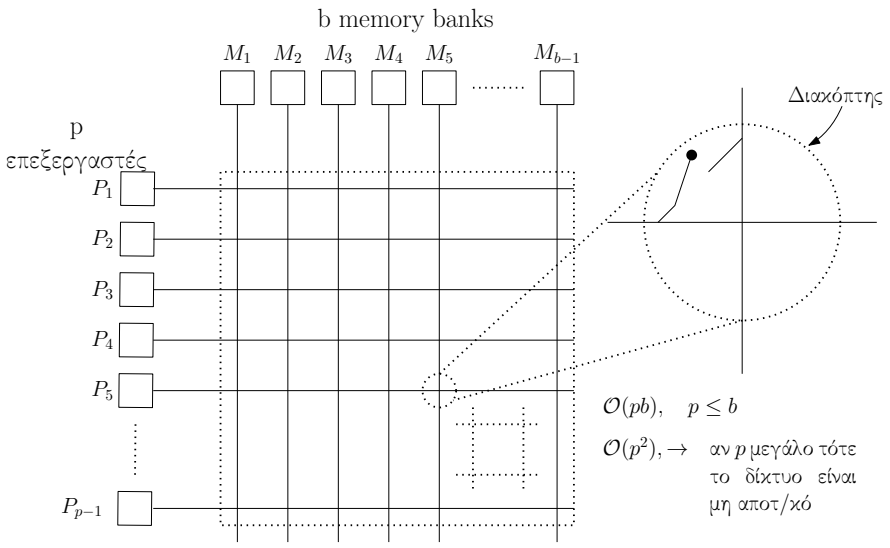
Για CREW SM SIMD:

- 1 βήμα για μετάδοση του x
- 1 βήμα για μετάδοση της F (τέλος κάθε αναζήτησης)

Άρα μόνο n/N βήματα για αναζήτηση.

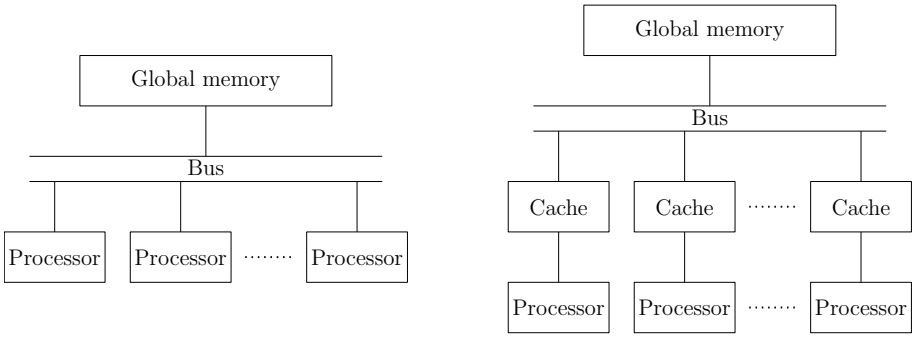
1.3 Δυναμικά Δίκτυα Διασύνδεσης

1.3.1 Crossbar Δίκτυα Διασύνδεσης

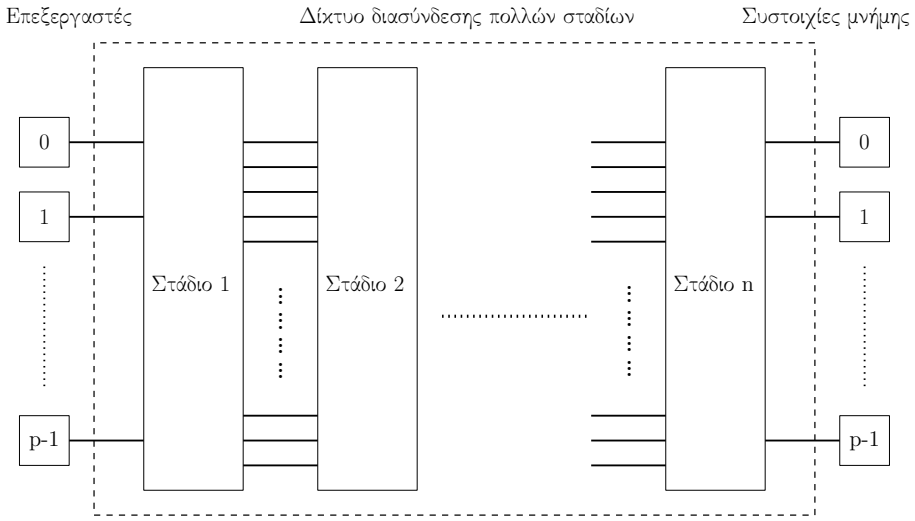


Σχήμα 1.9: Crossbar Switch: Cray Y-MP, Fujitsu VDD 500

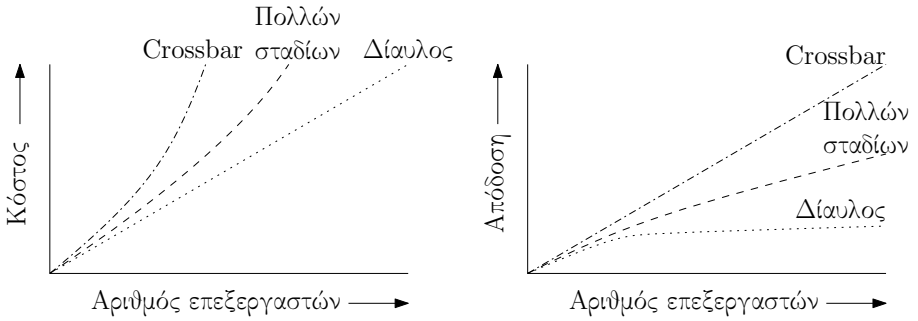
1.3.2 Bus-based Δίκτυα Διασύνδεσης



Σχήμα 1.10: Αρχιτεκτονική με bus: Symmetry, Multimax - κορεσμός με μικρό πλήθος επεξεργαστών



Σχήμα 1.11



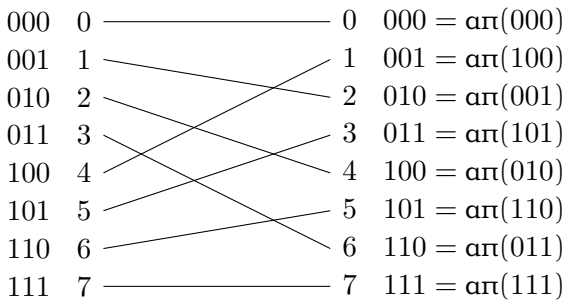
Σχήμα 1.12

1.3.3 Πολυφασικά Multistage Δίκτυα Διασύνδεσης

- Δίκτυο ωμέγα

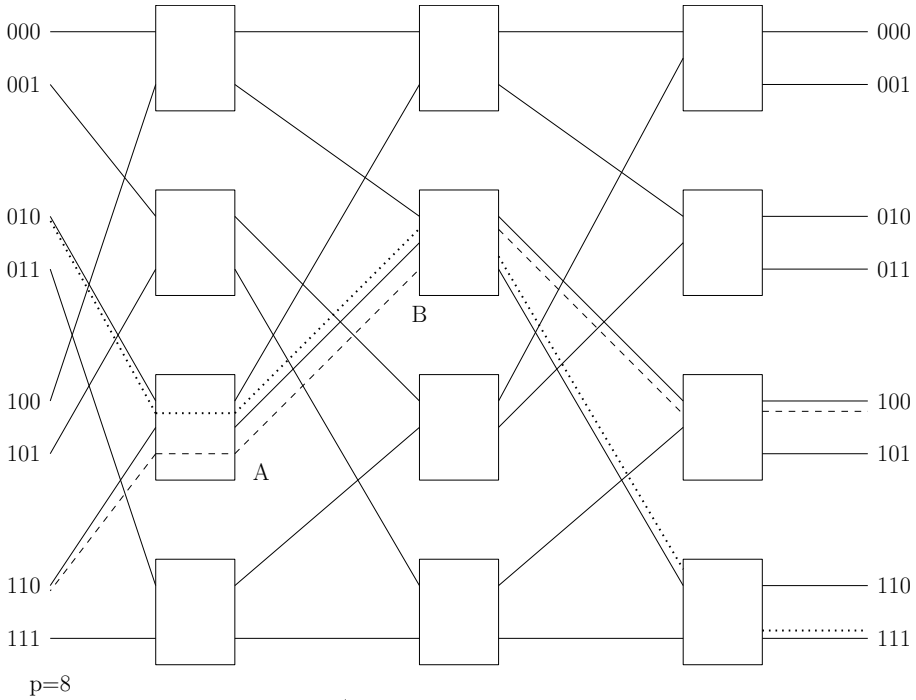
$$j = \begin{cases} 2i, & 0 \leqslant i \leqslant p/2 - 1 \\ 2i + 1 - p, & p/2 \leqslant i \leqslant p - 1 \end{cases}$$

Αριστερή περιστροφή στη δυαδική παράσταση i για να ληφθεί το j



Παράλληλοι υπολογιστές: BBN Butterfly, IBM RP-3, NYU Ultracomputer

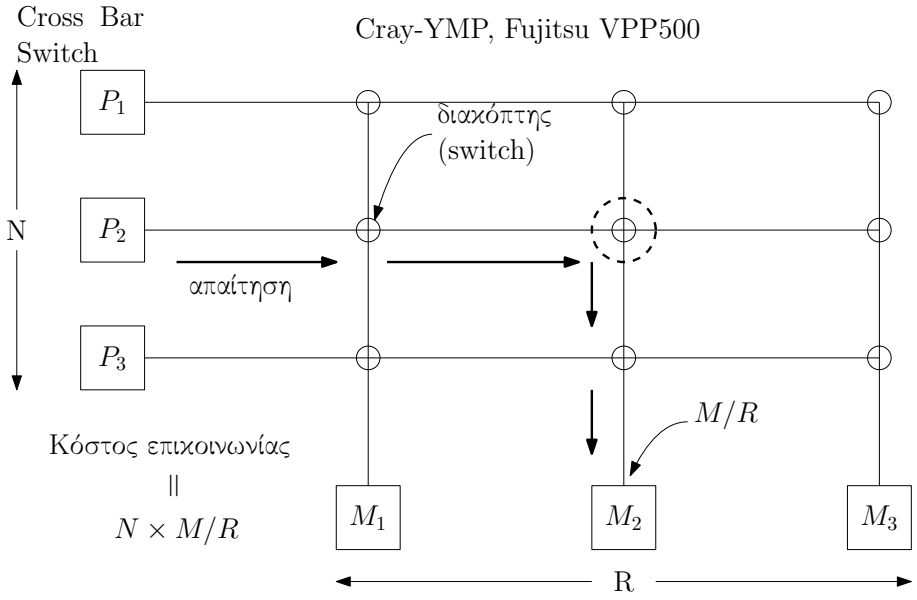
Παρατήρηση: Μόνο ένας επεξεργαστής μπορεί να διαβάσει ή να γράψει σε ένα τμήμα μνήμης.



Σχήμα 1.13: Δίκτυο Ωμέγα

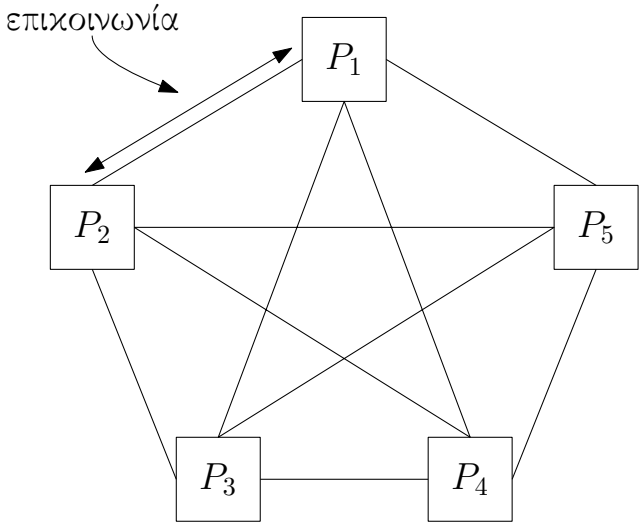
1.3.4 Δίκτυο Διασύνδεσης

- Οι μνήμες M_i είναι κατανεμημένες μεταξύ των N επεξεργαστών ($M_i = M/N$ θέσεις μνήμης)
- Διπλής κατεύθυνσης γραμμή επικοινωνίας μεταξύ των επεξεργαστών (μηνύματα)


Cross Bar
Switch
Cray-YMP, Fujitsu VPP500
P_1
P_2
P_3
διακόπτης
(switch)
N
απαίτηση
M/R
Κόστος επικοινωνίας
||
$N \times M/R$
M_1
M_2
M_3
R

Σχήμα 1.14

Κλίκα (clique)

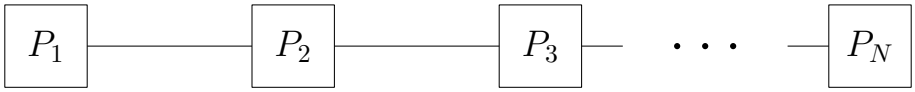


Σχήμα 1.15: Κλίκα (clique)

Σύνολο $\frac{N(N-1)}{2}$ γραμμές.

Εξωπραγματικό για μεγάλο N

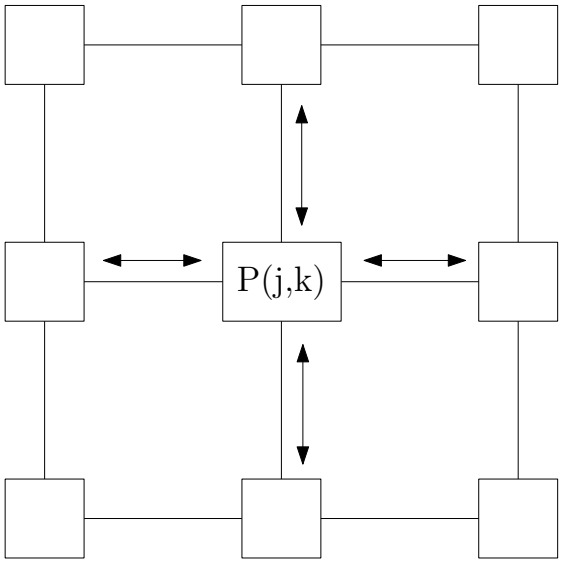
Γραμμικός πίνακας



Σχήμα 1.16: Γραμμικός πίνακας

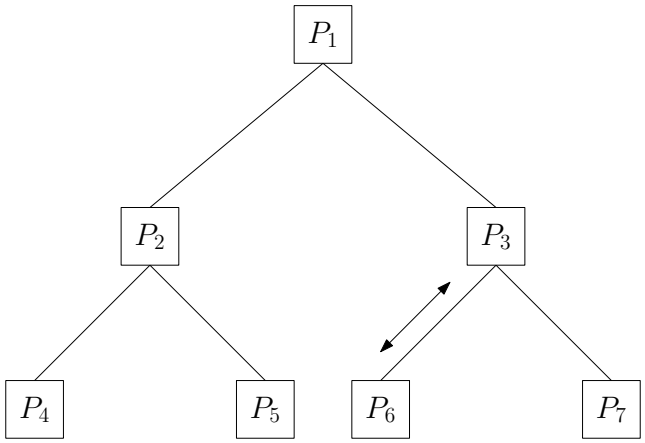
Διδιάστατος πίνακας

$\mathrm{m} \times \mathrm{m}, \quad \mathrm{m} = \sqrt{\mathrm{N}}$ (mesh)

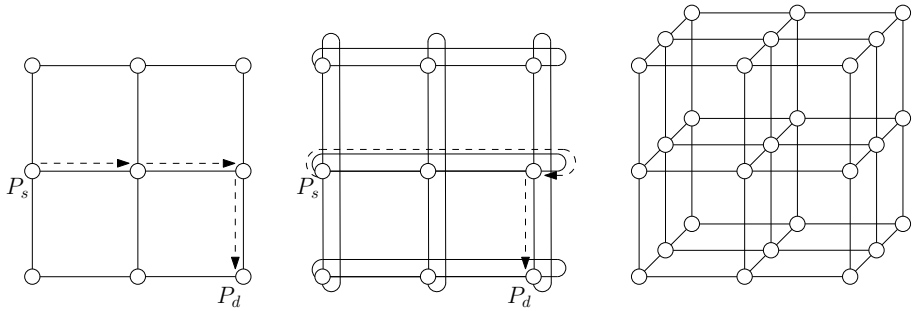


Σχήμα 1.17: Διδιάστατος πίνακας

Δέντρο

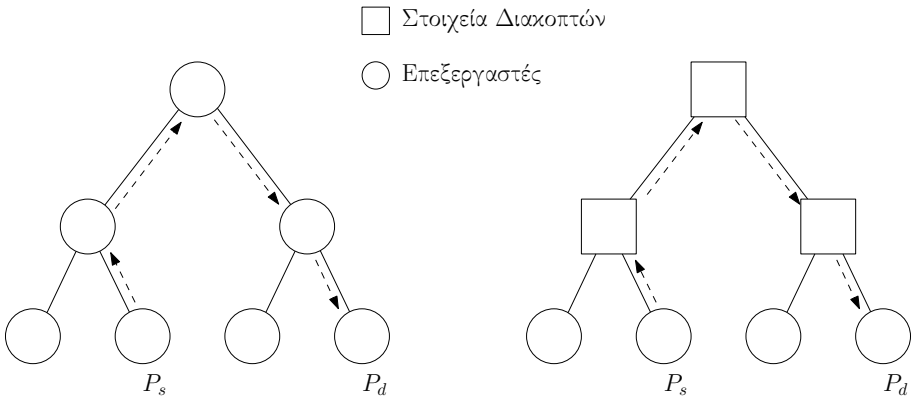


Σχήμα 1.18: Δέντρο



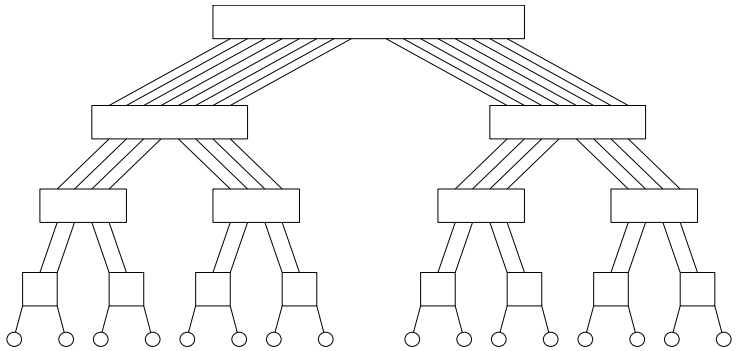
Σχήμα 1.19

Δίκτυο Δέντρου



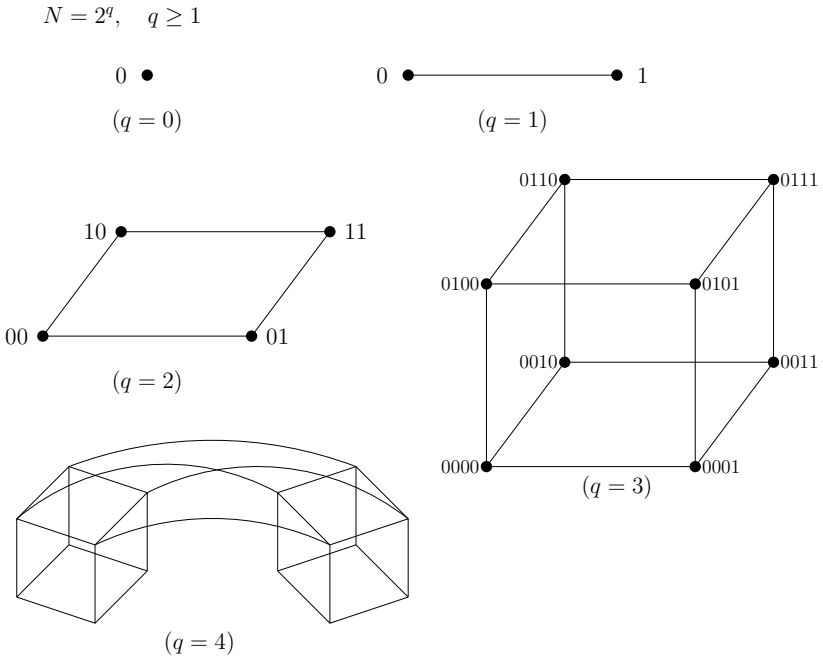
Σχήμα 1.20

- Δίκτυα Πλήρους Δυαδικού Δέντρου
- Διαδρομή Μηνύματος



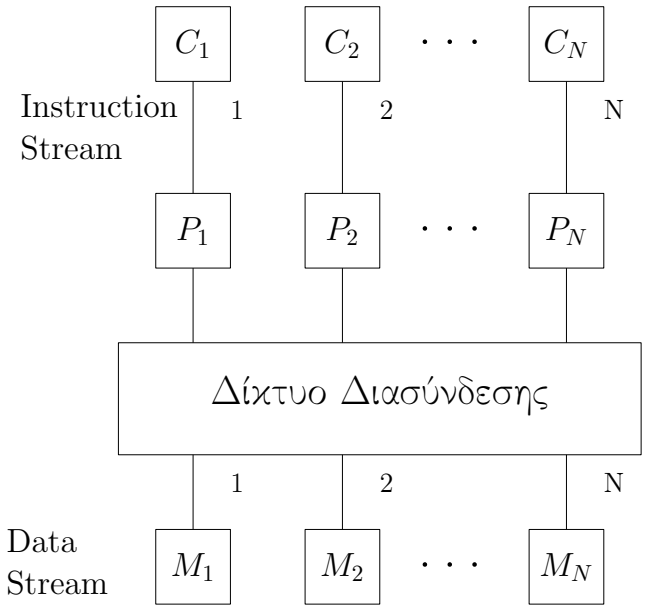
Σχήμα 1.21: fat tree network

Κύβος



Σχήμα 1.22

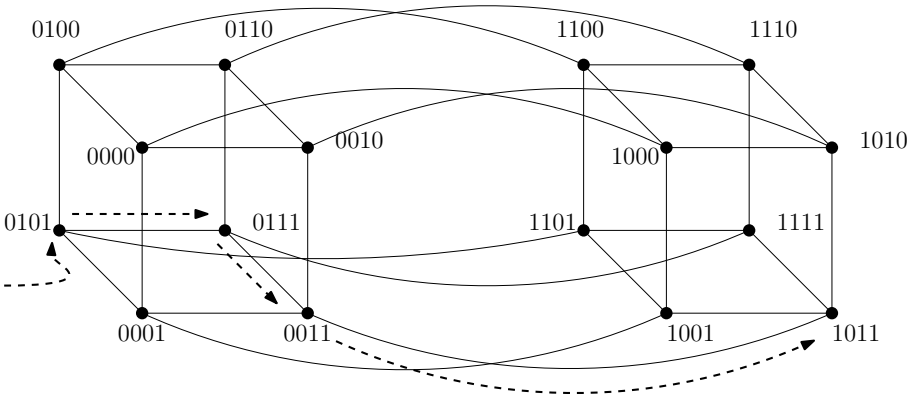
1.4 MIMD Υπολογιστές



Σχήμα 1.23: Multiple Instruction Multiple Data

- Σύγχρονη λειτουργία
- Ασύγχρονη λειτουργία

Δίκτυο Υπερκύβου



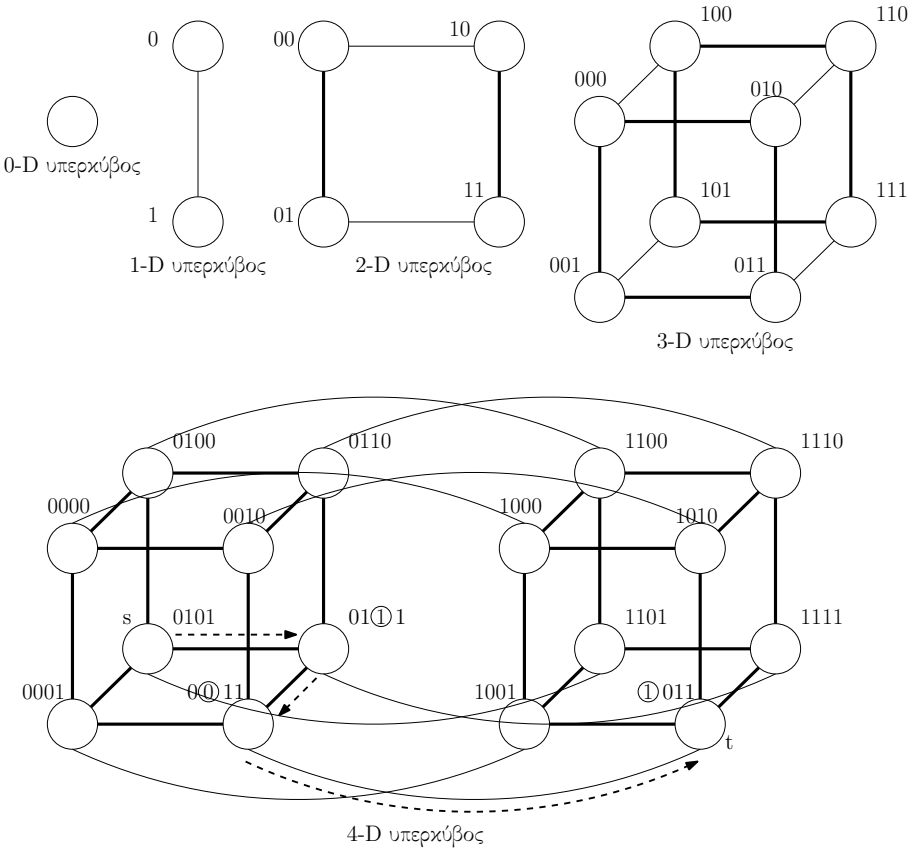
Σχήμα 1.24: Δίκτυο Υπερκύβου

d-διάστασης υπερκύβος
$p = 2^d$ επεξεργαστές

Ιδιότητες

- Δύο επεξεργαστές συνδέονται τότε και μόνο τότε αν διαφέρουν κατά ένα ακριβώς bit.
- Σε έναν d-διάστασης υπερκύβο ο κάθε επεξεργαστής συνδέεται άμεσα με d άλλους επεξεργαστές.
- Ο συνολικός αριθμός των bit θέσεων στα οποία διαφέρουν δύο επεξεργαστές λέγεται απόσταση του Hamming.

Αρίθμηση των επεξεργαστών



Σχήμα 1.25: Απόσταση Hamming 0101 και 1011 = 3 ($s \oplus t = 1110$)

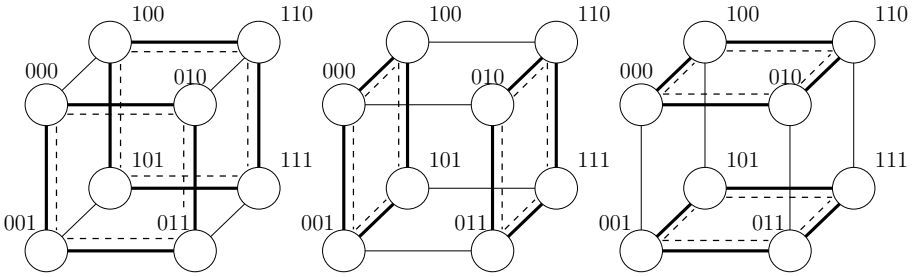
Όταν ένας υπερκύβος d+1-διάστασης κατασκευάζεται με τη σύνδεση δύο υπερκύβων d-διάστασης, στις ετικέτες του ενός υπερκύβου τίθεται το πρόθε-

μα 0 και στον άλλο υπερκύβο το πρόθεμα 1.

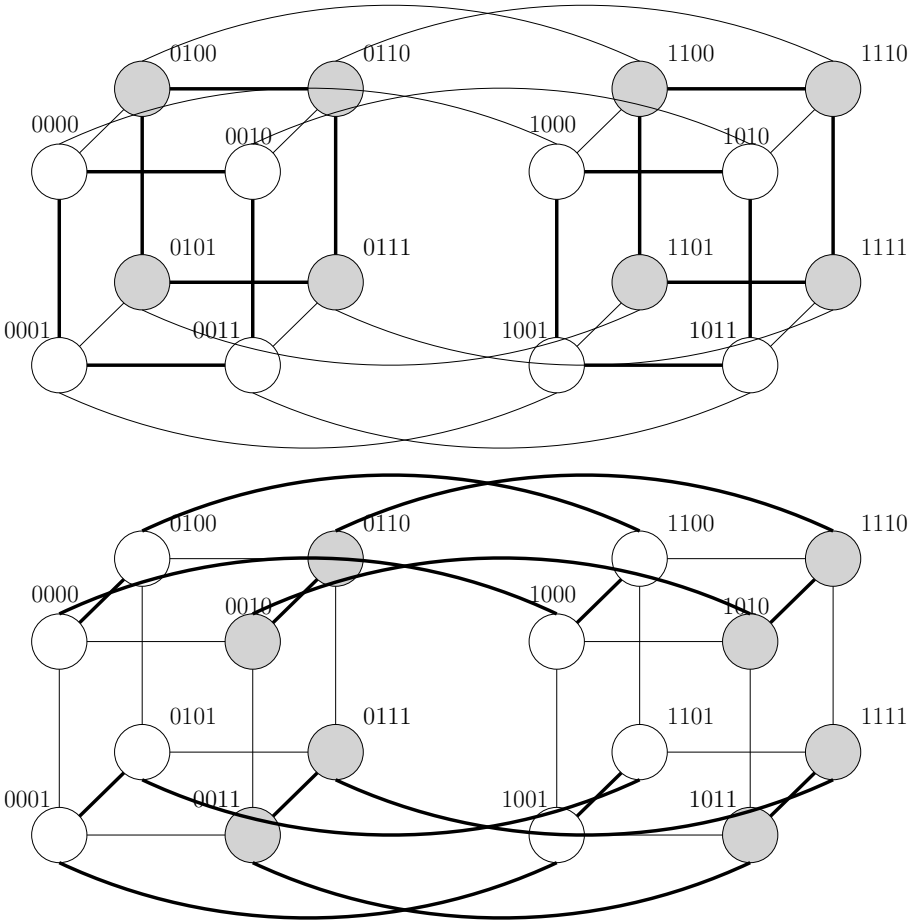
Ο μόνος επεξεργαστής (γειτονικός) του s που έχει bit 1 στη θέση ένα είναι αυτός που δείχνεται στο Σχήμα 1.25, κ.ο.κ.

- Η απόσταση του Hamming μεταξύ δύο επεξεργαστών s και t είναι ο συνολικός αριθμός των bits τα οποία είναι 1 στην $s \oplus t$ (όπου $\oplus$ συμβολίζει xor).
- Ο αριθμός των συνδέσμων επικοινωνίας στο συντομότερο μονοπάτι μεταξύ δύο επεξεργαστών είναι ίσος με την απόσταση του Hamming. Ένα μήνυμα δρομολογείται από τον επεξεργαστή s στον επεξεργαστή t διαμέσου διαστάσεων οι οποίες αντιστοιχούν σε θέσεις bit που έχουν 1 στη δυαδική παράσταση του $s \oplus t$. Επειδή η δυαδική παράσταση του $s \oplus t$ μπορεί να περιέχει το πολύ d μονάδες, το συντομότερο μονοπάτι μεταξύ δύο επεξεργαστών σε ένα υπερκύβο δε μπορεί να έχει περισσότερους από d συνδέσμους.

Παράλληλοι Υπολογιστές βασισμένοι σε δίκτυο υπερκύβου είναι οι: n CUBE, Cosmic Cube και iPSC.



Σχήμα 1.26

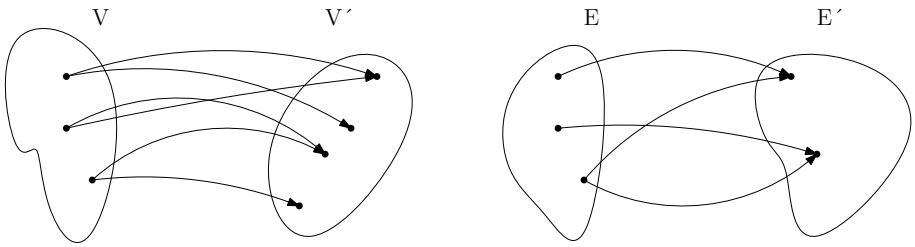

0100
0110
1100
1110
0000
0010
1000
1010
0101
0111
1101
1111
0001
0011
1001
1011
0100
0110
1100
1110
0000
0010
1000
1010
0101
0111
1101
1111
0001
0011
1001
1011

Σχήμα 1.27

Δίκτυο	**Διάμετρος**	**Εύρος Διχοτόμησης**	**Συνδεσιμότητα Τόξου**	**Κόστος (αριθ. συνδέσεων)**
Πλήρως συνδεδεμένο	1	$p^2/4$	$p-1$	$p(p-1)/2$
Αστέρας	2	1	1	$p-1$
Πλήρες Δυαδικό Δέντρο	$2\log((p+1)/2)$	1	1	$p-1$
Γραμμικός πίνακας	$p-1$	1	1	$p-1$
Δακτύλιος	$\lfloor p/2 \rfloor$	2	2	p
2-D πλέγμα χωρίς κύρτωση	$2(\sqrt{p}-1)$	$\sqrt{p}$	2	$2(p-\sqrt{p})$
2-D πλέγμα με κύρτωση	$2\lfloor \sqrt{p}/2 \rfloor$	$2\sqrt{p}$	4	$2p$
Υπερκύβος	$\log p$	$p/2$	$\log p$	$(p \log p)/2$
d-κύβος k-τάξης με κύρτωση	$d\lfloor k/2 \rfloor$	$2k^{d-1}$	$2d$	dp

1.4.1 Εμφύτευση Δικτύων στον Υπερκύβο

$$G(V,E) \xrightarrow{\text{εμφύτευση}} G'(V',E')$$



Σχήμα 1.28

Η εμφύτευση ενός γραφήματος εντός ενός άλλου είναι σημαντική διότι ένας αλγόριθμος που έχει σχεδιαστεί για ένα συγκεκριμένο δίκτυο διασύνδεσης μπορεί να είναι αναγκαίο να προσαρμοστεί σε ένα άλλο δίκτυο.

- συσσώρευση (congestion): Το μέγιστο πλήθος ακμών που απεικονίζονται σε μια ακμή στο E'.
- διαστολή (dilation): Το πλήθος συνδέσμων στο E' στους οποίους απεικονίζεται μία ακμή του E.
- επέκταση (expansion): Επεξεργαστές στο V' / Επεξεργαστές στο V.

Στη συνέχεια θα μελετηθούν οι εμφυτεύσεις διαφόρων δικτύων διασύνδεσης εντός του υπερκύβου. Η μελέτη θα περιοριστεί στην περίπτωση όπου

το πλήθος των επεξεργαστών είναι το ίδιο στα δύο δίκτυα (επέκταση = 1). Επιπλέον το πολύ μία ακμή του E απεικονίζεται σε μία ακμή του E' (συσσώρευση = 1).

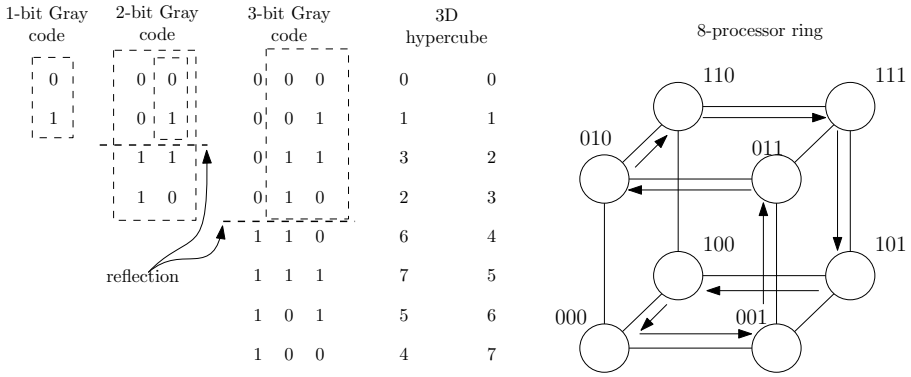
Εμφύτευση γραμμικού πίνακα σε υπερκύβο

Ένας γραμμικός πίνακας (ή ένας δακτύλιος) που αποτελείται από 2^d επεξεργαστές μπορεί να εμφυτευθεί σε ένα υπερκύβο d διάστασης απεικονίζοντας τον $i-$οστό επεξεργαστή του γραμμικού πίνακα στον επεξεργαστή $G(i,d)$ του υπερκύβου, όπου

$$G(0,1) = 0$$
$$G(1,1) = 1$$

$$G(i,x+1) = \begin{cases} G(i,x), & i < 2^x \\ 2^x + G(2^{x+1}-1-i,x), & i \geqslant 2^x \end{cases}$$

Η συνάρτηση G λέγεται binary reflected Gray code (RGC). Η $G(i,d)$ συμβολίζει το $i-$οστό στοιχείο στην ακολουθία των κωδίκων Gray με d bits. Οι κώδικες Gray d+1 bits παράγονται από ένα πίνακα των κωδίκων Gray με d bits ως εξής: Παίρνοντας το συμμετρικό πίνακα και τοποθετώντας στα στοιχεία του συμμετρικού πίνακα το πρόθεμα 1 και στα αρχικά στοιχεία το πρόθεμα 0.



Σχήμα 1.29: Εμφύτευση ενός δακτυλίου οκτώ επεξεργαστών σε ένα υπερκύβο διάστασης 3

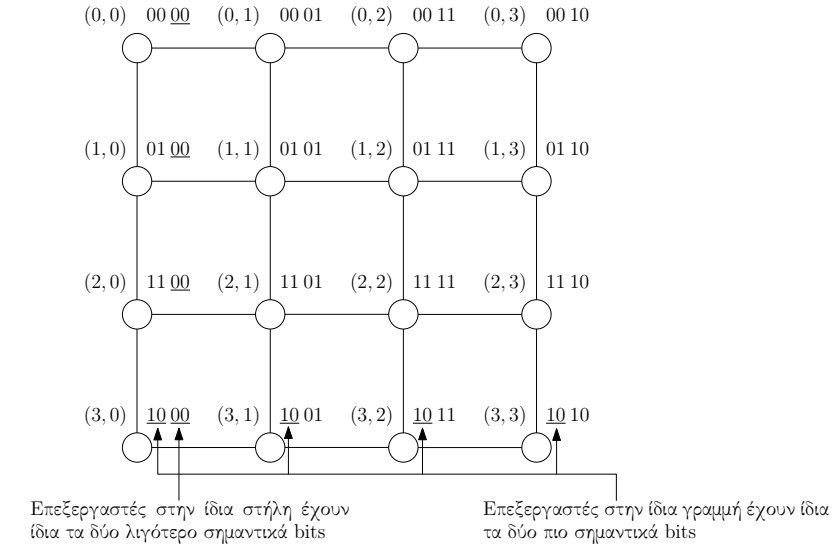
Τα στοιχεία $G(i,d)$ και $G(i+1,d)$ διαφέρουν μόνο σε ένα bit. Επειδή ο επεξεργαστής i στο γραμμικό πίνακα απεικονίζεται στον επεξεργαστή $G(i,d)$ του υπερκύβου και ο επεξεργαστής $i+1$ του γραμμικού πίνακα στον επεξεργαστή $G(i+1,d)$ του υπερκύβου, υπάρχει ένας άμεσος

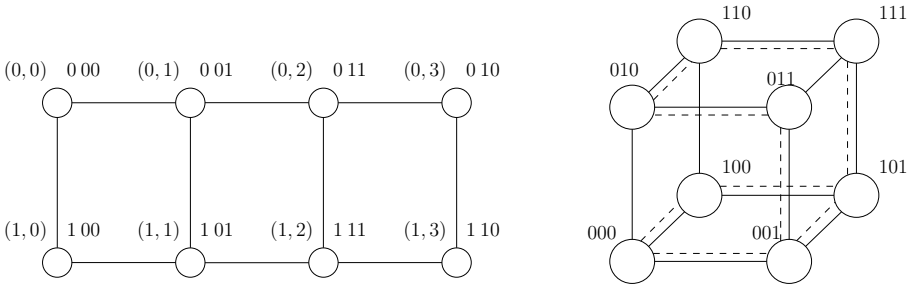
σύνδεσμος στον υπερκύβο, ο οποίος αντιστοιχεί σε κάθε άμεσο σύνδεσμο στο γραμμικό πίνακα (οι ετικέτες δύο γειτονικών επεξεργαστών διαφέρουν μόνο κατά ένα bit). Συνεπώς, η απεικόνιση που ορίζεται από την G έχει συσσώρευση ένα και διαστολή ένα.

Εμφύτευση δυδιάστατου δικτύου (mesh) σε υπερκύβο

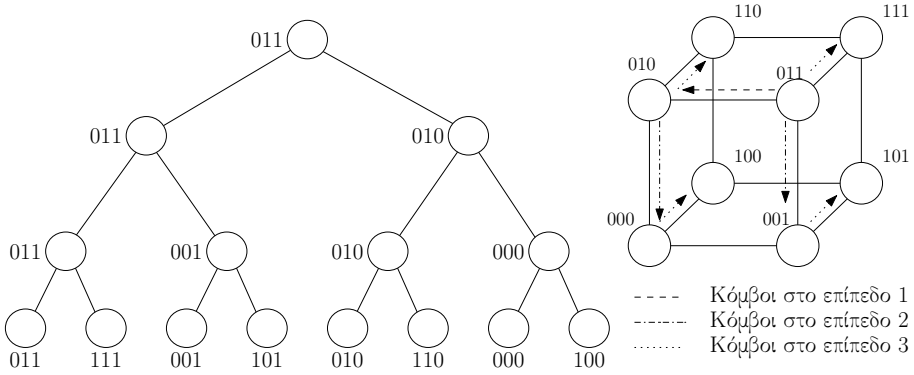
Μπορούμε να εμφυτεύσουμε ένα $2^r \times 2^s$ wraparound mesh σε έναν 2^{r+s} - επεξεργαστών υπερκύβο απεικονίζοντας τον επεξεργαστή (i, j) του mesh στον επεξεργαστή $G(i, r)||G(j, s)$ του υπερκύβου, όπου $||$ συμβολίζει τη συνένωση των δύο κωδίκων Gray. Ας σημειωθεί ότι οι άμεσοι γείτονες στο mesh απεικονίζονται σε επεξεργαστές του υπερκύβου των οποίων οι ετικέτες διαφέρουν μόνο σε ένα bit). Συνεπώς η απεικόνιση αυτή έχει μία διάδοση = 1 και μία συσσώρευση = 1.

$r = 1, s = 2, \qquad (i, j) \longrightarrow G(i, 1)||G(j, 2)$
$(0, 0) \longrightarrow G(0, 1)||G(0, 2) = 0||00 = 000$
$(0, 1) \longrightarrow G(0, 1)||G(1, 2) = 0||01 = 001$



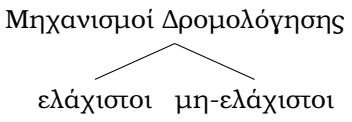


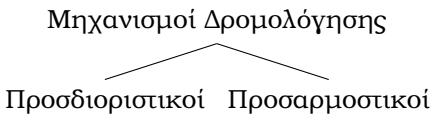
Σχήμα 1.30: Εμφύτευση 2×4 mesh

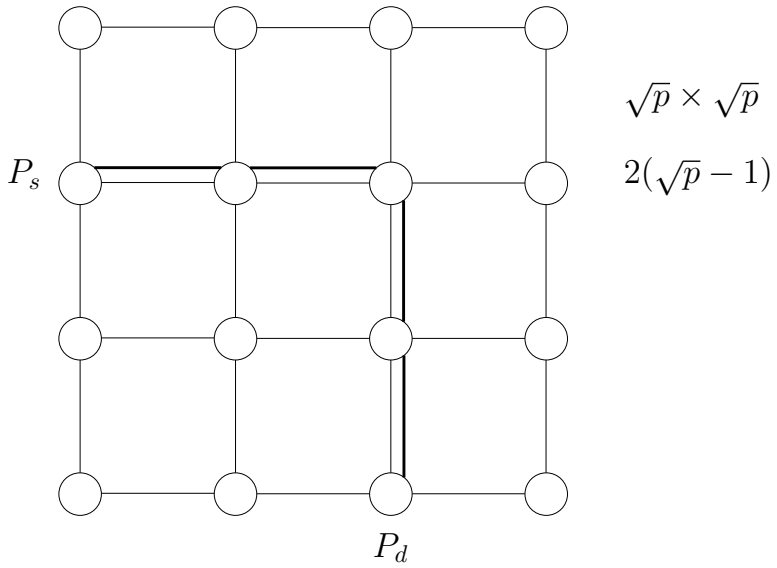


Σχήμα 1.31

Δρομολόγηση Μηνυμάτων







Σχήμα 1.32: XY - δρομολόγηση σε mesh

Κόστος Επικοινωνίας

- *καθυστέρηση επικοινωνίας (latency)*

Ο απαιτούμενος χρόνος για την επικοινωνία ενός μηνύματος μεταξύ δύο επεξεργαστών.

- *χρόνος ξεκινήματος (startup time)* (t_s)

Ο απαιτούμενος χρόνος για τη μεταχείριση του μηνύματος (προετοιμασία).

- *per-hop χρόνος* (t_h)

Ο απαιτούμενος χρόνος για να φθάσει η επικεφαλίδα του μηνύματος στον προορισμό της.

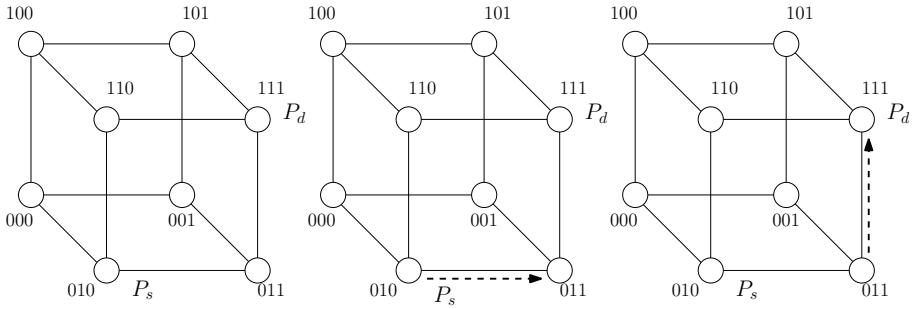
- *χρόνος ανά word* (t_w)

$$t_w = 1/r$$

όπου r = αριθμός words ανά δευτερόλεπτο

Καθυστέρηση Επικοινωνίας

τοπολογία δικτύου τεχνικές μεταφοράς μηνυμάτων



010 ⊕ 110 = 1⓪①
010 ⊕ 111 = 101
010 ⊕ 011 = 001
010 ⊕ 000 = 010
010 ⊕ 110 = 100

Βήμα 1 (010 → 01①)
010 ⊕ 111 = 0①0
011 ⊕ 111 = 100

Βήμα 2 (011 → 111)
011 ⊕ 111 =

Σχήμα 1.33: E-cube δρομολόγηση

1.4.2 Υπολογισμός κόστους επικοινωνίας

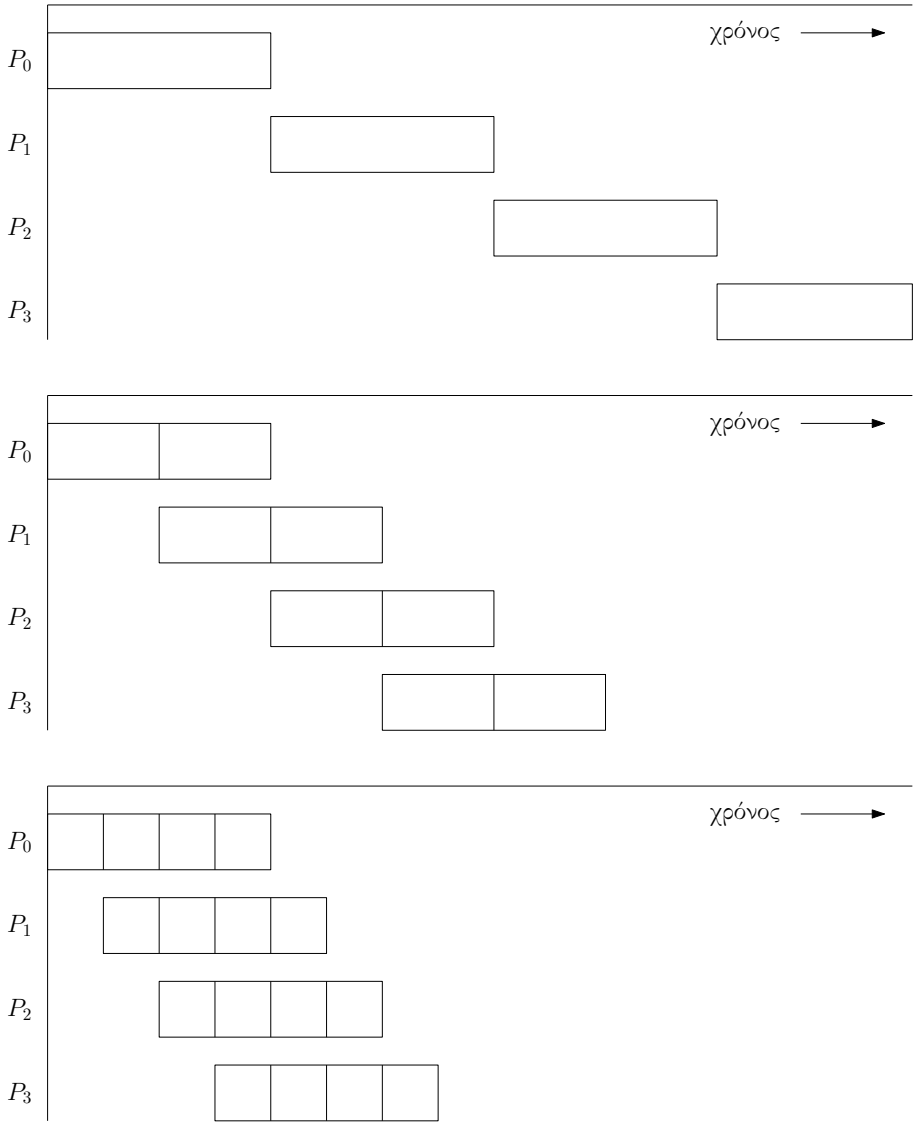
$$t_{comm} = t_s + (m\ t_w + t_h)\ l$$

όπου m = πλήθος words, l = πλήθος συνδέσμων

$$t_{comm} = t_s + m\ t_w\ l, \quad \mathcal{O}(ml) \quad \text{store and forward}$$

$$t_{comm} = t_s + l\ t_h + m\ t_w, \quad \mathcal{O}(l+m) \quad \text{cut through}$$

$l = 1$ τοπικότητα



Σχήμα 1.34

1.5 Βασικές Λειτουργίες Επικοινωνίας

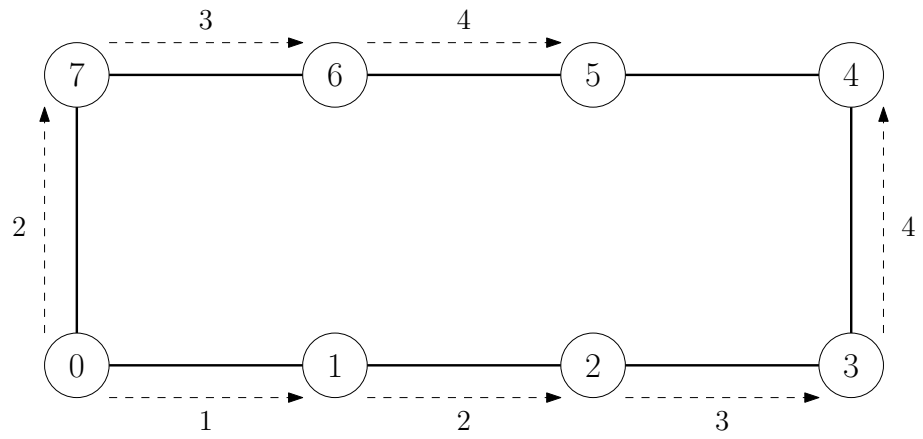
- Οι σύνδεσμοι επικοινωνίας είναι διπλής κατεύθυνσης.
- Ο κάθε επεξεργαστής μπορεί να στείλει ένα μήνυμα σε ένα μόνο από τους συνδέσμους του κάθε φορά.
- Όμοια μπορεί να λάβει ένα μήνυμα από ένα σύνδεσμο κάθε φορά.
- Μπορεί να λάβει ένα μήνυμα ενώ στέλνει ένα άλλο την ίδια στιγμή στον ίδιο ή διαφορετικό σύνδεσμο.

SF δρομολόγηση

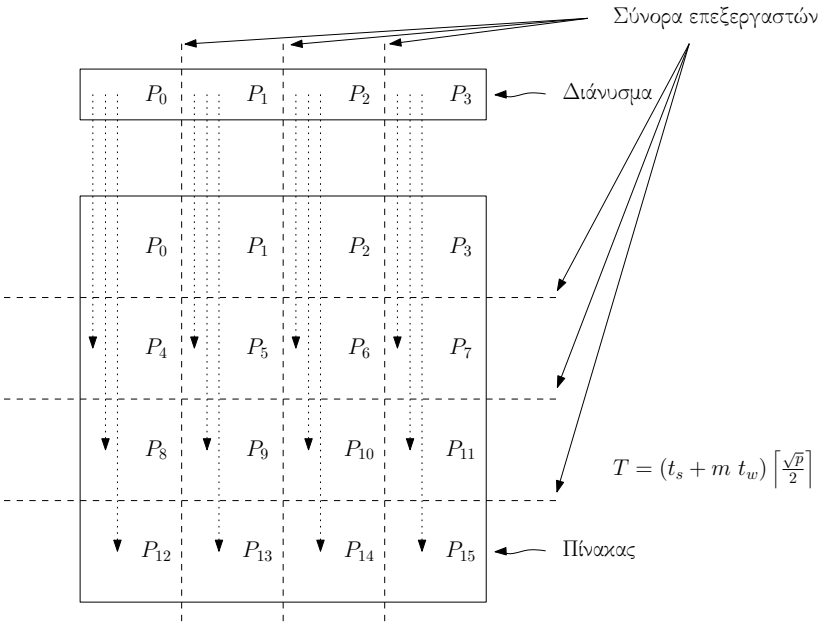
- $t_s + t_w m\lfloor p/2 \rfloor$ δακτύλιο
- $t_s + 2t_w m\lfloor p/2 \rfloor$ δίκτυο
- $t_s + t_w m \log p$ υπερκύβο

CT δρομολόγηση

$t_s + m\ t_w + t_h\ l$
Ένας προς όλους Μετάδοση
Δακτύλιος SF



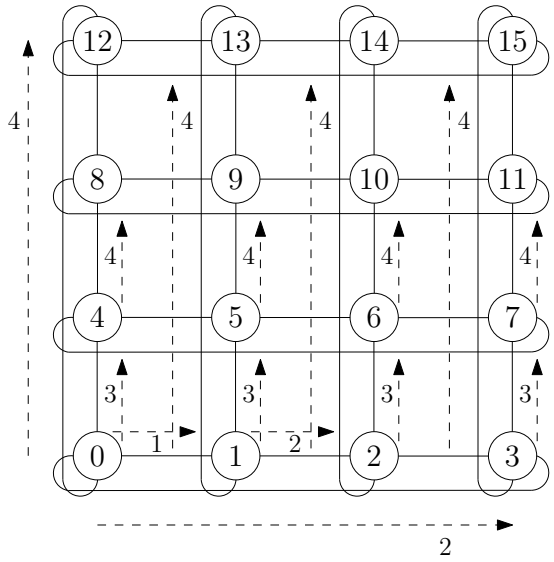
Σχήμα 1.35: Δακτύλιος, $T = t_s + t_w m\lceil p/2 \rceil$



Σχήμα 1.36: Ένας προς όλους μετάδοση για τον πολλαπλασιασμό ενός 4×4 πίνακα με ένα 4×1 διάνυσμα

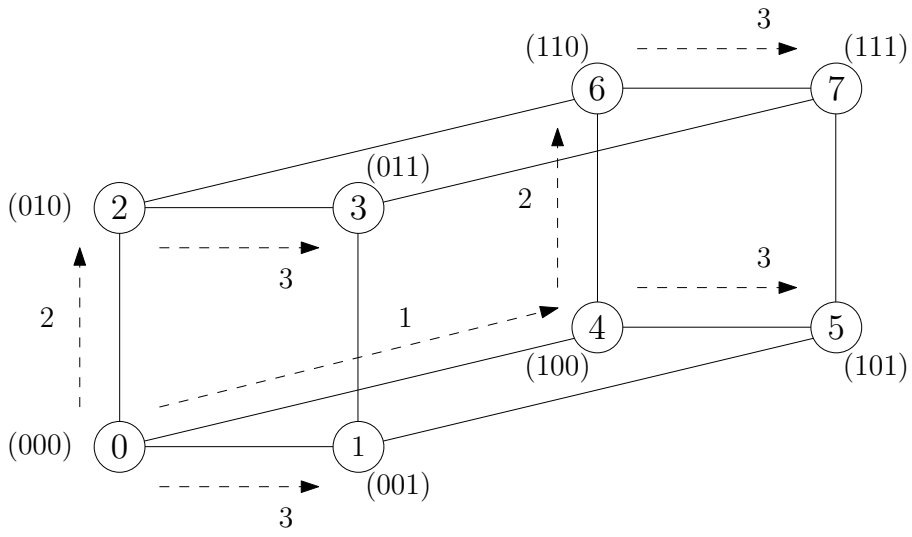
$$T = 2(t_s + t_w\ m)\left\lceil \frac{\sqrt{p}}{2} \right\rceil$$

$$T = 3(t_s + t_w\ m)\left\lceil \frac{p^{1/3}}{2} \right\rceil \quad 30 \text{ mesh}$$



Σχήμα 1.37: Ένας προς όλους μετάδοση σε πλέγμα 16 επεξεργαστών με store and forward δρομολόγηση

$$T = (t_s + t_w\ m) \log p$$



Σχήμα 1.38: Ένας προς όλους μετάδοση σε 3-D υπερκύβο

Διαδικασία 1: ONE TO ALL BC $(\mathrm{d}, \mathrm{my\_id}, \mathrm{X})$

$\overbrace{\mathrm{mask}}^{111} = 2^{\mathrm{d}} - 1 \quad (\mathrm{d} = 3)$ ▷ όλα τα `d bits` της μάσκας ίσα με 1
για $\mathrm{i} = \mathrm{d} - 1$ **έως** 0 ▷ Εξωτερικό `loop` $(\mathrm{i} = 2)$
κάνε
  $\mathrm{mask} = \mathrm{mask}$ XOR 2^{i} ▷ `i bit` της μάσκας ίσο με 0
  αν my_id *AND* mask $= 0$ ▷ αν τα μικρότερα `i bits` του my_id είναι 0
  τότε
    αν my_id *AND* $2^{\mathrm{i}} = 0$ ▷ $000 \xrightarrow{\texttt{send}} 100$
    τότε
      msg_destination $=$ my_id XOR 2^{i}
      Στείλε Q στο msg_destination
    τέλος
  αλλιώς
    msg_source $=$ my_id XOR 2^{i} ▷ $000 \xrightarrow{\texttt{receive}} 100$
    Λάβε X από msg_source
  τέλος
τέλος

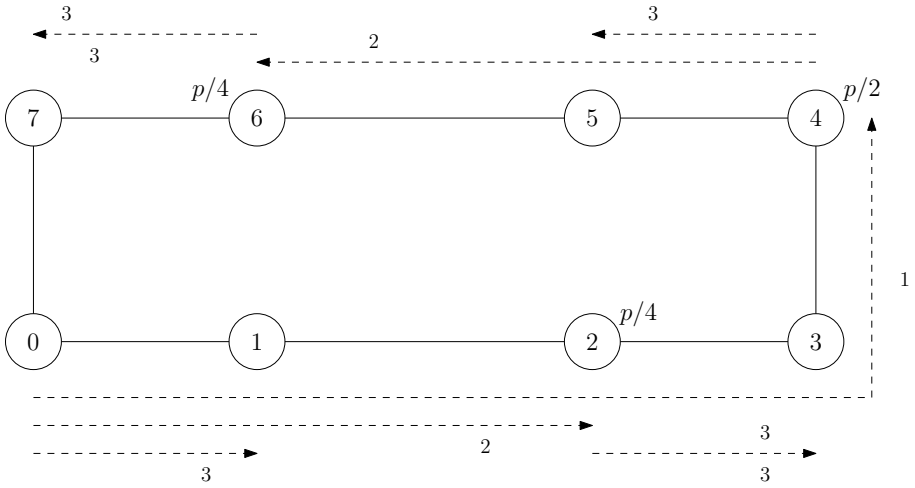
Διαδικασία 2: GENERAL ONE TO ALL BC $(d, my\_id, source, X)$

```
my_virtual_id = my_id XOR source  mask = 2^d − 1
για i = d − 1 έως 0 κάνε
    mask = mask XOR 2^i                    ▷ i bit της μάσκας ίσο με 0
    αν my_virtual_id AND mask = 0 τότε
        αν my_virtual_id AND 2^i = 0 τότε
            virtual_dest = my_virtual_id XOR 2^i
            Στείλε Q στο virtual_dest XOR source  ▷ μετατροπή
             virtual_dest στην ετικέτα του φυσικού προορισμού
        τέλος
    αλλιώς
        virtual_source = my_virtual_id XOR 2^i
        Λάβε X στο virtual_source XOR source      ▷ μετατροπή
         virtual_source στην ετικέτα της φυσικής πηγής
    τέλος
τέλος
```

Διαδικασία 3: SINGLE NODE ACC (d, my_id, m, X, sum)

```
για j = 0 έως m − 1 κάνε
    sum[j] = X[j]  mask = 0
τέλος
για i = 0 έως d − 1 κάνε
    Επιλογή επεξεργαστών με μικρότερα i bits ίσα με 0
    αν my_id AND mask = 0 τότε
        αν my_id AND 2^i ≠ 0 τότε
            msg_destination = my_id XOR 2^i
            Στείλε sum στο msg_destination
        τέλος
    αλλιώς
        msg_source = my_id XOR 2^i
        Λάβε X από msg_source για j = 0 έως m − 1 κάνε
            sum[j] = sum[j] + X[j]
        τέλος
    τέλος
    mask = mask XOR 2^i                    ▷ i bit της μάσκας ίσο με 0
τέλος
```

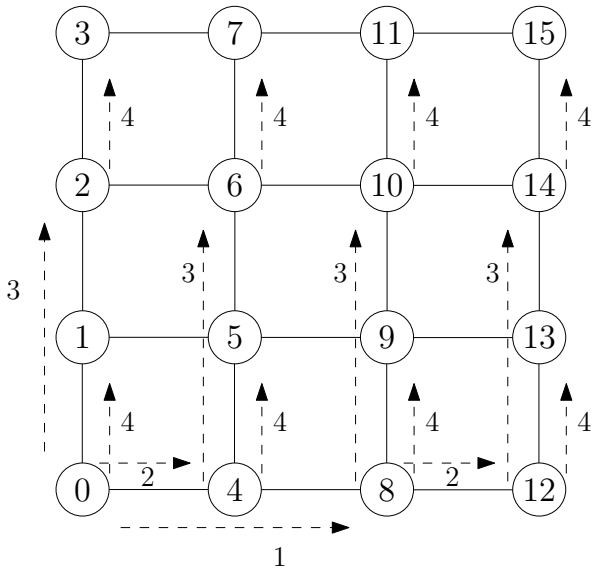
- Ένας σε όλους μετάδοση με cut through δρομολόγηση
- Στο i-οστό βήμα κάθε επεξεργαστής που έχει τα δεδομένα τα στέλνει σε έναν επεξεργαστή απόστασης $p/2^i$



Σχήμα 1.39: Ένας σε όλους μετάδοση με CT δρομολόγηση

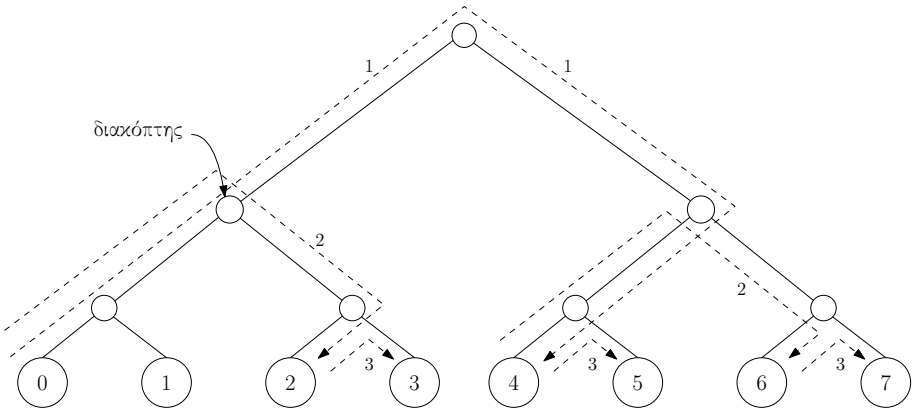
$$T_{\text{ένας-προς-όλους}} = \sum_{i=1}^{\log p} (t_s + t_w\, m + t_h\, p/2^i) = (t_s + t_w\, m)\log p + t_h(p-1)$$

- Μετάδοση ένας προς όλους με CT δρομολόγηση
- $t_s + m\ t_w \log \sqrt{p} + (\sqrt{p} - 1)t_h$ κάθε φάση



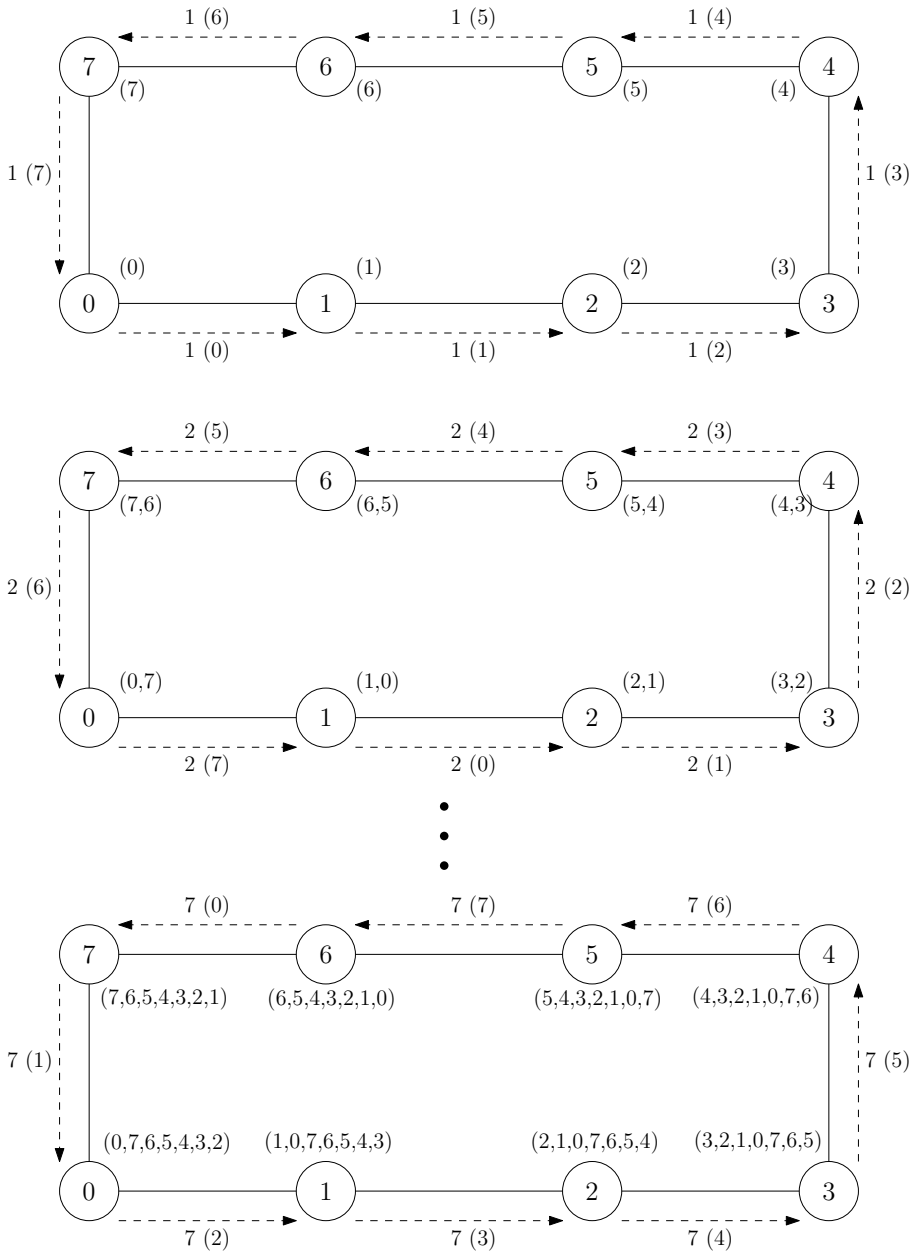
Σχήμα 1.40

$$T_{\text{ένας-προς-όλους}} = (t_s + m\ t_w) \log p + 2(\sqrt{p} - 1)t_h$$

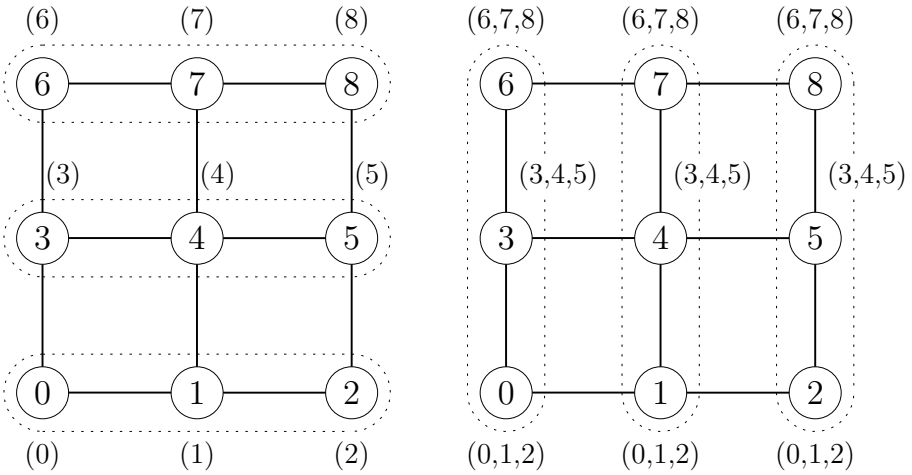


Σχήμα 1.41: Ισοζυγισμένο Δυαδικό Δέντρο

$$T_{\text{ένας-προς-όλους}} = (t_s + m\ t_w + (\log p + 1)) \log p$$


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

Σχήμα 1.42



Σχήμα 1.43

Διαδικασία 4: ALL TO ALL BC RING $(\mathtt{my\_id}, \mathtt{my\_msg}, \mathtt{p}, \mathtt{result})$

$\mathtt{left} = (\mathtt{my\_id} - 1) \mod \mathtt{p}$
$\mathtt{right} = (\mathtt{my\_id} + 1) \mod \mathtt{p}$
$\mathtt{result} = \mathtt{my\_msg}$
$\mathtt{msg} = \mathtt{result}$
για $\mathtt{i} = 1$ **έως** $\mathtt{p} - 1$ **κάνε**
  Στείλε msg από left
  $\mathtt{result} = \mathtt{result} \cup \mathtt{msg}$
τέλος

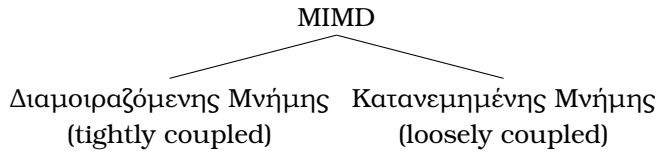
Διαδικασία 5: ALL TO ALL BC MESH $(\mathrm{my\_id}, \mathrm{my\_msg}, \mathrm{p}, \mathrm{result})$

▷ Επικοινωνία κατά μήκος γραμμής
$\mathrm{left} = (\mathrm{my\_id} - 1) \mod \mathrm{p}$
$\mathrm{right} = (\mathrm{my\_id} + 1) \mod \mathrm{p}$
$\mathrm{result} = \mathrm{my\_msg}$
$\mathrm{msg} = \mathrm{result}$
για $\mathrm{i} = 1$ **έως** $\sqrt{\mathrm{p}} - 1$ **κάνε**
  Στείλε msg στο right
  Λάβε msg από left
  $\mathrm{result} = \mathrm{result} \cup \mathrm{msg}$
τέλος

▷ Επικοινωνία κατά μήκος στήλης
$\mathrm{up} = (\mathrm{my\_id} - \sqrt{\mathrm{p}}) \mod \mathrm{p}$
$\mathrm{down} = (\mathrm{my\_id} + \sqrt{\mathrm{p}}) \mod \mathrm{p}$
$\mathrm{msg} = \mathrm{result}$
για $\mathrm{i} = 1$ **έως** $\sqrt{\mathrm{p}} - 1$ **κάνε**
  Στείλε msg στο down
  Λάβε msg από up
  $\mathrm{result} = \mathrm{result} \cup \mathrm{msg}$
τέλος

1.6 Αξιολόγηση MIMD Παράλληλων Συστημάτων

- Scheduling



Ανάλυση Παραλληλων Αλγορίθμων

$$T_p = T_{comp} + T_{comm}$$

Ταχύτητα και Αποδοτικότητα

$$S_p = \frac{T_1}{T_p}$$

S_p : ταχύτητα
T_1 : σειριακός χρόνος
T_p : παράλληλος χρόνος σε p επεξεργαστές

$$E_p = \frac{S_p}{p} \qquad \text{αποδοτικότητα}$$

Είναι φανερό ότι $S_p \geqslant 1$. Επίσης ένα βήμα ενός παράλληλου αλγορίθμου χρησιμοποιώντας p επεξεργαστές χρειάζεται p βήματα το πολύ όταν υλοποιηθεί σειριακά. Άρα

$$T_1 \leqslant p\, T_p$$

Συνεπώς

$$1 \leqslant S_p \leqslant p \quad \text{και} \quad 0 \leqslant E_p \leqslant 1$$

Ο νόμος του Amdahl

$$T_1 = T_{seq} + T_{par}, \quad T_{seq} = f\, T_1 \quad \text{και} \quad T_{par} = (1-f)\, T_1$$

f : Ποσοστό σειριακού χρόνου

Με p επεξεργαστές

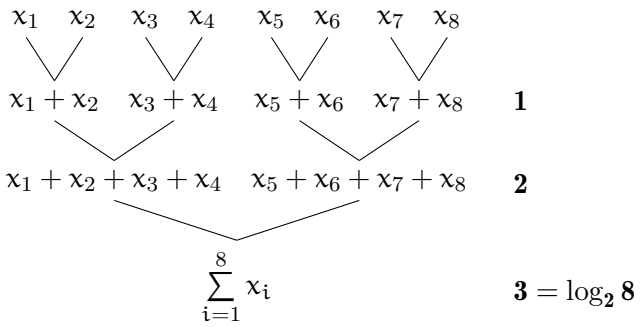
$$T_p \geqslant T_{seq} + T_{par}/p$$

$$S_p = \frac{T_1}{T_f} \leqslant \frac{1}{f + \frac{1-f}{p}} \leqslant \frac{1}{f}$$

Αν $f = 10\% \rightarrow S_p \leqslant 10$ για οποιοδήποτε p!

Παράδειγμα (SIMD)

Υπολογισμός του $\sum_{i=1}^{N} x_i$



Αριθμός Επεξεργαστών $p = \frac{N}{2}$ CREW

$\log N$ βήματα

$$S_p = \frac{T_1}{T_p} = \frac{N-1}{\log N} \approxeq \frac{N}{\log N}$$

$$E_p \approxeq \frac{2}{\log N} \rightarrow 0 \quad \text{για} \quad N \rightarrow \infty$$

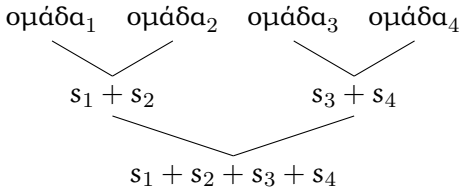
Ανάλογος αλγόριθμος για προβλήματα

- εσωτερικο γινόμενο διανυσμάτων
- εύρεση μέγιστου - ελάχιστου αριθμού κ.α.

Παράδειγμα

Υπολογισμός του $\sum_{i=1}^{N} x_i$ με τη χρήση p επεξεργαστών $1 \leqslant p \leqslant N$.

1. Χωρισμός των N αριθμών σε p ομάδες. Κάθε ομάδα περιέχει το πολύ $\left\lceil \frac{N}{p} \right\rceil$ αριθμούς
2. Καταχώρηση μιας ομάδας σε κάθε ένα από τους p επεξεργαστές
3. Υπολογισμός του αθροίσματος των αριθμών σε κάθε ομάδα σειριακά σε $\left\lceil \frac{N}{p} \right\rceil - 1$ βήματα
4. Υπολογισμός του αθροίσματος των επιμέρους αθροισμάτων (σε μορφή δυαδικού δέντρου)



$$T_p = \left\lceil \frac{N}{p} \right\rceil - 1$$

$$S_p = \frac{\overbrace{N-1}^{T_1}}{\lceil N/p \rceil + \lceil \log p \rceil - 1} \qquad \text{Αν} \qquad N = L\ p\ \log p$$

$$S_p = \frac{L\ p}{L+1} \to \lim_{L\to\infty} S_p = p \qquad \text{γραμμική στο } p.$$

Επίσης

$$E_p = \frac{1}{L+1} \to \lim_{L\to\infty} E_p = 1$$

Παράδειγμα

Δίνονται τα $X = (x_1, x_2, \ldots, x_N), \quad Y = (y_1, y_2, \ldots, y_N)$. Να υπολογιστεί το $\sum_{i=1}^{N} x_i y_i$.

Έστω ότι το p διαιρεί το N. Τότε

$$\sum_{i=1}^{N} x_i y_i = \sum_{i=1}^{N/p} x_i y_i + \sum_{i=N/p+1}^{2(N/P)} x_i y_i + \cdots + \sum_{i=(N/p)(k-1)+1}^{k(N/P)} x_i y_i + \cdots + \sum^{N} x_i y_i$$

Άρα ο k επεξεργαστής υπολογίζει

$$\sum_{i=(N/p)(k-1)+1}^{k(N/P)} x_i y_i, \quad k = 1, 2, \ldots, p$$

Για τον ανωτέρω υπολογισμό απαιτούνται

$$\frac{N}{p} \qquad \text{πολ/σμοί}$$

$$\frac{N}{p} - 1 \qquad \text{προσθέσεις}$$

Τα μερικά αθροίσματα προστίθενται με τη μορφή του δυαδικού δέντρου σε $\lceil \log p \rceil$ βήματα. Άρα

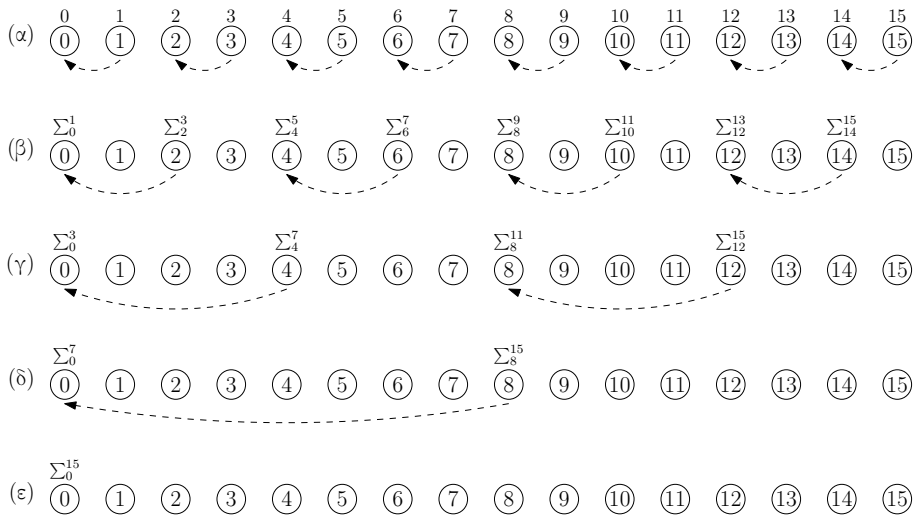
$$T_p = 2(\frac{N}{p} - 1) + \lceil \log p \rceil$$

και

$$S_p = \frac{2N - 1}{2(\frac{N}{p} - 1) + \lceil \log p \rceil}$$

Αν $p = 2^k$ και $N = L\ p\ \log p$ τότε

$$S_p = \frac{2L}{2L + 1} p \rightarrow \lim_{L \to \infty} S_p = p \quad !$$

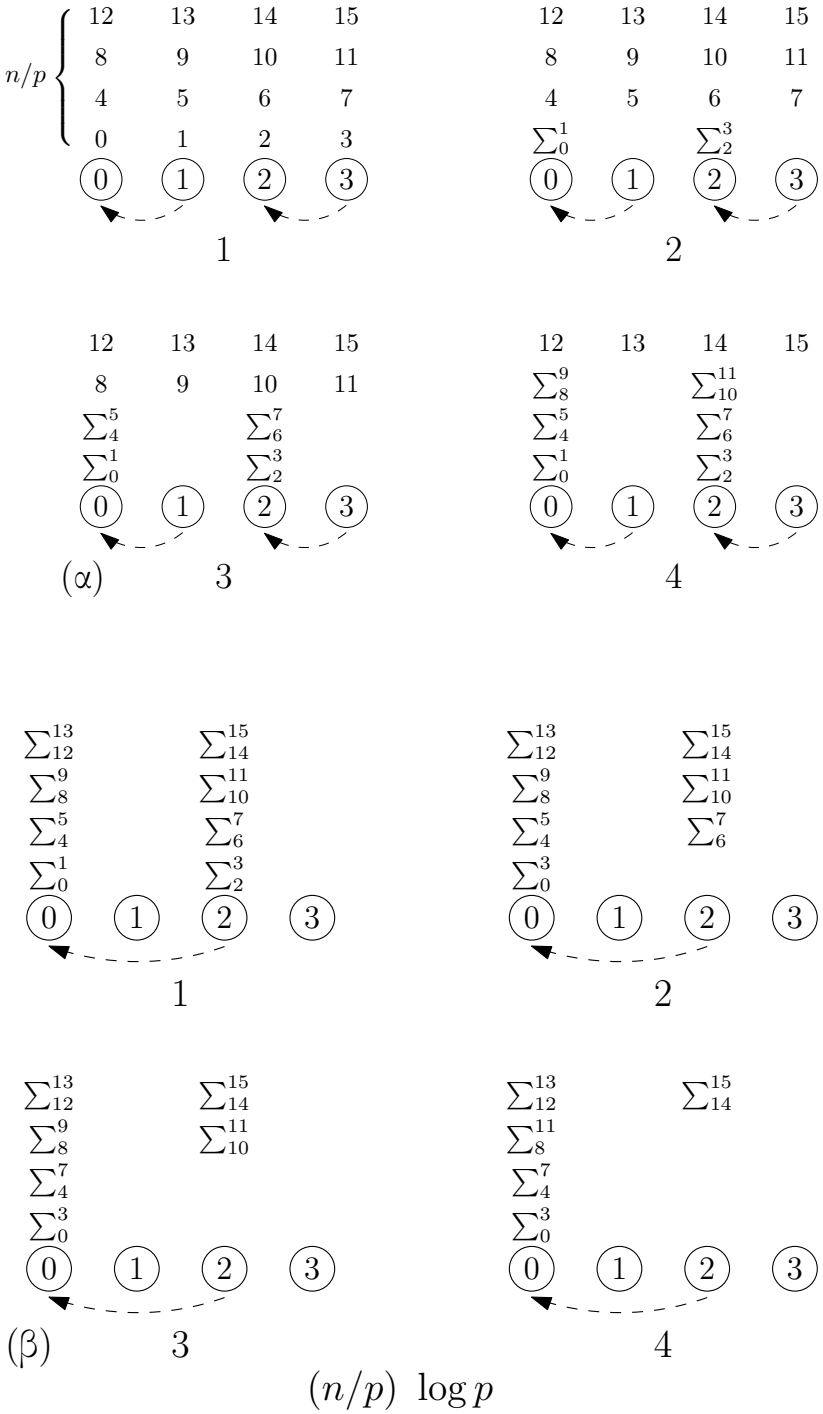


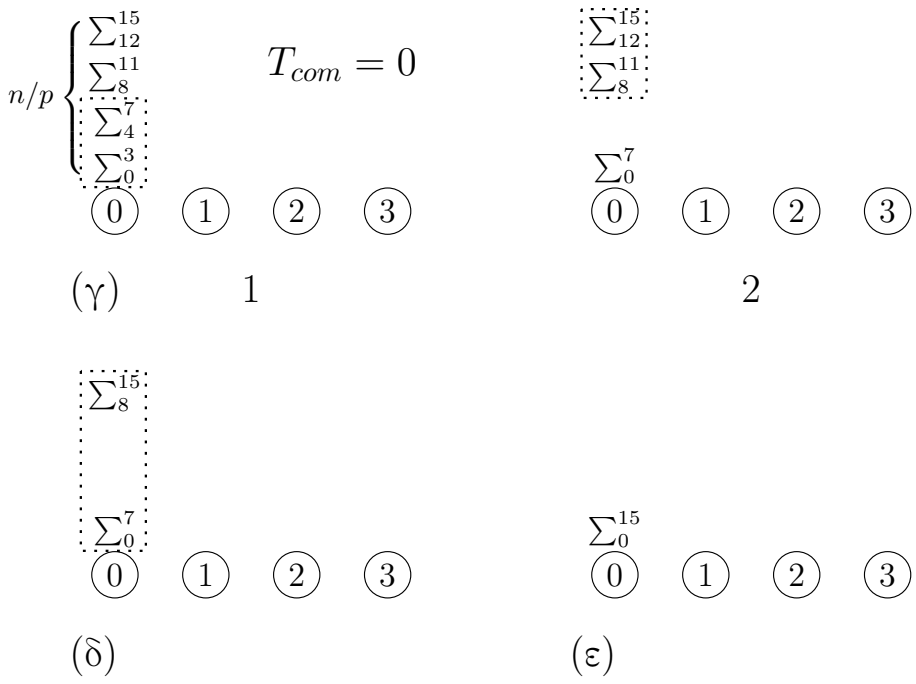
Σχήμα 1.44: Πρόσθεση n αριθμών σε ένα Υπερκύβο με n επεξεργαστές

$$T_p = \Theta(\log n), \quad S_p = \Theta(\frac{n}{\log n})$$

$$E_p = \Theta(\frac{1}{\log n}), \quad c_p = \Theta(n \log n)$$

όχι βέλτιστο κόστος


n/p
12 13 14 15
8 9 10 11
4 5 6 7
0 1 2 3
$\sum_0^1$ $\sum_2^3$
$\sum_4^5$ $\sum_6^7$
$\sum_8^9$ $\sum_{10}^{11}$
$\sum_{12}^{13}$ $\sum_{14}^{15}$
$\sum_0^3$
$\sum_8^{11}$
$\sum_4^7$
$\sum_0^7$
0 1 2 3
1
2
(α)
3
4
(β)
3
4
$(n/p)\ \log p$

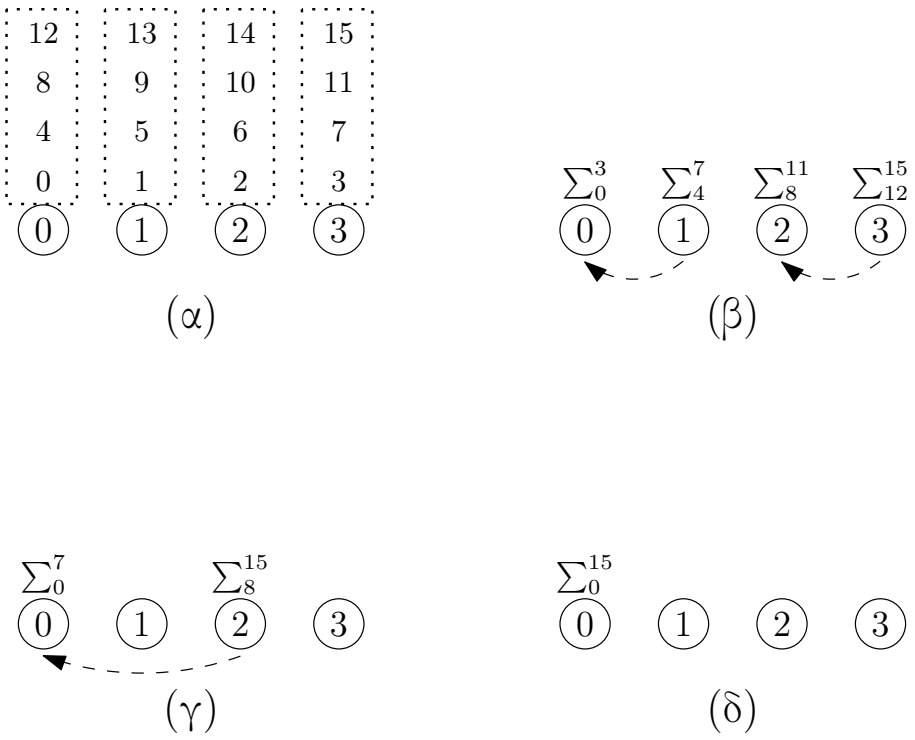


Σχήμα 1.45: Πρόσθεση n αριθμών σε υπερκύβο με p επεξεργαστές, όπου $p < n$. Τα n, p είναι δυνάμεις του 2

$$T_p = \Theta\left(\left(\frac{n}{p}\right) \log p\right)$$

$$c_p = \Theta(n \log p) > \Theta(n) = c_1$$

όχι βέλτιστο κόστος

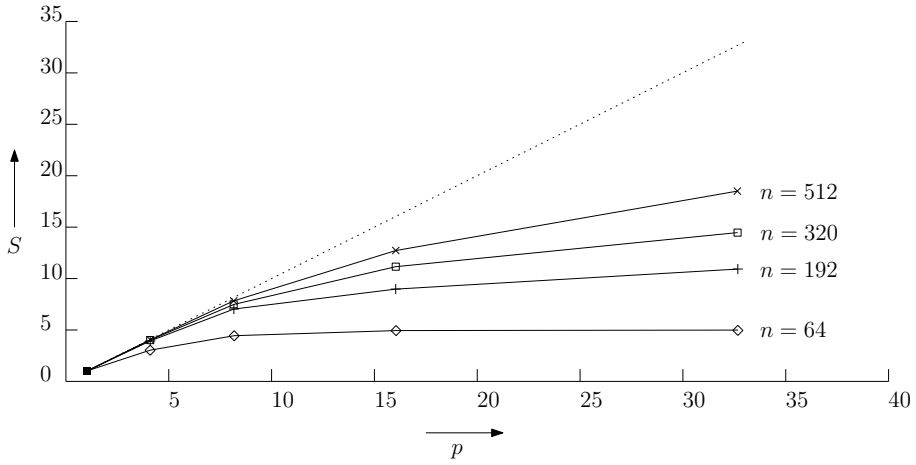


Σχήμα 1.46: Διαφορετικός τρόπος για την εύρεση του προηγούμενου αθροίσματος

$$T_p = \Theta\left(n/p + \log p\right), \quad c_p = \Theta(n + p\log p)$$
$$\text{Αν} \quad n = \Omega(p\log p) \quad \text{τότε} \quad c_p = \Theta(n)\ !$$

$$T_p = \frac{n}{p} + 2t_{com}\log p, \quad S = \frac{np}{n + 2p\log p}$$

$$E_p = \frac{n}{n + 2p\log p} \qquad \begin{cases} c_p = \mathcal{O}(n + 2p\log p) \\ n = \Omega(p\log p) \end{cases} \quad \text{βέλτιστο κόστος}$$



n	p = 1	p = 4	p = 8	p = 16	p = 32
64	1.0	.80	.57	.33	.17
192	1.0	.92	.80	.60	.38
320	1.0	.95	.87	.71	.50
512	1.0	.97	.91	.80	.62

Πίνακας 1.1: E_p

$$n = 8p \log p$$

- Η ταχύτητα δεν αυξάνει γραμμικά όταν αυξάνει το πλήθος των επεξεργαστών
- Για μεγαλύτερα προβλήματα η ταχύτητα και η αποδοτικότητα αυξάνουν

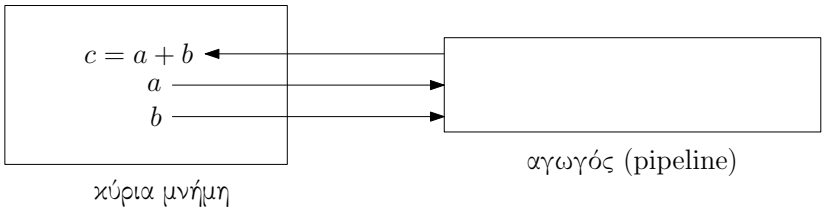
$N = 2^n,\ n \geqslant 1.\ x_1, x_2, \ldots, x_N \to M_1, M_2, \ldots, M_N$
Το τελικό αποτέλεσμα στην M_1.

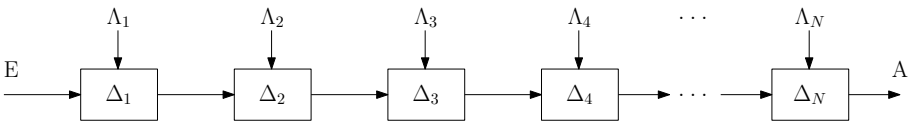
Διαδικασία 6: ASSOCIATIVE FAN-IN $(\mathrm{inc}, \mathrm{N})$

```
inc = 1 για j = 1 έως log N κάνε
    για i ∈ {1 + 2k · inc | k = 0, 1, 2, ..., N/2^j − 1} κάνε παράλληλα
        // επεξεργαστής p_i
        διάβασε M_i και M_{i_time}
        πρόσθεσε περιεχόμενα M_i και M_{inc+i}
        γράψε το άθροισμα στο M_i
        inc = 2inc
    τέλος
τέλος
```

j	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
0	1	2	3	4	5	6	7	8
1	3		7		11		15	
2	10				26			
3	36							

1.7 Διανυσματικοί Υπολογιστές





$E =$ εντολή$_i$ $\quad \Delta_i =$ δεδομένο$_i$

$\Lambda_i = i -$ οστή λειτουργία, $\quad$ Α = αποτέλεσμα

Σωλήνωση τριών τμημάτων

Περίοδος χρόνου	**1**	**2**	**3**	**4**		**n**	**n + 1**
πάρε	a_1	a_2	a_3	a_4	...	a_n	-
πάρε και πρόσθεσε	-	$a_1 + b_1$	$a_2 + b_2$	$a_3 + b_3$		$a_{n-1} + b_{n-1}$	
Εκτύπωσε	-	-	$a_1 + b_1$	$a_2 + b_2$		$a_{n-2} + b_{n-2}$	

Σχήμα 1.47

Κεφάλαιο 2

ΕΠΙΛΟΓΗ (Selection)

2.1 Το πρόβλημα της επιλογής

Δίνεται μια ακολουθία S από n στοιχεία και ένας ακέραιος k, όπου $1 \leqslant k \leqslant n$. Να προσδιοριστεί το k μικρότερο στοιχείο στην S.

Υπόθεση: Η ανάπτυξη του παράλληλου αλγόριθμου θα γίνει για το SM SIMD μοντέλο.

Ορισμός: Τα στοιχεία ενός συνόλου A ικανοποιούν μια γραμμική διάταξη $<$ τότε και μόνον τότε αν

(i) για $a, b \in A, a < b, a = b$ ή $b < a$ και

(ii) για $a, b, c \in A$, αν $a < b$ και $b < c$, τότε $a < c$

Παράδειγμα: Το σύνολο των ακεραίων αριθμών ικανοποιεί μια γραμμική διάταξη.

Βαθμός: Για μια ακολουθία $S = \{s_1, s_2, \cdots, s_n\}$ της οποίας τα στοιχεία είναι και στοιχεία ενός συνόλου με γραμμική διάταξη, ο *βαθμός* ενός στοιχείου s_i του S είναι το πλήθος των στοιχείων του S που προηγούνται του s_i συν 1.

Παράδειγμα
$S = \{9, -4, 3, -6, 7, 0\}$ ο βαθμός του 0 είναι 3.
Αν $s_i = s_j$ τότε το s_i προηγείται του s_j τότε και μόνον τότε αν $i < j$.

Επιλογή

Δίνεται μία ακολουθία $S = \{s_1, s_2, \cdots, s_n\}$, της οποίας τα στοιχεία έχουν παρθεί από ένα γραμμικά διατεταγμένο σύνολο, και ένας ακέραιος k, $\{1 \leqslant k \leqslant n\}$. Ζητείται ο προσδιορισμός του στοιχείου της S με βαθμό k. Το στοιχείο με βαθμό k θα συμβολίζεται με $s_{(k)}$.

Πολυπλοκότητα (κάτω όριο)
Αν τα στοιχεία του S ήταν ταξινομημένα, δηλαδή

$$S = \{s_{(1)}, s_{(2)}, \cdots, s_{(n)}\}$$

τότε το $s_{(k)}$ λαμβάνεται σε ένα βήμα.
Προφανώς δεν υποθέτουμε ότι έχουμε αυτή την περίπτωση ούτε επιθυμούμε να ταξινομήσουμε τα στοιχεία του S.

Αν $k = 1$ ή $k = n$, τότε εξετάζοντας όλα τα στοιχεία της S διατηρώντας κάθε φορά το μικρότερο $(k = 1)$ (ή το μεγαλύτερο $(k = n)$) βρίσκεται το ζητούμενο στοιχείο. Ωστόσο αυτός ο αλγόριθμος δεν μπορεί να εφαρμοστεί αν $1 < k < n$. Στην καλύτερη περίπτωση λοιπόν η πολυπλοκότητα του αλγορίθμου είναι

$$\Omega(n) \quad \text{(κάτω όριο)}$$

2.2 Ένας ακολουθιακός αλγόριθμος

- Αναδρομικός αλγόριθμος
- Προσέγγιση ¨διαίρει και βασίλευε¨ (divide and conquer)
- απορρίπτει ένα πλήθος στοιχείων σε κάθε στάδιο

Διαδικασία 7: SEQUENTIAL SELECT (S, k)

Βήμα 1
αν $|S| < Q$ **τότε**
| ταξινόμησε την S και επίστρεψε το $k^{στο}$ στοιχείο
αλλιώς
| υποδιαίρεσε την S σε $|S|/Q$ υποακολουθίες με Q στοιχεία η κάθε μία (μέχρι $Q-1$ εναπομείναντα στοιχεία)
τέλος

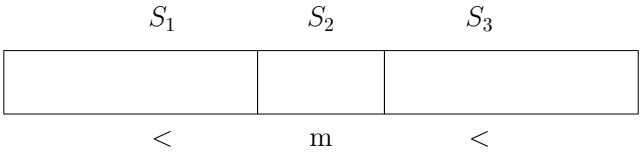
Βήμα 2
Ταξινόμησε κάθε υποακολουθία και υπολόγισε το μέσο της

Βήμα 3
Κάλεσε την SEQUENTIAL SELECT αναδρομικά για τον υπολογισμό του m, το μέσο των $|S|/Q$ μέσων που βρέθηκαν στο βήμα 2

Βήμα 4
Δημιούργησε τρεις υποακολουθίες $S_1, S_2,$ και S_3 με στοιχεία της S μικρότερα, ίσα και μεγαλύτερα από το m, αντίστοιχα

Βήμα 5
αν $|S_1| \geqslant k$ *{Το* $k^{στό}$ *στοιχείο της S πρέπει να είναι στο* S_1*}* **τότε**
| κάλεσε την SEQUENTIAL SELECT αναδρομικά για να βρείς το $k^{στό}$ στοιχείο του S_1
αλλιώς
| **αν** $|S_1| + |S_2| \geqslant k$ **τότε**
| | επίστρεψε m
| | {διότι το k στοιχείο ανήκει στο S_2}
| **αλλιώς**
| | κάλεσε την SEQUENTIAL SELECT αναδρομικά για να βρείς το $(k - |S_1| - |S_2|)^{στό}$ στοιχείο του S_3
| | {διότι το k στοιχείο ανήκει στο S_3}
| **τέλος**
τέλος

(Divide and conquer)



Σχήμα 2.1: Βήμα 5

Παράδειγμα SEQUENTIAL SELECT (S,k) Δίνεται η ακολουθία

$$S = \{1, 14, 3, 4, 2, 6, 7, 4, 5, 8, 2, 1, 5, 12, 4, 3, 2, 8, 9, 10, 5, 2, 12\}$$

και ένας ακέραιος k, $1 < k < 23$. Ζητείται να προσδιοριστεί το στοιχείο της ακολουθίας με βαθμό 19, δηλαδή $k = 19$.

Λύση: Επιλέγουμε το Q ώστε να πληρείται η σχέση $\frac{n}{Q} + \frac{3n}{4} < n \Leftrightarrow Q > 4$. Έστω $Q = 5$.

Βήμα 1: Επειδή $|S| = 23 > 5 = Q$, θα υποδιαιρέσουμε την S σε $|S|/Q$ υποακολουθίες, δηλαδή σε 4 υποακολουθίες με 5 στοιχεία και σε 1 υποακολουθία με 3 στοιχεία.

1	14	3	4	2		6	7	4	5	8		2	1	5	12	4		3	2	8	9	10		5	2	12

Βήμα 2: α) Ταξινόμηση κάθε υποακολουθίας β) Βρίσκουμε το μέσο m_i κάθε υποακολουθίας

1	2	3	4	14		4	5	6	7	8		1	2	4	5	12		2	3	8	9	10		2	5	12
		$m_1 = 3$						$m_2 = 6$						$m_3 = 4$						$m_4 = 8$						$m_5 = 5$

Βήμα 3: Κλήση της Sequential Select για τον υπολογισμό του μέσου των μέσων.

B1: | 3 | 6 | 4 | 8 | 5 | → B2α: | 3 | 4 | 5 | 6 | 8 | → B2β: | $m = 5$ |

m_1 m_2 m_3 m_4 m_5 — Ταξινόμηση των m_i — Υπολογισμός μέσου

Βήμα 4: Δημιουργία 3 υποακολουθιών S_1, S_2, S_3 με στοιχεία $<, =, > 5$, αντίστοιχα.

1	3	4	2	1	2	1	4	3	2	2		5	5	5		14	6	7	8	12	8	9	10	12
						S_1								S_2						S_3				

Παρατηρούμε ότι $|S_1| = 11$, $|S_2| = 3$, $|S_3| = 9$.

Βήμα 5: Παρατηρούμε ότι $k = 19 > |S_1| + |S_2| = 14$ οπότε καλούμε την Sequential Select στην S_3 για να βρούμε το $k_{new}^{(1)} = k - |S_1| - |S_2| = 19 - 14 \Rightarrow k_{new}^{(1)} = 5$, δηλαδή το 5^o στοιχείο της ακολουθίας S_3.

Β1: Για $Q = 5$ είναι $|S_3| = 9 > 5 = Q$ οπότε $|S_3|/Q = 9/5 = 1.8$ δηλαδή θα υποδιαιρέσουμε την S_3 σε 1 υποακολουθία με 5 στοιχεία και σε 1 υποακολουθία με 4 στοιχεία.

14	6	7	8	12

8	9	10	12

Β2: α)Ταξινόμηση κάθε υποακολουθίας β)Εύρεση μέσου κάθε υποακολουθίας.

6	7	8	12	14

$m_1' = 8$

8	9	10	12

$m_2' = 9$

Β3: Το μέσο των 8,9 είναι το $m' = 8$

Β4: Δημιουργία 3 υποακολουθιών S_1', S_2', S_3' με στοιχεία $<, =, > 8$, αντίστοιχα.

6	7

S_1'

8	8

S_2'

14	12	9	10	12

S_3'

Β5: Παρατηρούμε ότι $|S_1'| = 2, |S_2'| = 2, |S_3'| = 5$. Παρατηρούμε ότι $k_{new}^{(1)} = 5 > |S_1'| + |S_2'| = 4$ οπότε καλούμε τη Sequential Select στην S_3' για να βρούμε το $k_{new}^{(2)} = k_{new}^{(1)} - |S_1'| - |S_2'| = S - 4 \Rightarrow k_{new}^{(2)} = 1$, δηλαδή το 1^o στοιχείο της ακολουθίας S_3'.

Β1': Είναι $|S_3'| = 5 = Q$ οπότε ταξινομούμε την S_3' και επιστρέφουμε το 1^o στοιχείο. Είναι

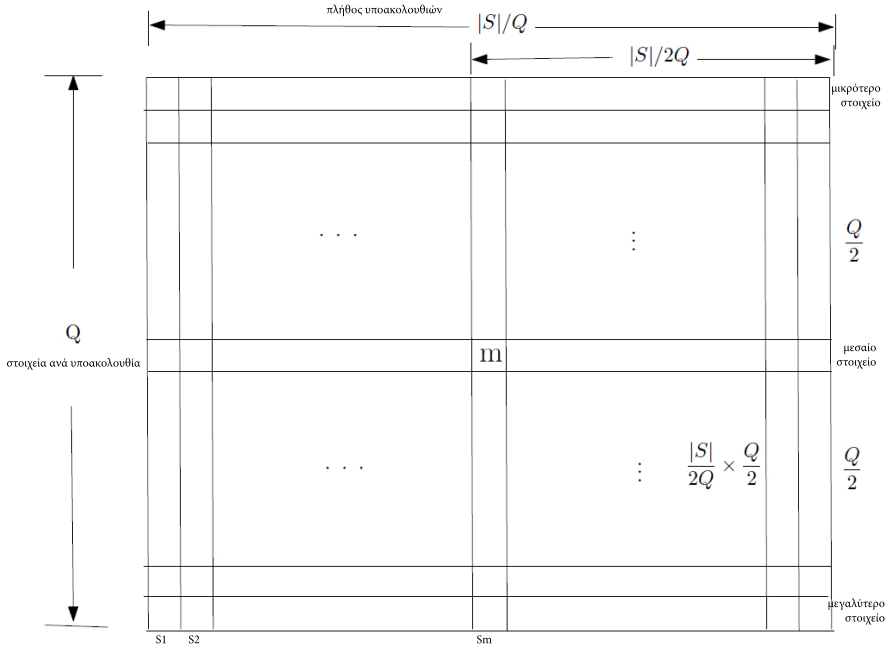
14	12	9	10	12

S_3'

9	10	12	12	14

S_3' ταξινομημένη

Το 1^o στοιχείο της S_3' μετά την ταξινόμησή της είναι το 9 και κατά συνέπεια αυτό είναι το ζητούμενο στοιχείο.



Σχήμα 2.2: Κύρια ιδέα της διαδικασίας SEQUENTIAL SELECT

Ανάλυση Πολυπλοκότητας της διαδικασίας SEQUENTIAL SELECT

Βήμα 1: Ταξινόμηση της S όταν $|S| < Q$ απαιτεί σταθερό χρόνο $O(1)$. Διαφορετικά, υποδιαίρεση της S απαιτεί χρόνο $c_1 n$.
Βήμα 2: Η ταξινόμηση κάθε $|S|/Q$ υποακολουθίας (έχει Q στοιχεία) απαιτεί σταθερό χρόνο.
Για όλες τις υποακολουθίες απαιτείται χρόνος $c_2 n$.
Βήμα 3: $t(n/Q), t(n)$: χρόνος εκτέλεσης της SEQUENTIAL SELECT.
Βήμα 4: Απαιτείται χρόνος $c_3 n$.
Βήμα 5: $(|S|/2Q) \times (Q/2) = |S|/4$ στοιχεία της S (βλέπε σχήμα) είναι μεγαλύτερα ή ίσα με m. Συνεπώς, $|S_1| \leqslant 3|S|/4$. Όμοια, $|S_3| \leqslant 3|S|/4$. Συνεπώς μια αναδρομική κλήση της SEQUENCIAL SELECT απαιτεί $t(3n/4)$ χρόνο.

Συνολικά:

$$t(n) = c_4 n + t(n/Q) + t(3n/4)$$

, όπου

$$c_4 = c_1 + c_2 + c_3$$

.

Καθορισμός του Q: Αν επιλεχθεί Q τέτοιο ώστε

$$n/Q + 3n/4 < n$$

τότε οι δυο αναδρομικές κλήσεις στη διαδικασία εφαρμόζονται σε φθίνουσες ακολουθίες. Οποιαδήποτε τιμή του $Q \geqslant 5$ ικανοποιεί την ανωτέρω ανισότητα. Για $Q = 5$ έχουμε:

$$t(n) = c_4 n + t(n/5) + t(3n/4)$$

Υποθέτοντας

$$t(n) \leqslant c_5 n$$

έχουμε

$$t(n) \leqslant c_4 n + c_5(19n/20)$$

και για $c_5 = 20c_4$:

$$t(n) \leqslant c_5(n/20) + c_5(19n/20) = c_5 n$$

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

βέλτιστος χρόνος!

2.3 Δύο χρήσιμοι αλγόριθμοι

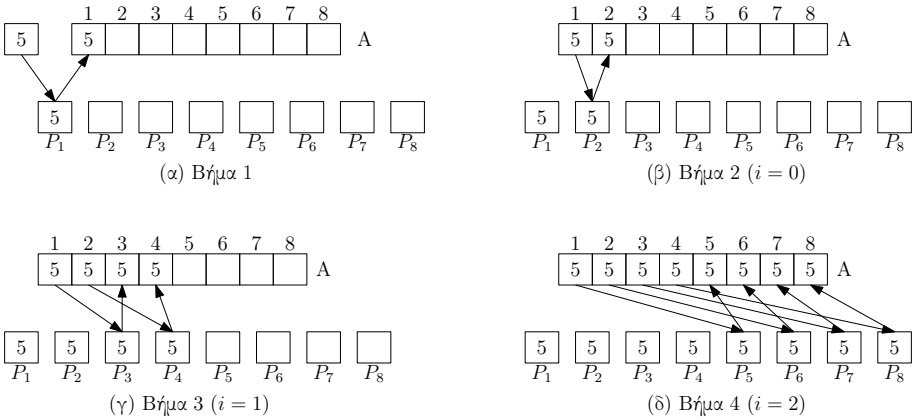
Στο EREW SM SIMD μοντέλο δεν είναι δυνατό να προσπελάσουν δύο επεξεργαστές ταυτόχρονα την ίδια τοποθεσία μνήμης.

(i) **Μετάδοση (broadcasting):** Όλοι οι επεξεργαστές χρειάζονται να διαβάσουν ένα δεδομένο από μια συγκεκριμένη τοποθεσία της κοινής μνήμης.

(ii) **Επικοινωνία (όλοι με ένα):** Κάθε επεξεργαστής πρέπει να κάνει υπολογισμούς στα δεδομένα τα οποία υπάρχουν στους άλλους επεξεργαστές και συνεπώς χρειάζεται να λάβει αυτά τα δεδομένα.

Οι δύο αυτές λειτουργίες δεν μπορούν να εκτελεστούν σε ένα βήμα στο EREW μοντέλο και πρέπει να προσομοιωθούν.

Υπόθεση: Ν επεξεργαστές $P_1, P_2, \cdots, P_N$ είναι διαθέσιμοι σε ένα EREW SM SIMD υπολογιστή.



Σχήμα 2.3: Διαμοίραση δεδομένου σε 8 επεξεργαστές μέσω της διαδικασίας BROADCAST

Διαδικασία 8: BROADCAST $(\mathrm{D}, \mathrm{N}, \mathrm{A})$

Βήμα 1
Ο επεξεργαστής P_1
(i) διαβάζει την τιμή από την D,
(ii) την αποθηκεύει στην μνήμη του, και
(iii) τη γράφει στην $\mathrm{A}(1)$.

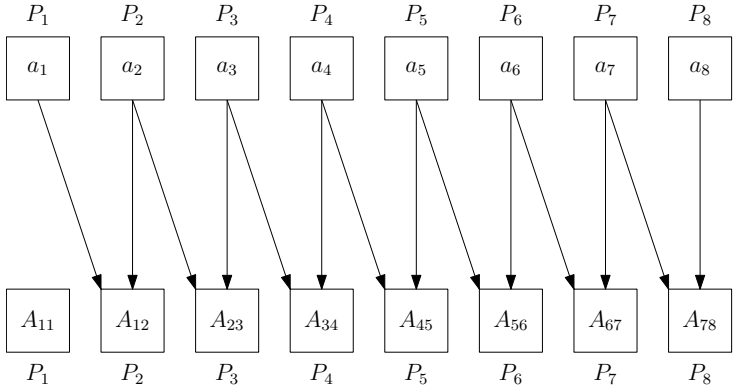
Βήμα 2
για $\mathrm{i} = 0$ **έως** $(\log \mathrm{N} - 1)$ **κάνε**
  για $\mathrm{j} = 2^{\mathrm{i}} + 1$ **έως** $2^{\mathrm{i}+1}$ **κάνε παράλληλα**
    Ο επεξεργαστής P_j
    (i) διαβάζει την τιμή από την $\mathrm{A}(\mathrm{j} - 2^{\mathrm{i}})$,
    (ii) την αποθηκεύει στην μνήμη του, και
    (iii) τη γράφει στην $\mathrm{A}(\mathrm{j})$.
  τέλος
τέλος

Υπολογισμός των αθροισμάτων
Υπόθεση: P_i έχει το α_i στην τοπική μνήμη.

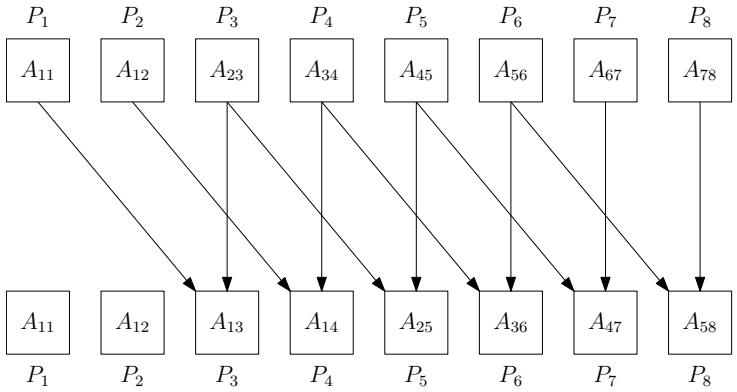
$$\mathrm{P}_\mathrm{i} := \alpha_1 + \alpha_2 + \cdots + \alpha_\mathrm{i}$$

Διαδικασία 9: ALLSUMS $(\alpha_1, \alpha_2, \cdots, \alpha_N)$

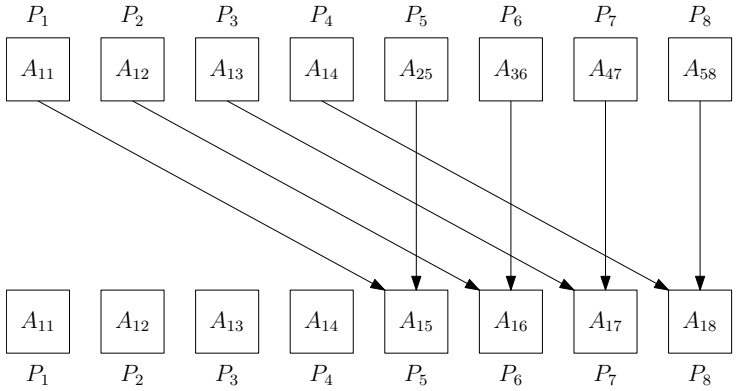
για $j = 0$ **έως** $(\log N - 1)$ **κάνε**
  για $i = 2^j + 1$ **έως** N **κάνε παράλληλα**
    Ο επεξεργαστής P_i
    (i) διαβάζει το α_{i-2^j} από το P_{i-2^j} μέσω κοινής μνήμης, και
    (ii) αντικαθιστά το α_i, με το $\alpha_{i-2^j} + \alpha_i$.
  τέλος
τέλος



(α) $j = 0$



(β) $j = 1$



(γ) $j = 2$

Σχήμα 2.4: Υπολογισμός αθροίσματος με χρήση της διαδικασίας ALL-SUMS

2.4 Ένας αλγόριθμος για την παράλληλη επιλογή (EREW SM SIMD)

Υποθέσεις:

1. $S = \{S_1, S_2, \cdots, S_n\}, k, 1 \leqslant k \leqslant n$
2. $P_1, P_2, \cdots, P_n$ επεξεργαστές
3. Κάθε επεξεργαστής έχει λάβει το n και έχει υπολογίσει το x από την $N = n^{1-x}$, όπου $0 < x < 1$.
4. Κάθε ένας από τους n^{1-x} επεξεργαστές μπορεί να αποθηκεύσει n^x στοιχεία στην τοπική μνήμη του
5. Κάθε επεξεργαστής μπορεί να εκτελέσει τις SEQUENTIAL SELECT, BROADCAST και ALLSUMS.
6. M είναι ένας μονοδιάστατος πίνακας στην κοινή μνήμη μήκους N.

Διαδικασία 10: PARALLEL SELECT (S, k)

Βήμα 1
αν $|S| \leqslant 4$ **τότε**
| ο P_1 χρησιμοποιεί το πολύ πέντε συγκρίσεις για να επιστρέψει το $k^{στο}$ στοιχείο
αλλιώς
| (i) Η ακολουθία S υποδιαιρείται σε $|S|^{1-x}$ υποακολουθίες S_i, μήκους $|S|^x$ η κάθε μία, όπου $1 \leqslant i \leqslant |S|^{1-x}$.
| (ii) Η υπακολουθία S_i ανατίθεται στον επεξεργαστή P_i.
τέλος

Βήμα 2
για $i = 1$ **έως** $|S|^{1-x}$ **κάνε παράλληλα**
| (2.1) Ο P_i αποκτά το μέσο m_i, δηλαδή το $\lceil |S_i|/2 \rceil$-οστό στοιχείο της υποακολουθίας του SEQUENTIAL SELECT $(S_i, \lceil |S_i|/2 \rceil)$.
| (2.2) Ο P_i αποθηκεύει το m_i, στο M(i)
τέλος

Βήμα 3
{Το υποπρόγραμμα καλείται αναδρομικά για να αποκτηθεί το μέσο m του M}
PARALLEL SELECT $(M, \lceil |M|/2 \rceil)$.

Βήμα 4
Η ακολουθία S υποδιαιρείται σε τρεις υποακολουθίες:
$L = \{s_i \in S : s_i < m\}$,
$E = \{s_i \in S : s_i = m\}$, και
$G = \{s_i \in S : s_i > m\}$.

Βήμα 5
αν $|L| \geqslant k$ **τότε**
| PARALLEL SELECT (L,k)
αλλιώς
| **αν** $|L| + |E| \geqslant k$ **τότε**
| | επίστρεψε m
| **αλλιώς**
| | PARALLEL SELECT $(G, k - |L| - |E|)$
| **τέλος**
τέλος

Ανάλυση της διαδικασίας PARALLEL SELECT

Βήμα 1: Για την εκτέλεση του βήματος αυτού απαιτείται η διεύθυνση αρχής A της ακολουθίας S στην κοινή μνήμη, το μέγεθος $|S|$ και η τιμή k. Οι τιμές αυτές μεταδίδονται σε όλους τους επεξεργαστές με την BROADCAST σε $O(\log n^{1-x})$ χρόνο. Αν $|S| \leqslant 4$, τότε ο P_1 επιστρέφει το k - οστό στοιχείο σε σταθερό χρόνο. Διαφορετικά ο P_i υπολογίζει τη διεύθυνση του πρώτου και του τελευταίου στοιχείου της S_i από τους

$$A + (i-1)n^x$$

και

$$A + in^x - 1$$

σε σταθερό χρόνο. Συνεπώς το βήμα 1 απαιτεί $c_1 \log n$ μονάδες χρόνου για κάποια σταθερά c_1.

Βήμα 2: Η SEQUENTIAL SELECT βρίσκει το μέσο μιας ακολουθίας μήκους n^x σε $c_2 n^x$ μονάδες χρόνου για κάποια σταθερά c_2.

Βήμα 3: Καλείται η PARALLEL SELECT για μια ακολουθία μήκους n^{1-x}. Συνεπώς το παρόν βήμα απαιτεί $t(n^{1-x})$ χρόνο.

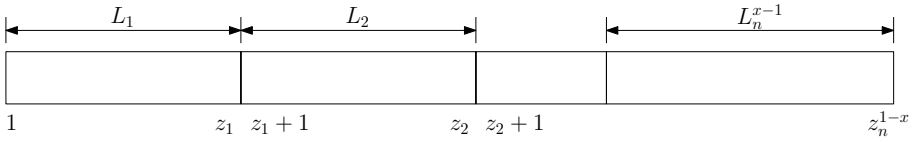
Βήμα 4: Η ακολουθία S χωρίζεται στις υποακολουθίες L, E και G ως εξής:

(i) Το m μεταδίδεται σε όλους τους επεξεργαστές σε $O(\log n^{1-x})$ χρόνο με τη χρήση της BROADCAST.

(ii) Κάθε επεξεργαστής P_i χωρίζει την S_i σε τρείς υποακολουθίες L_i, E_i και G_i με στοιχεία μικρότερα, ίσα και μεγαλύτερα από το m, αντίστοιχα. Αυτή η εργασία μπορεί να γίνει σε χρόνο γραμμικό ως πρός το μέγεθος της S_i, δηλαδή σε $O(n^x)$ χρόνο.

(iii) Οι υποακολουθίες L_i, E_i και G_i συγχωνεύονται για να σχηματίσουν τις L, E και G. Στη συνέχεια δείχνεται πως αυτή η εργασία μπορεί να γίνει για την L_i. Όμοια και με τον ίδιο χρόνο μπορεί να γίνει για τη συγχώνευση των E_i και G_i, αντίστοιχα. Θέτοντας $\alpha_i = |L_i|$, υπολογίζεται το άθροισμα

$$z_i = \sum_{j=1}^{i} \alpha_j, \quad 1 \leqslant i \leqslant n^{1-x}$$



$$z_1 = a_1 = |L_1|, \quad z_2 = a_1 + a_2 = |L_1| + |L_2|$$

Σχήμα 2.5: Σχηματισμός της L

Τα αθροίσματα αυτά υπολογίζονται σε $O(\log n^{1-x})$ χρόνο από τους n^{1-x} επεξεργαστές, χρησιμοποιώντας την ALLSUMS. Όλοι οι επεξεργαστές συγχωνεύουν τις L_i υποακολουθίες τους για τον σχηματισμό της L.

Ο επεξεργαστής P_i αντιγράφει την L_i στην L ξεκινώντας από τη θέση $z_{i-1}+1$, όπου $z_0 = 0$. Η εργασία αυτή μπορεί να γίνει σε $O(n^x)$ χρόνο. Συνεπώς ο χρόνος που απαιτείται για το παρόν βήμα είναι $c_3 n^x$ για κάποια σταθερά c_3.

Βήμα 5: Το μέγεθος της L που χρειάζεται στο βήμα αυτό έχει υπολογιστεί στο βήμα 4 και είναι z_n^{1-x}. Η ίδια παρατήρηση ισχύει για τα μεγέθη των E και G. Στη συνέχεια πρέπει να υπολογιστεί ο χρόνος που απαιτείται για κάθε μία από τις δύο αναδρομικές κλήσεις της PARALLEL SELECT. *Επειδή το* m *είναι το μέσο του* M*, έπεται ότι* $n^{1-x}/2$ *στοιχεία είναι μεγαλύτερα από αυτό. Επίσης κάθε στοιχείο του* M *είναι μικρότερο από* $n^x/2$ *τουλάχιστον στοιχεία της* S. Άρα $\frac{n^{1-x}}{2} \cdot \frac{n^x}{2} = \frac{n}{4}$ της L είναι τουλάχιστον μεγαλύτερα ή ίσα του m. Συνεπώς $|L| \leqslant \frac{3n}{4}$. Όμοια, $|G| \leqslant \frac{3n}{4}$. Τελικά, το παρόν βήμα απαιτεί το πολύ $t(3n/4)$ χρόνο.

Η προηγούμενη ανάλυση έχει σαν αποτέλεσμα την ακόλουθη αναδρομική εξίσωση για το συνολικό χρόνο

$$t(n) = c_1 \log n + c_2 n^x + t(n^{1-x}) + c_3 n^x + t(3n/4)$$

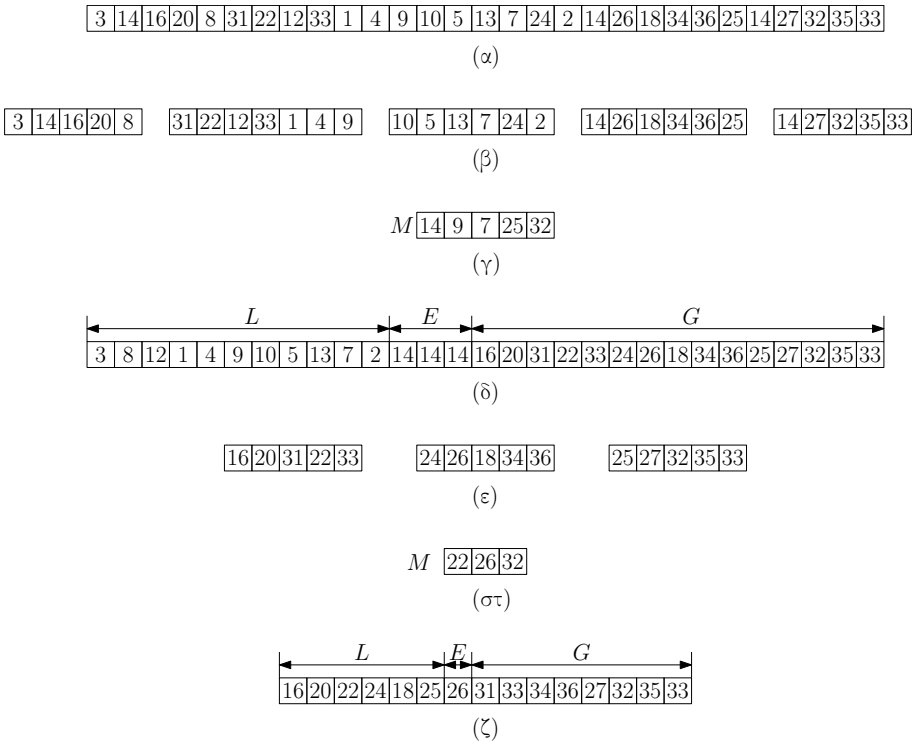
της οποίας η λύση είναι $t(n) = O(n^x)$ για $n > 4$. Επίσης
$c(n) = p(n) \cdot t(n) = n^{1-x} \cdot O(n^x) = O(n)$. Το κόστος αυτό είναι βέλτιστο αφού $\Omega(n)$.

Παράδειγμα

$S = \{3, 14, 16, 20, 8, 31, 22, 12, 33, 1, 4, 9, 10, 5, 13, 7, 24, 2, 14, 26, 18, 34, 36, 25, 14, 27, 32, 35, 33\}$

$$n = 29, \quad k = 21, \quad N = 5$$

Συνεπώς, $|S|^{1-x} = 5$ ή $29^{1-x} = 5$ ή $1 - x = 0.47796$



Σχήμα 2.6: Επιλογή 21 στοιχείων μιας ακολουθίας με τη διαδικασία PARALLEL SELECT

Αφού (Σχήμα 2.6(γ)) $|L| = 11, |E| = 3, |L| + |E| < k$ αναδρομική κλήση με $S = G$ και $k = 21 - (11 + 3) = 7$.
Επειδή $|G| = 15$ χρησιμοποιούνται $15^{1-x} = 3.6485$, δηλαδή 3 επεξεργαστές (Σχήμα 2.6(ζ)).

Επειδή $|L| = 6, |E| = 1, |L| + |E| = 7 = k$ άρα $m = 26$.

- $p(n) < n^{1/2}$, προσαρμοστικός
- $t(n)$ μικρό, προσαρμοστικό
- $c(n) = p(n) \times t(n)$ βέλτιστο κόστος όταν $c(n) = \Omega(n)$ του ακολουθιακού αλγόριθμου

Κεφάλαιο 3

ΣΥΓΧΩΝΕΥΣΗ (merging)

Έστω

$$A = \{\alpha_1, \alpha_2, \cdots, \alpha_r\} \quad B = \{b_1, b_2, \cdots, b_s\}$$

ταξινομημένες ακολουθίες αριθμών σε μη φθίνουσα σειρά. Να σχηματισθεί η ακολουθία

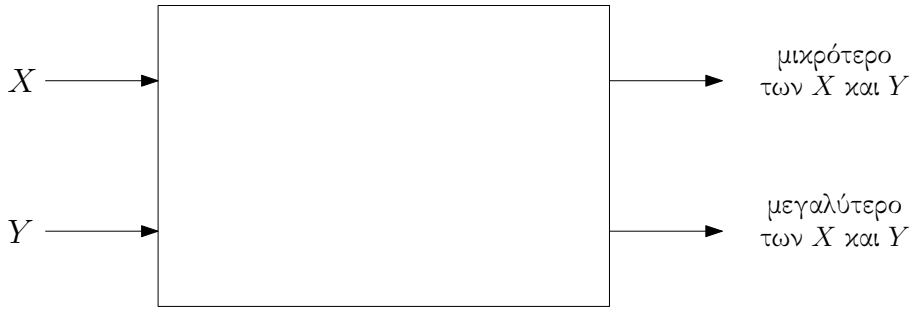
$$C = \{c_1, c_2, \cdots, c_{r+s}\}$$

ταξινομημένη σε μη φθίνουσα σειρά. Αν $r = s = n$ (χειρότερη περίπτωση), τότε ο ακολουθιακός αλγόριθμος έχει πολυπλοκότητα $O(n)$, η οποία είναι βέλτιστη $(\Omega(n))$. Ο σκοπός μας είναι να μελετηθεί το πρόβλημα της συγχώνευσης σε μια ποικιλία παράλληλων υπολογιστικών μοντέλων. Με βάση το κάτω όριο ας σημειωθεί ότι απαιτείται $\Omega(n/N)$ χρόνος για οποιοδήποτε παράλληλο αλγόριθμο συγχώνευσης που χρησιμοποιεί N επεξεργαστές.

3.1 Ένα δίκτυο για συγχώνευση

(r, s) δίκτυο συγχώνευσης: ειδικού σκοπού παράλληλη αρχιτεκτονική.

- όλοι οι επεξεργαστές είναι ίδιοι, απλοί και επικοινωνούν με ένα ειδικού σκοπού δίκτυο (συγκριτές - comparators).

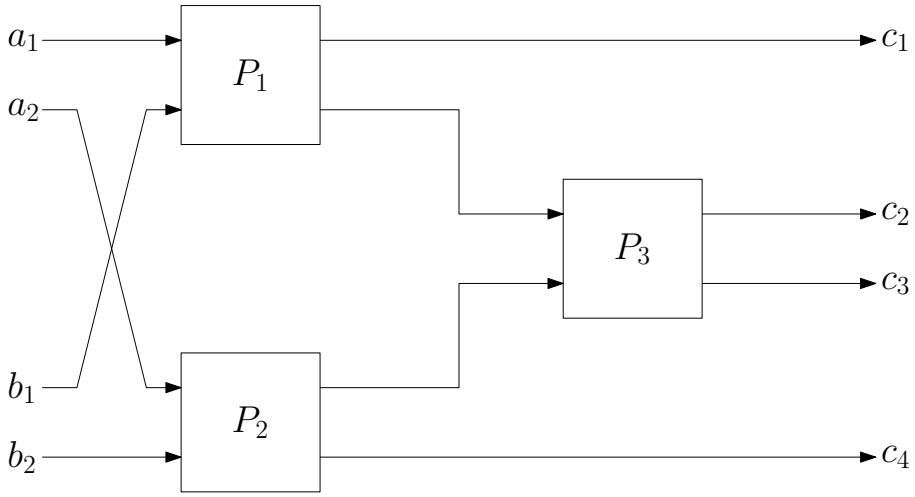


Σχήμα 3.1: Συγκριτής (περίπτωση $n = 1$)

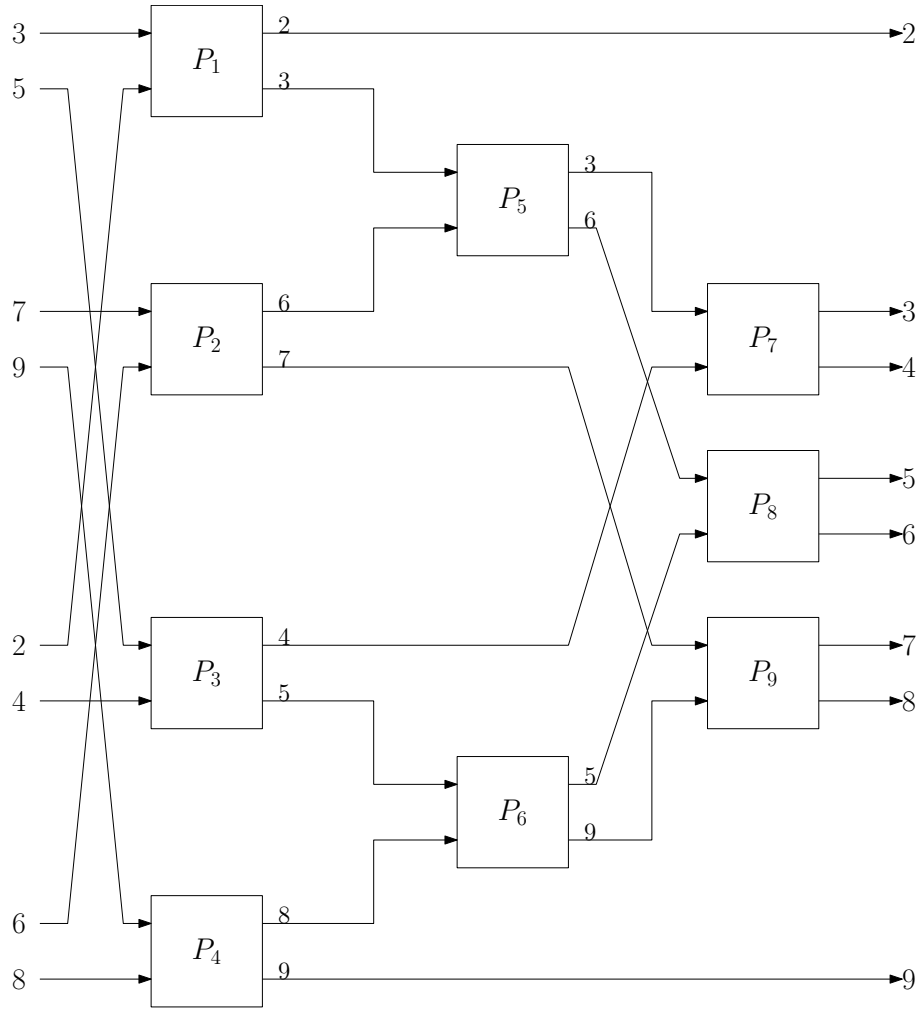
Υποθέσεις:

1. $r = s = n \geqslant 1$
2. $n = 2^k$

Με τη χρήση των συγκριτών θα κατασκευαστεί ένα δίκτυο.
Για $n = 1$: Αρκεί ένας συγκριτής.
Για $n = 2$: $A = \{\alpha_1, \alpha_2\}$ και $B = \{b_1, b_2\}$.
Για $n = 4$: $A = \{3, 5, 7, 9\}$ και $B = \{2, 4, 6, 8\}$.



Σχήμα 3.2: Περίπτωση $n = 2$ - Συγχώνευση δύο ακολουθιών με τέσσερα στοιχεία η καθεμία



Σχήμα 3.3: Περίπτωση $n = 3$

Γενικά ένα (n, n) δίκτυο συγχώνευσης σχηματίζεται από δύο $(n/2, n/2)$ άλλα δίκτυα. Τα στοιχεία στις περιττές θέσεις των A και B, δηλαδή $\{\alpha_1, \alpha_3, \alpha_5, \cdots, \alpha_{n-1}\}$ και $\{b_1, b_3, b_5, \cdots, b_{n-1}\}$ συγχωνεύονται στο ένα $(n/2, n/2)$ δίκτυο συγχώνευσης, δημιουργώντας την $\{d_1, d_2, d_3, \cdots, d_n\}$. Ταυτόχρονα για τα στοιχεία στις άρτιες θέσεις σχηματίζεται η

$$\{e_1, e_2, \cdots, e_n\}$$

με τη χρήση του άλλου $(n/2, n/2)$ δικτύου. Η τελική ακολουθία $\{c_1, c_2, \cdots, c_n\}$ λαμβάνεται από odd-even merging

$$c_1 = d_1, \quad c_{2n} = e_n, \quad c_{2i} = \min(d_{i+1}, e_i)$$

και

$$c_{2i+1} = max(d_{i+1}, e_i), \quad i = 1, 2, \cdots, n-1$$

Στην $\{d_1, d_2, \cdots, d_n\}$
i στοιχεία $\leqslant d_{i+1} \longrightarrow$
$2i$ στοιχεία των A και B $\leqslant d_{i+1}$
ή

$$c_{2i} \leqslant d_{i+1}$$

όμοια

$$c_{2i} \leqslant e_i$$

Επίσης

$$d_{i+1} \leqslant c_{2i+1}$$

και

$$e_i \leqslant c_{2i+1}$$

Επειδή

$$c_{2i} \leqslant c_{2i+1}$$

συνεπάγεται

$$c_{2i} = min(d_{i+1}, e_i), \quad c_{2i+1} = max(d_{i+1}, e_i)$$

Ανάλυση

Χρόνος:

$$\begin{cases} t(2) = 1, & \text{για} \quad n = 1 \\ t(2n) = t(n) + 1, & \text{για} \quad n > 1 \end{cases}$$

άρα

$$t(2n) = 1 + \log n \quad O(n) \quad \text{ακολουθιακός}$$

Επεξεργαστές:

$$\begin{cases} p(2) = 1, & \text{για} \quad n = 1 \\ p(2n) = 2p(n) + n - 1 & \text{για} \quad n > 1 \end{cases}$$

άρα

$$p(2n) = 1 + n\log n$$

Κόστος:

$$\begin{aligned} c(2n) &= p(2n) \cdot t(2n) \\ &= O(n\log^2 n) \end{aligned}$$

Το δίκτυο δεν είναι βέλτιστου κόστους καθώς

$$\underbrace{O(n)}_{\text{σειριακό κόστος}} < O(n\log^2 n)$$

3.2 Παράλληλη συγχώνευση

Υποθέτουμε ότι έχουμε στη διάθεσή μας $P_1, P_2, \cdots, P_N$ επεξεργαστές, $N \leqslant r \leqslant s$ και στη χειρότερη περίπτωση $r = s = n$. Επίσης κάθε επεξεργαστής είναι σε θέση να εκτελεί τις εξής ακολουθιακές διαδικασίες:

1. Τη διαδικασία SEQUENTIAL MERGE
2. Τη διαδικασία BINARY SEARCH

Η εισαγωγή της παραλληλίας στο πρόβλημα της συγχώνευσης βασίζεται στη μέθοδο του χωρισμού (partitioning). Δίνονται οι δύο ταξινομημένες ακολουθίες:
$A = \{\alpha_1, \alpha_2, \cdots, \alpha_r\}$ και $B = \{b_1, b_2, \cdots, b_s\}$
οι οποίες χωρίζονται σε ένα πλήθος ζευγών από υποακολουθίες έτσι ώστε να είναι δυνατός ο σχηματισμός της τελικής ταξινομημένης ακολουθίας από την ταυτόχρονη συγχώνευση των ζευγών υποακολουθιών. Χωρισμός

$$\begin{aligned} &\alpha_1 \cdots \overbrace{\alpha_{\lceil r/N \rceil}}^{\alpha'_1} | \alpha_{\lceil r/N \rceil + 1} \cdots \overbrace{\alpha_{2\lceil r/N \rceil}}^{\alpha'_2} | \alpha_{2\lceil r/N \rceil + 1} \cdots | \cdots | \alpha_{i\lceil r/N \rceil + 1} \cdots \\ &\alpha_{(i+1)\lceil r/N \rceil} | \cdots b_1 \cdots \underbrace{b_{\lceil s/N \rceil}}_{b'_1} | b_{\lceil s/N \rceil + 1} \cdots \underbrace{b_{2\lceil s/N \rceil}}_{b'_2} | \cdots | b_{i\lceil s/N \rceil + 1} \cdots \\ &\qquad b_{(i+1)\lceil s/N \rceil} | \cdots \end{aligned}$$

Εφαρμογή της BINARY SEARCH για την εύρεση του βαθμού του b_j στην A

Συγκρίνεται το b_j με το μεσαίο στοιχείο της A. Ανάλογα με το αποτέλεσμα της σύγκρισης, η αναζήτηση περιορίζεται στο πρώτο ή δεύτερο μισό τμήμα

της Α. Η εργασία αυτή συνεχίζεται (επαναλαμβάνεται) μέχρις ότου το b_j απομονωθεί μεταξύ δύο διαδοχικών στοιχείων της Α, δηλαδή όταν

$$\alpha_i < b_j < \alpha_{i+1}$$

οπότε βαθμός $(b_j : A) = i$. Στο σημείο αυτό κάνουμε χρήση της υπόθεσης ότι τα στοιχεία των Α και Β είναι διάφορα μεταξύ τους. Το πρόβλημα της συγχώνευσης μπορεί να θεωρηθεί σαν το πρόβλημα του προσδιορισμού του βαθμού κάθε στοιχείου x από την Α ή την Β στην ακολουθία $A \cup B$. Αν βαθμός $(x : A \cup B) = i$ τότε $c_i = x$, όπου c_i είναι το i -ιοστό στοιχείο της τελικής ταξινομημένης ακολουθίας. Επειδή βαθμός $(x : A \cup B) =$ βαθμός $(x : A) +$ βαθμός $(x : B)$, το πρόβλημα της συγχώνευσης επιλύεται με τον προσδιορισμό των:

$$\text{βαθμός}\,(A : B) \quad \text{και βαθμός}\,(B : A)$$

όπου

$$\text{βαθμός}\,(Y : X) = (r_1, r_2, \cdots, r_s)$$

με

$$r_i = \text{βαθμός}\,(y_i : X), \quad Y = \{y_1, y_2, \cdots, y_s\}$$

Παρατηρήσεις

1. Αν $\alpha_i = b_j$ τότε αυθαίρετα αποφασίζεται ότι το α_i είναι μικρότερο.
2. Η πράξη του ταυτόχρονου διαβάσματος εκτελείται κατά τη διάρκεια της BINARY SEARCH. Σε κάθε τέτοια χρονική στιγμή ορισμένοι επεξεργαστές εκτελούν μια δυαδική αναζήτηση στην ίδια ακολουθία.

Διαδικασία 11: BINARY SEARCH (S,x,k)

Βήμα 1
(1.1) $i \leftarrow 1$
(1.2) $h \leftarrow n$
(1.3) $k \leftarrow 0$

Βήμα 2
όσο $i \leqslant h$ **κάνε**
  (2.1) $m \leftarrow \lfloor (i+h)/2 \rfloor$
  (2.2) **αν** $x = s_m$ **τότε**
    (i) $k \leftarrow m$
    (ii) $i \leftarrow h+1$
  αλλιώς
    αν $x < s_m$ **τότε**
      $h \leftarrow m-1$
    αλλιώς
      $i \leftarrow m+1$
    τέλος
  τέλος
τέλος

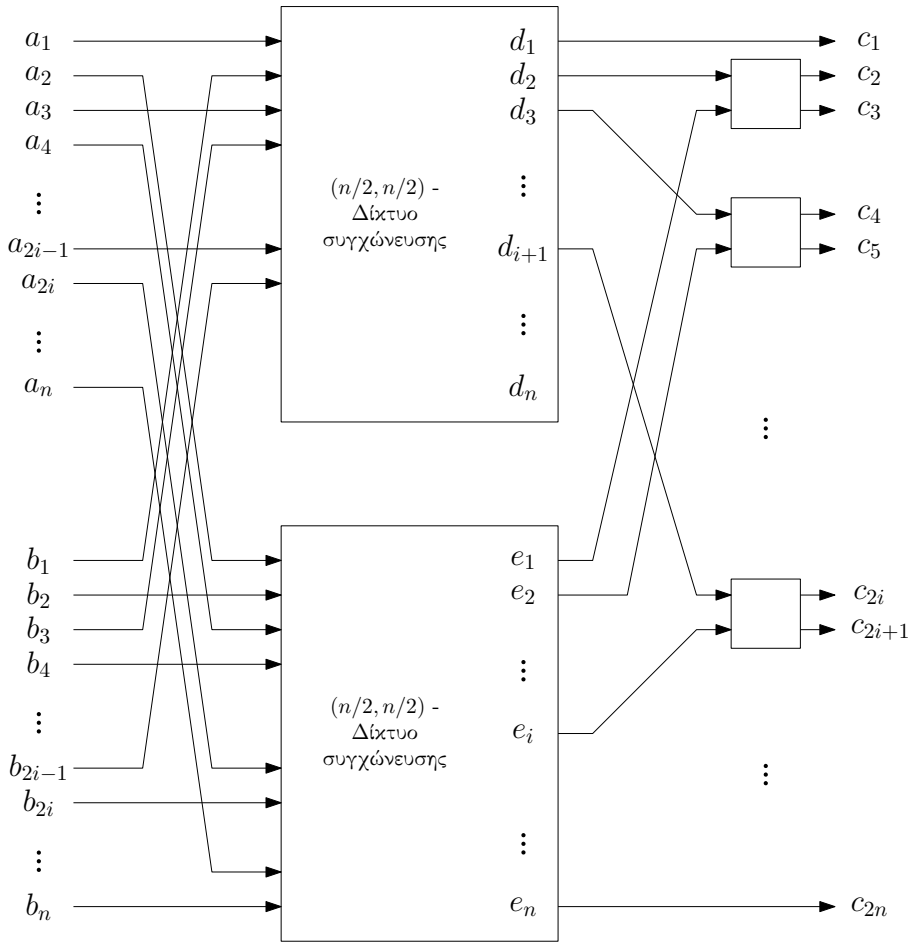
Πολυπλοκότητα: $O(\log n)$

Διαδικασία 12: SEQUENTIAL MERGE (A,B,C)

Βήμα 1
(1.1) $i \leftarrow 1$
(1.2) $j \leftarrow 1, \quad a_{r+1} \leftarrow \infty, \quad b_{s+1} \leftarrow \infty$

Βήμα 2
για $k = 1$ **έως** $r+s$ **κάνε**
  αν $a_i < b_j$ **τότε**
    (i) $c_k \leftarrow a_i$
    (ii) $i \leftarrow i+1$
  αλλιώς
    (i) $c_k \leftarrow b_j$
    (ii) $j \leftarrow j+1$
  τέλος
τέλος

$O(n) = \Omega(n) =$ κάτω όριο συγχώνευσης
Η SEQUENTIAL MERGE είναι βέλτιστη!



Σχήμα 3.4: Συγχώνευση odd-even

3.3 Συγχώνευση CREW SM SIMD

Διαδικασία 13: CREW MERGE (A, B, C)

Βήμα 1

{Επίλεξε $N-1$ στοιχεία της A που υποδιαιρούν την ακολουθία αυτή σε N υποακολουθίες του ιδίου περίπου μεγέθους. Ονόμασε την υποακολουθία που δημιουργήθηκε από αυτά τα $N-1$ στοιχεία A'. Δημιούργησε μια υποακολουθία B' από $N-1$ στοιχεία της B με τον ίδιο τρόπο. Η εκτέλεση του βήματος αυτού είναι η ακόλουθη:}

για $i = 1$ **έως** $N-1$ **κάνε παράλληλα**
  Ο επεξεργαστής P_i υπολογίζει το a'_i και b'_i από
  (1.1) $a'_i \leftarrow a_{i\lceil r/N \rceil}$
  (1.2) $b'_i \leftarrow b_{i\lceil s/N \rceil}$
τέλος

Βήμα 2

{Συγχώνευσε τις A' και B' σε μια ακολουθία τριάδων $V = \{v_1, v_2, \cdots, v_{2N-2}\}$, όπου κάθε τριάδα αποτελείται από ένα στοιχείο της A' ή B', ακολουθούμενο από τη θέση του στην A' ή B', ακολουθούμενο από το όνομα της αρχικής ακολουθίας, δηλαδή, της A ή B. Αυτό γίνεται με τον ακόλουθο τρόπο:}

(2.1) **για** $i = 1$ **έως** $N-1$ **κάνε παράλληλα**
  (i) Ο επεξεργαστής P_i χρησιμοποιεί την BINARY SEARCH στη B' για να βρεί το μικρότερο j έτσι ώστε $a'_i < b'_j$
  (ii) **αν** *υπάρχει* j **τότε**
    $v_{i+j-1} \leftarrow (a'_i, i, A)$
  αλλιώς
    $v_{i+N-1} \leftarrow (a'_i, i, A)$
  τέλος
τέλος
(2.2) **για** $i = 1$ **έως** $N-1$ **κάνε παράλληλα**
  (i) Ο επεξεργαστής P_i χρησιμοποιεί την BINARY SEARCH στην A' για να βρεί το μικρότερο j έτσι ώστε $b'_i < a'_j$
  (ii) **αν** *υπάρχει* j **τότε**
    $v_{i+j-1} \leftarrow (b'_i, i, B)$
  αλλιώς
    $v_{i+N-1} \leftarrow (b'_i, i, B)$
  τέλος
τέλος

Βήμα 3
{Κάθε επεξεργαστής συγχωνεύει και εισάγει στην C τα στοιχεία δύο υποακολουθιών, ένα από την A και ένα από την B. Οι δείκτες των δύο στοιχείων (ένα από την A και ένα από την B) από όπου κάθε επεξεργαστής πρόκειται να αρχίσει τη συγχώνευση, υπολογίζονται πρώτα και αποθηκεύονται σε έναν πίνακα Q από ταξινομημένα ζεύγη. Το βήμα αυτό εκτελείται ως εξής:}

(3.1) $Q(1) \leftarrow (1,1)$
(3.2) **για** $i = 2$ **έως** N **κάνε παράλληλα**
  αν $\nu_{2i-2} = (a'_k, k, A)$ **τότε**
    ο επεξεργαστής P_i
    (i) χρησιμοποιεί την BINARY SEARCH στο B για να βρεί το μικρότερο j τέτοιο ώστε $b_j > a'_k$
    (ii) $Q(i) \leftarrow (k\lceil r/N \rceil, j)$
  αλλιώς
    ο επεξεργαστής P_i
    (i) χρησιμοποιεί την BINARY SEARCH στο A για να βρεί το μικρότερο j τέτοιο ώστε $a_j > b'_k$
    (ii) $Q(i) \leftarrow (j, k\lceil s/N \rceil)$
  τέλος
τέλος
(3.3) **για** $i = 1$ **έως** N **κάνε παράλληλα**
  Ο επεξεργαστής P_i χρησιμοποιεί την SEQUENTIAL MERGE και τον $Q(i) = (x, y)$ για να συγχωνεύσει δύο υποακολουθίες, η μία ξεκινώντας από το a_x και η άλλη από το b_y και τοποθετεί τα αποτελέσματα της συγχώνευσης στον πίνακα C αρχίζοντας στη θέση $x + y - 1$. Η συγχώνευση συνεχίζεται μέχρις ότου
  (i) βρεθεί ένα στοιχείο μεγαλύτερο ή ίσο με το πρώτο στοιχείο του ν_{2i}, σε κάθε μία από τις A και B (όταν $i \leqslant N-1$)
  (ii) δεν υπάρχει κανένα στοιχείο στην A ή B (όταν $i = N$)
τέλος

Ανάλυση

Βήμα 1: Κάθε επεξεργαστής υπολογίζει δύο δείκτες. Συνεπώς το βήμα αυτό απαιτεί σταθερό χρόνο $O(1)$.

Βήμα 2: Το βήμα αυτό αποτελείται από δύο εφαρμογές της BINARY SEARCH σε μία ακολουθία με $N-1$ στοιχεία. Κάθε εφαρμογή της BINARY

SEARCH ακολουθείται από μία εντολή καταχώρησης. Συνεπώς απαιτείται $O(\log n)$ χρόνος.

Βήμα 3: Το βήμα (3.1) αποτελείται από μία καταχώρηση σταθερού χρόνου, και το βήμα (3.2) απαιτεί το πολύ $O(\log s)$ χρόνο (με $(r \leqslant s)$) . Για την ανάλυση του βήματος (3.3), παρατηρούμε ότι η V περιέχει $2N-2$ στοιχεία τα οποία διαιρούν τη C σε $2N-1$ υποακολουθίες με μέγιστο μέγεθος ίσο με $\lceil r/N \rceil + \lceil s/N \rceil$. Το μέγιστο μέγεθος αυτό προκύπτει αν, για παράδειγμα, ένα στοιχείο a' της A' είναι ίσο με ένα στοιχείο b' της B'. Τότε τα $\lceil r/N \rceil$ στοιχεία μικρότερα ή ίσα του a'_i (και μεγαλύτερα ή ίσα του a'_{i-1}) είναι επίσης μικρότερα ή ίσα του b'_j και όμοια τα $\lceil s/N \rceil$ στοιχεία μικρότερα ή ίσα του b'_j (και μεγαλύτερα του b'_{j-1}) είναι επίσης μικρότερα ή ίσα του a'_i. Στο βήμα αυτό κάθε επεξεργαστής δημιουργεί δύο τέτοιες υποακολουθίες της C των οποίων το ολικό μέγεθος δεν είναι μεγαλύτερο από $2(\lceil r/N \rceil + \lceil s/N \rceil)$, εκτός του P_N, ο οποίος δημιουργεί μόνο μία υποακολουθία της C. Συνεπώς η SEQUENTIAL MERGE απαιτεί το πολύ $O((r+s)/N)$ χρόνο.
Στην χειρότερη περίπτωση $r = s = n$ και επειδή $n \geqslant N$, ο χρόνος εκτέλεσης του αλγορίθμου ουσιαστικά είναι ο χρόνος εκτέλεσης του βήματος 3. Συνεπώς

$$t(2n) = O((n/N) + \log N)$$

Επειδή $p(2n) = N$, $c(2n) = p(2n) \cdot t(2n) = O(n + N\log n)$. Ο αλγόριθμος έχει βέλτιστο κόστος αν και μόνο αν $N \leqslant n/\log n$.

Παράδειγμα

Έστω $N = 4$, $A = \{2, 3, \textcircled{4}, 6, 11, \textcircled{12}, 13, 15, \textcircled{16}, 20, 22, 24\}$
$B = \{1, 5, \textcircled{7}, 8, 9, \textcircled{10}, 14, 17, \textcircled{18}, 19, 21, 23\}$, δηλαδή $r = s = 12$

Βήμα 1: $A' = \{4, 12, 16\}$, $B' = \{7, 10, 18\}$ με $\lceil r/N \rceil = \lceil s/N \rceil = 3$

Βήμα 2: Συγχώνευση των A', B' και δημιουργία της
$V = \{(4, 1, A), (7, 1, B), (10, 2, B), (12, 2, A), (16, 3, A), (18, 3, B)\}$

Βήμα 3: $Q(1) = (1, 1), Q(2) = (5, 3), Q(3) = (6, 7), Q(4) = (10, 9)$
Βήμα 3.3: Ο επεξεργαστής P_1 αρχίζει από τα στοιχεία $a_1 = 2$ και $b_1 = 1$ και συγχωνεύει όλα τα στοιχεία των A και B μικρότερα του 7, δημιουργώντας την υποακολουθία {1,2,3,4,5,6} της C. Ομοίως ο επεξεργαστής P_2 αρχίζει από τα στοιχεία $a_5 = 11$ και $b_3 = 7$ και συγχωνεύει όλα τα στοιχεία μικρότερα του 12, δημιουργώντας την {7,8,9,10,11}. Ο επεξεργα-

στής P_3 αρχίζει από τα στοιχεία $a_6 = 12$ και $b_7 = 14$ και δημιουργεί την $\{12,13,14,15,16,17\}$. Τέλος, ο P_4 αρχίζει από τα $a_{10} = 20$ και $b_9 = 18$ και δημιουργεί την $\{18, 19, 20, 21, 22, 23, 24\}$. Η τελική ακολουθία C είναι συνεπώς η

$$C = \{1, 2, 3, ④, 5, 6, ⑦, 8, 9, ⑩, 11, ⑫, 13, 14, 15, ⑯, 17, ⑱, 19, 20, 21, 22, 23, 24\}$$

3.4 Συγχώνευση στο μοντέλο EREW

Σε διάφορα σημεία της διαδικασίας CREW MERGE εκτελούνται εργασίες ταυτόχρονης ανάγνωσης. Η εν λόγω διαδικασία μπορεί να προσαρμοστεί ώστε να εκτελείται σε έναν υπολογιστή EREW SM SIMD N-επεξεργαστών, ο οποίος απαγορεύει εξ ορισμού την ανάγνωση μίας τοποθεσίας μνήμης από περισσότερους του ενός επεξεργαστές.
Ένας αλγόριθμος σχεδιασμένος να εκτελείται σε υπολογιστή CREW SM SIMD απαιτεί συνολικά M τοποθεσίες κοινόχρηστης μνήμης. Για προσομοίωση αυτού του αλγορίθμου στο μοντέλο EREW με N επεξεργαστές, όπου

$$N = 2^q, \quad q \geqslant 1$$

το μέγεθος της μνήμης αυξάνεται:

$$M \to M(2N - 1)$$

Κάθε τοποθεσία μνήμης θεωρείται ως ρίζα ενός δυαδικού δέντρου με N φύλλα. Ένα τέτοιο δέντρο έχει $q + 1$ επίπεδα και $2N - 1$ κόμβους.
$d(i) + (N - 1) + (i - 1) =$ φύλλο με ρίζα στην $d(i)$.

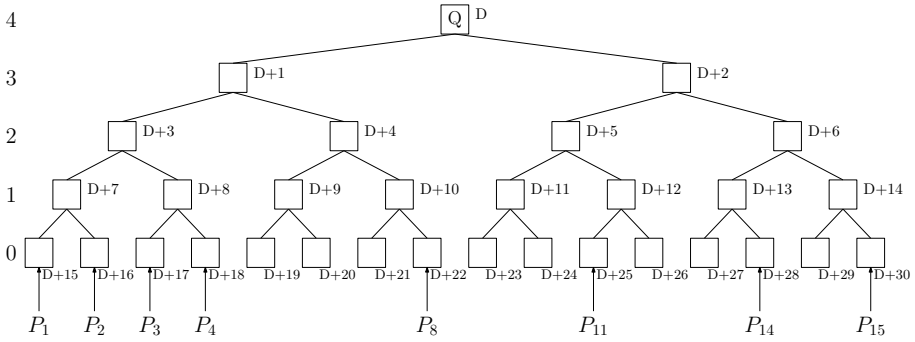
Έστω ότι ο επεξεργαστής P_i διαβάζει από μία θέση μνήμης $d(i)$. Το αίτημά του τοποθετείται στη θέση $d(i) + (N - 1) + (i - 1)$, ένα φύλλο του δέντρου με ρίζα το $d(i)$. Αυτό επιτυγχάνεται με αρχικοποίηση δύο μεταβλητών, τοπικών στο P_i:

1. level(i), που αποθηκεύει το τρέχον επίπεδο του δέντρου που έχει φτάσει το αίτημα του P_i, αρχικοποιημένο στο μηδέν, και

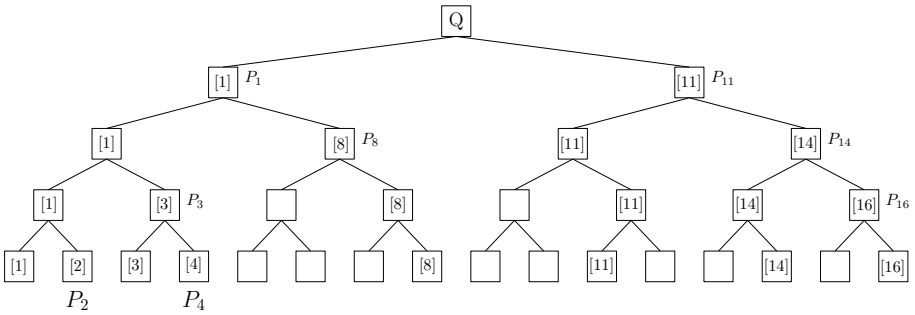
2. loc(i), που αποθηκεύει τον τρέχοντα κόμβο του δέντρου που έχει φτάσει το αίτημα του P_i, αρχικοποιημένο στο $(N - 1) + (i - 1)$.

Η εξομοίωση έχει δύο στάδια: ανάβαση και κατάβαση. Κατά την ανάβαση, ένας επεξεργαστής P_i που καταλαμβάνει ένα αριστερό παιδί λαμβάνει

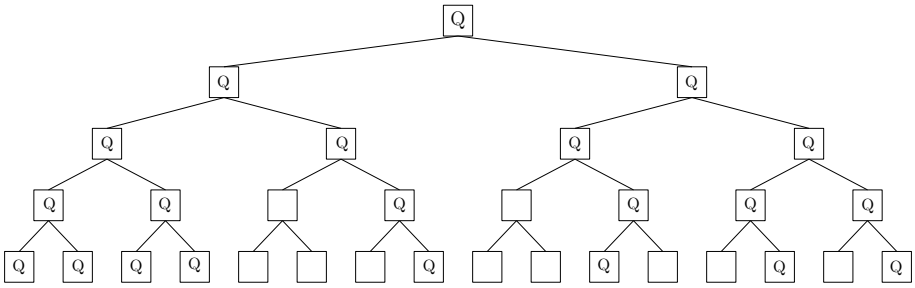
προτεραιότητα να προωθήσει το αίτημά του ένα επίπεδο πιο πάνω στο δέντρο.



Σχήμα 3.5: Οργάνωση μνήμης για πολλαπλή αναμετάδοση



Σχήμα 3.6: Περιεχόμενα μνήμης μετά το βήμα 2 της MULTIPLE BROADCAST



Σχήμα 3.7: Περιεχόμενα μνήμης στο τέλος της MULTIPLE BROADCAST

Αυτό επιτυγχάνεται με σήμανση της τοποθεσίας-γονέα, έστω $[i]$. Έπειτα ενημερώνεται το επίπεδο και η θέση του. Πιθανό αίτημα στο δεξιό παιδί παγώνει για το υπόλοιπο της διαδικασίας. Αλλιώς, ένας επεξεργαστής που

βρίσκεται στο δεξιό παιδί μπορεί πλέον να διεκδικήσει την τοποθεσία-γονέα. Τα παραπάνω συνεχίζονται μέχρι το πολύ δύο επεξεργαστές να φτάσουν το επίπεδο $(\log N) - 1$.

Διαδικασία 14: MULTIPLE BROADCAST $(d(1), d(2), \cdots, d(N))$

Βήμα 1
για $i = 1$ **έως** N **κάνε παράλληλα**
  Ο P_i δίνει αρχικές τιμές στο επίπεδο(i) και loc(i)
  (1.1) επίπεδο (i) $\leftarrow 0$
  (1.2) $loc(i) \leftarrow N + i - 2$
  (1.3) αποθήκευσε [i] στην τοποθεσία $d(i) + loc(i)$
τέλος

{Ανοδική πορεία στο δέντρο}
Βήμα 2
για $\nu = 0$ **έως** $(\log N) - 2$ **κάνε**
  (2.1) **για** $i = 1$ **έως** N **κάνε παράλληλα**
    {Ο P_i σε ένα αριστερό παιδί προχωράει προς τα πάνω στο δέντρο του}
    (2.1.1) $x \leftarrow \lfloor (loc(i) - 1)/2 \rfloor$
    (2.1.2) **αν** $loc(i)$ *είναι περιττό και επίπεδο(i)*$=\nu$ **τότε**
      (i) $loc(i) \leftarrow x$
      (ii) αποθήκευσε [i] στη θέση $d(i) + loc(i)$
      (iii) επίπεδο(i) $\leftarrow$ επίπεδο(i) $+1$
    τέλος
  τέλος
  (2.2) **για** $i = 1$ **έως** N **κάνε παράλληλα**
    {Ο P_i σε ένα δεξί παιδί προχωράει πάνω στο δέντρο του, αν είναι δυνατόν}
    αν $d(i) + x$ *δεν περιέχει ήδη ένα σημάδι* [j] *για κάποιο* $1 \leqslant j \leqslant N$ **τότε**
      (i) $loc(i) \leftarrow x$
      (ii) αποθήκευσε [i] στη θέση $d(i) + loc(i)$
      (iii) επίπεδο(i) $\leftarrow$ επίπεδο(i) $+1$
    τέλος
  τέλος
τέλος

{Καθοδική πορεία στο δέντρο}
Βήμα 3
για $\nu = (\log N) - 1$ **έως** 0 **κάνε**
  (3.1) **για** $i = 1$ **έως** N **κάνε παράλληλα**
    {ο P_i σε ένα αριστερό παιδί διαβάζει από τον πατέρα του και κατεβαίνει το δέντρο}
    (3.1.1) $x \leftarrow \lfloor (loc(i) - 1)/2 \rfloor$
    (3.1.2) $y \leftarrow (2 \times loc(i)) + 1$
    (3.1.3) **αν** $loc(i)$ *είναι περιττό και επίπεδο(i)*$= \nu$ **τότε**
      (i) διάβασε τα περιεχόμενα του $d(i) + x$
      (ii) γράψε τα περιεχόμενα του $d(i) + x$ στη θέση $d(i) + loc(i)$
      (iii) επίπεδο(i) $\leftarrow$ επίπεδο(i)-1
      (iv) **αν** *η θέση* $d(i) + y$ *περιέχει* [i] **τότε**
        $loc(i) \leftarrow y$
      αλλιώς
        $loc(i) \leftarrow y + 1$
      τέλος
    τέλος
  τέλος
  (3.2) **για** $i = 1$ **έως** N **κάνε παράλληλα**
    {ο P_i σε ένα δεξί παιδί διαβάζει από τον πατέρα του και κατεβαίνει το δέντρο}
    αν $loc(i)$ *είναι άρτιο και επίπεδο(i)*$= \nu$ **τότε**
      (i) διάβασε τα περιεχόμενα του $d(i) + x$
      (ii) γράψε τα περιεχόμενα του $d(i) + x$ στη θέση $d(i) + loc(i)$
      (iii) επίπεδο(i) $\leftarrow$ επίπεδο(i)-1
      (iv) **αν** *η θέση* $d(i) + y$ *περιέχει* [i] **τότε**
        $loc(i) \leftarrow y$
      αλλιώς
        $loc(i) \leftarrow y + 1$
      τέλος
    τέλος
  τέλος
τέλος

Συνολικός χρόνος $O(\log N)$

Κάθε ένα από τα βήματα 2 και 3 της MULTIPLE BROADCAST απαιτεί χρόνο $O(\log N)$.

Πολυπλοκότητα προσαρμογής της CREW MERGE για το EREW μοντέλο

$$t(2n) = \underbrace{O(\log N)}_{\substack{\text{MULTIPLE}\\ \text{BROADCAST}}} \cdot \underbrace{O(n/N + \log n)}_{\text{CREW MERGE}}$$

$$t(2n) = O((n/N)\log n + \log^2 n)$$

$$(\log N \approx \log n)$$

$$c(2n) = t(2n) \cdot p(2n) = O(n\log n + N\log^2 n)$$

Το κόστος δεν είναι βέλτιστο. Επίσης η CREW MERGE χρησιμοποιεί $O(n)$ τοποθεσίες κοινής μνήμης και οι απαιτήσεις της προσαρμογής της στο EREW από άποψη μνήμης είναι $O(Nn)$.

3.5 Ένας καλύτερος αλγόριθμος για το EREW μοντέλο

Σε αυτή την ενότητα περιγράφεται ένας αποτελεσματικότερος αλγόριθμος για συγχώνευση στο EREW SM SIMD μοντέλο. Ο αλγόριθμος συγχωνεύει δύο ακολουθίες

$$A = \{\alpha_1, \alpha_2, \cdots, \alpha_r\}, \qquad B = \{b_1, b_2, \cdots, b_s\}$$

σε μία ακολουθία

$$C = \{c_1, c_2, \cdots, c_{r+s}\}$$

χρησιμοποιώντας N επεξεργαστές

$$P_1, P_2, \cdots, P_N, \quad 1 \leqslant N \leqslant r + s$$

Εύρεση του μέσου δύο ταξινομημένων ακολουθιών

Δεδομένων δύο ταξινομημένων ακολουθιών $A = \{\alpha_1, \alpha_2, \cdots, \alpha_r\}$ και $B = \{b_1, b_2, \cdots, b_s\}$ όπου $r, s \geqslant 1$ συμβολίζουμε με $A \cdot B$ την ακολουθία μήκους $m = r + s$ που προκύπτει από τη συγχώνευση των A και B. Ζητείται να βρεθεί το μέσο, δηλαδή το $\lceil m/2 \rceil$ στοιχείο της $A \cdot B$ χωρίς να σχηματιστεί (βρεθεί) η $A \cdot B$. Ο αλγόριθμος επιστρέφει ένα ζεύγος (α_x, b_y) που ικανοποιεί τις ιδιότητες:

1. Ένα από τα a_x ή b_y είναι το μέσον της $A \cdot B$ δηλαδή το a_x ή το b_y είναι μεγαλύτερο από $\lceil m/2 \rceil - 1$ στοιχεία και μικρότερο από $\lfloor m/2 \rfloor$ στοιχεία.

2. Αν το a_x είναι το μέσο τότε το b_y είτε είναι

   (i) το μεγαλύτερο στοιχείο του B μικρότερο ή ίσο του a_x ή

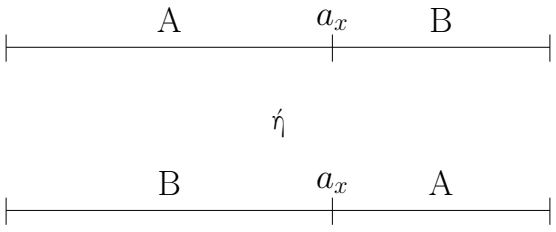
   (ii) το μικρότερο στοιχείο του B μεγαλύτερο ή ίσο του a_x

   Διαφορετικά αν το b_y είναι το μέσον, τότε a_x είναι είτε:

   (i) το μεγαλύτερο στοιχείο της A και μικρότερο ή ίσο του b_y ή

   (ii) το μικρότερο στοιχείο της A μεγαλύτερο ή ίσο του b_y.

3. Αν περισσότερα του ενός ζεύγη ικανοποιούν τα 1 και 2, τότε ο αλγόριθμος επιστρέφει το ζεύγος για το οποίο το $x + y$ είναι το μικρότερο.



Σχήμα 3.8: Εύρεση μέσου

Το (a_x, b_y) λέγεται μεσαίο ζεύγος της $A \cdot B$ συνεπώς x και y είναι οι δείκτες του μεσαίου ζεύγους. Ας σημειωθεί ότι a_x είναι το μέσον της $A \cdot B$ αν

(i) $a_x > b_y$ και $x + y - 1 = \lceil m/2 \rceil - 1$ ή

(ii) $a_x < b_y$ και $m - (x + y - 1) = \lfloor m/2 \rfloor$

Διαφορετικά το b_y είναι το μέσον της $A \cdot B$.

Ας σημειωθεί ότι η TWO-SEQUENCE MEDIAN επιστρέφει τους δείκτες του ζεύγους μέσων (a, b_y) και όχι το ίδιο το ζεύγος.

Παράδειγμα

$$A = \{10, 11, 12, 13, 14, 15, 16, 17, 18\}$$

$$B = \{3, 4, 5, 6, 7, 8, 19, 20, 21, 22\}$$

Διαδικασία 15: TWO-SEQUENCE MEDIAN (A, B, x, y)

Βήμα 1
(1.1) $\text{χαμηλό}_B \leftarrow 1$
(1.2) $\text{χαμηλό}_B \leftarrow 1$
(1.3) $\text{ψηλό}_A \leftarrow r$
(1.4) $\text{ψηλό}_B \leftarrow s$
(1.5) $n_A \leftarrow r$
(1.6) $n_B \leftarrow s$

Βήμα 2 όσο $n_A > 1$ ***και*** $n_B > 1$ **κάνε**
(2.1) $u \leftarrow \text{χαμηλό}_A + \lceil(\text{ψηλό}_A - \text{χαμηλό}_A - 1)/2\rceil$
(2.2) $v \leftarrow \text{χαμηλό}_B + \lceil(\text{ψηλό}_B - \text{χαμηλό}_B - 1)/2\rceil$
(2.3) $w \leftarrow \min(\lfloor n_A/2\rfloor, \lfloor n_B/2\rfloor)$
(2.4) $n_A \leftarrow n_A - w$
(2.5) $n_B \leftarrow n_B - w$
(2.6) **αν** $a_u \geqslant b_v$ **τότε**
(i) $\text{ψηλό}_A \leftarrow \text{ψηλό}_A - w$
(ii) $\text{χαμηλό}_B \leftarrow \text{χαμηλό}_B + w$
αλλιώς
(i) $\text{χαμηλό}_A \leftarrow \text{χαμηλό}_A + w$
(ii) $\text{ψηλό}_B \leftarrow \text{ψηλό}_B - w$
τέλος
τέλος

Βήμα 3 Επίστρεψε σαν x και y τους δείκτες του ζεύγους από $\{a_{u-1}, a_u, a_{u+1}\} \times \{b_{v-1}, b_v, b_{v+1}\}$ που ικανοποιούν τις ιδιότητες 1-3 ενός μεσαίου ζεύγους.

$$low_A = low_B = 1 \quad high_A = n_A = 9 \quad high_B = n_B = 10$$

Βήμα 2

1η επανάληψη

$$u = v = 5, \quad w = min(4, 5) = 4, \quad n_A = 5, \quad n_B = 6$$

Επειδή $\alpha_5 > b_5$, $\quad high_A = low_B = 5$

2η επανάληψη

$$u = 3, v = 7, w = min(2, 3) = 2, n_A = 3, n_B = 4$$

Επειδή $\alpha_3 < b_7$, $\quad low_A = 3$ και $high_B = 8$

3η επανάληψη

$$u = 4, v = 6, w = min(1, 2) = 1, \quad n_A = 2, n_B = 3$$

Επειδή $\alpha_4 > b_6$, $\quad high_A = 4 \quad low_B = 6$

4η επανάληψη

$$u = 3, v = 7, w = min(1, 1) = 1, \quad n_A = 1, n_B = 2$$

Επειδή $\alpha_3 < b_7$, $\quad low_A = 4$ και $high_B = 7$

Βήμα 3

Έχουμε 9 ζεύγη $\{11, 12, 13\} \times \{8, 19, 20\}$
Αυτά που ικανοποιούν τις δύο πρώτες ιδιότητες είναι τα εξής δύο:

$$(\alpha_4, b_6) = (13, 8) \qquad \text{και} \qquad (\alpha_4, b_7) = (13, 19)$$

οπότε επιστρέφεται το $(4, 6)$.

3.6 Συγχώνευση στο μοντέλο EREW

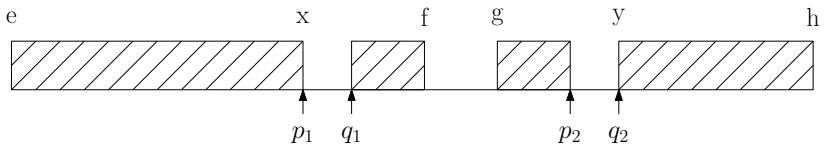
Δεδομένων δύο ταξινομημένων ακολουθιών $A = \{\alpha_1, \alpha_2, \cdots, \alpha_r\}$ και $B = \{b_1, b_2, \cdots, b_s\}$ υποθέτουμε την ύπαρξη N επεξεργαστών $P_1, P_2, \cdots, P_N$, όπου N είναι μια δύναμη του 2 και $1 \leqslant N \leqslant r + s$. Ο αλγόριθμος που θα αναπτυχθεί στη συνέχεια συγχωνεύει τις A και B σε μία ταξινομημένη ακολουθία $C = \{c_1, c_2, \cdots, c_{r+s}\}$ σε δυο φάσεις ως εξής:
Φάση 1: Κάθε μια από τις δύο ακολουθίες A και B χωρίζονται σε N (πιθανές κενές) υποακολουθίες $A_1, A_2, \cdots, A_N$ και $B_1, B_2, \cdots, B_N$ τέτοιες ώστε

(i) $|A_i| + |B_i| = (r + s)/N$ για $1 \leqslant i \leqslant N$

(ii) όλα τα στοιχεία στην $A_i \cdot B_i$ είναι μικρότερα ή ίσα από όλα τα στοιχεία στην $A_{i+1} \cdot B_{i+1}$ για $1 \leqslant i \leqslant N$

(I) Περίπτωση όπου

- α_x είναι το μεσαίο στοιχείο
- $\alpha_x \leqslant b_y$



Σχήμα 3.9: $\alpha_x \leqslant b_y$

Παρατήρηση

- Οι υποακολουθίες με τα μικρότερα στοιχεία στέλνονται σε ένα επεξεργαστή P_{2i-1}, ενώ οι άλλες με τα μεγαλύτερα στοιχεία στέλνονται στον επεξεργαστή P_{2i}
- Το b_y συμπεριλαμβάνεται στην υπακολουθία με τα μεγαλύτερα στοιχεία

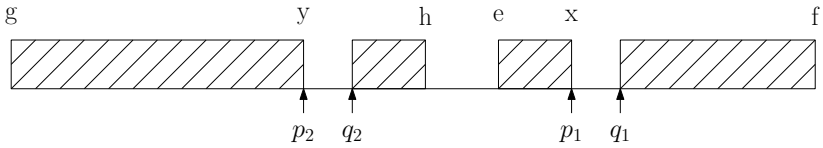
(II) Περίπτωση όπου

- α_x είναι το μεσαίο στοιχείο
- $b_y \leqslant \alpha_x$

Διαδικασία 16: EREW MERGE (A, B, C)

Βήμα 1
(1.1) Ο επεξεργαστής P_1 λαμβάνει την τετράδα $(1, r, 1, s)$
(1.2) **για** $j = 1$ **έως** $\log N$ **κάνε**
  για $i = 1$ **έως** 2^{j-1} **κάνε παράλληλα**
    Ο επεξεργαστής P_i έχοντας λάβει την τετράδα (e, f, g, h)
    (1.2.1) {Βρίσκει το μεσαίο ζεύγος των δύο ακολουθιών TWO-SEQUENCE MEDIAN $(A[e, f], B[g, h], x, y)$}
    (1.2.2) {Υπολογίζει τέσσερις δείκτες p_1, p_2, q_1 και q_2 ως εξής}
    αν α_x *είναι το μεσαίο* **τότε**
      (i) $p_1 \leftarrow x$
      (ii) $q_1 \leftarrow x + 1$
      (iii) **αν** $b_y \leqslant \alpha_x$ **τότε**
        (a) $p_2 \leftarrow y$
        (b) $q_2 \leftarrow y + 1$
      αλλιώς
        (a) $p_2 \leftarrow y - 1$
        (b) $q_2 \leftarrow y$
      τέλος
    αλλιώς
      (i) $p_2 \leftarrow y$
      (ii) $q_2 \leftarrow y + 1$
      (iii) **αν** $\alpha_x \leqslant b_y$ **τότε**
        (a) $p_1 \leftarrow x$
        (b) $q_1 \leftarrow x + 1$
      αλλιώς
        (a) $p_1 \leftarrow x - 1$
        (b) $q_1 \leftarrow x$
      τέλος
    τέλος
    (1.2.3) Μεταδίδει τις τετράδες (e, p_1, g, p_2) στον P_{2i-1}
    (1.2.4) Μεταδίδει τις τετράδες (q_1, f, q_2, h) στον P_{2i}
  τέλος
τέλος

Βήμα 2
για $i = 1$ **έως** N **κάνε παράλληλα**
  Ο επεξεργαστής P_i έχοντας λάβει την τετράδα (a, b, c, d)
  (2.1) $w \leftarrow 1 + (i - 1)\lceil (r + s)/N \rceil$
  (2.2) $z \leftarrow \min\{i\lceil (r + s)/N \rceil, (r + s)\}$
  (2.3) SEQUENTIAL MERGE $(A[a, b], B[c, d], C[w, z])$
τέλος



Σχήμα 3.10: $b_y \leqslant \alpha_x$

(ii) Όλα τα ζεύγη A_i και $B_i, 1 \leqslant i \leqslant N$ συγχωνεύονται ταυτόχρονα και τοποθετούνται στην C.

Παράδειγμα

$$A = \{10, 11, 12, 13, 14, 15, 16, 17, 18\},$$
$$B = \{3, 4, 5, 6, 7, 8, 19, 20, 21, 22\}, N = 4$$
$$r = 9, s = 10$$

Βήμα 1.1: Ο P_1 λαμβάνει (1,9,1,10).

Βήμα 1.2: Ο P_1 προσδιορίζει το μεσαίο ζεύγος των A και B που είναι το $(4, 6)$. Διατηρεί το $(1, 4, 1, 6)$ και μεταδίδει το $(5, 9, 7, 10)$ στον P_2. Στη δεύτερη επανάληψη, ο P_1 υπολογίζει τους δείκτες του μεσαίου ζεύγους των $A[1, 4] = \{10, 11, 12, 13\}$ και $B[1, 6] = \{3, 4, 5, 6, 7, 8\}$ που είναι το (1,5). Ταυτόχρονα ο P_2 κάνει το ίδιο με τις $A[5, 9] = \{14, 15, 16, 17, 18\}$ και $B[7, 10] = \{19, 20, 21, 22\}$ και λαμβάνει το ζεύγος $(9, 7)$. Ο P_1 κρατά το $(1, 0, 1, 5)$ και μεταδίδει το $(1, 4, 6, 6)$ στον P_2. Όμοια ο P_2 μεταδίδει το $(5, 9, 7, 6)$ στον P_3 και το $(10, 9, 7, 10)$ στον P_4.

Βήμα 2: Οι P_1, P_2, P_3 και P_4 δημιουργούν ταυτόχρονα την $C[1, 19]$ ως εξής:

- Η τελευταία τετράδα που έλαβε ο P_1 είναι η $(1, 0, 1, 5)$ και ο P_1 υπολογίζει $w = 1, z = 5$ και αντιγράφει την $B[1, 5] = \{3, 4, 5, 6, 7\}$ στην $C[1, 5]$.

- Όμοια ο P_2, έχοντας λάβει τελευταία την $(1, 4, 6, 6)$, υπολογίζει $w = 6$ και $z = 10$ και συγχωνεύει τις $A[1, 4]$ και $B[6, 6]$ για το σχηματισμό της $C[6, 10] = \{8, 10, 11, 12, 13\}$.

- Ο P_3 έχοντας λάβει την τελευταία $(5, 9, 7, 6)$ υπολογίζει $w = 11$ και $z = 15$ και αντιγράφει την $A[5, 9] = \{14, 15, 16, 17, 18\}$ στην $C[11, 15]$. Τέλος ο P_4 έχοντας λάβει στο τέλος την $(10, 9, 7, 10)$ υπολογίζει $w = 16, z = 19$ και αντιγράφει την $B[7, 10] = \{19, 20, 21, 22\}$ στην

$C[16, 19]$.

Ανάλυση

Βήμα 1.1: Ο P_1 διαβάζει από τη μνήμη σε σταθερό χρόνο.

Βήμα 1.2: Κατά τη διάρκεια της $j-$ ιοστής επανάληψης κάθε επεξεργαστής υπολογίζει το μεσαίο ζεύγος από $(r+s)/2^{j-1}$ στοιχεία. Η εργασία αυτή γίνεται με την κλήση της TWO-SEQUENCE MEDIAN σε $O(\log[(r+s)/2^{j-1}])$ χρόνο, ή $O(\log(r+s))$. Τα άλλα δύο βήματα στο βήμα 1.2 απαιτούν σταθερό χρόνο. Επειδή εκτελούνται $\log N$ επαναλήψεις του βήματος 1.2 το Βήμα 1 απαιτεί χρόνο

$$O(\log N \cdot \log(r+s))$$

Στο βήμα 2 κάθε επεξεργαστής συγχωνεύει το πολύ $(r+s)/N$ στοιχεία. Η εργασία αυτή γίνεται με την SEQUENTIAL MERGE και απαιτεί $O((r+s)/N)$ χρόνο.

Συνολικά

$$O((r+s)/N) + \log N \cdot \log(r+s)$$

Στη χειρότερη περίπτωση $r = s = n$ και

$$t(2n) = O(n/N + \log^2 n)$$

με κόστος

$$c(2n) = O(n + N\log^2 n)$$

Επειδή $\Omega(n)$ είναι το κάτω φράγμα του αριθμού των πράξεων για συγχώνευση το ανωτέρω κόστος είναι βέλτιστο όταν $N \leqslant n/\log^2 n$.

Κεφάλαιο 4

ΤΑΞΙΝΟΜΗΣΗ (sorting)

Δεδομένης μιας ακολουθίας $S = \{s_1, s_2, \cdots, s_n\}$ από n στοιχεία στα οποία έχει οριστεί η γραμμική διάταξη $<$, να βρεθεί μια νέα ακολουθία $S' = \{s'_1, s'_2, \cdots, s'_n\}$ τέτοια ώστε $s'_i < s'_{i+1}$ για $i = 1, 2, \cdots, n-1$.

4.1 Ένα κάτω όριο

Η ταξινόμηση με σύγκριση θεωρητικά μπορεί να μελετηθεί με τα δέντρα απόφασης (decision trees). Κάθε μία από τις $n!$ μεταθέσεις των n στοιχείων πρέπει να εμφανίζεται σαν ένα από τα φύλλα του δέντρου απόφασης προκειμένου ο αλγόριθμος ταξινόμησης να ταξινομεί κανονικά (σωστά). Συνεπώς ο αριθμός των συγκρίσεων στη χειρότερη περίπτωση για μια ταξινόμηση με σύγκριση αντιστοιχεί με το ύψος του δέντρου απόφασης. Ένα κάτω όριο των υψών των δέντρων απόφασης είναι συνεπώς ένα κάτω όριο του χρόνου εκτέλεσης οποιουδήποτε αλγόριθμου ταξινόμησης με σύγκριση.

Θεώρημα 4.1.1. *Ένα δέντρο απόφασης, το οποίο ταξινομεί n στοιχεία έχει ύψος $\Omega(n \log n)$*

Απόδειξη. Επειδή υπάρχουν $n!$ μεταθέσεις των n στοιχείων, το δέντρο πρέπει να έχει τουλάχιστον $n!$ φύλλα. Επειδή ένα δυαδικό δέντρο ύψους h έχει το πολύ 2^h φύλλα, έχουμε

$$n! \leqslant 2^h \qquad \text{ή} \qquad h \geqslant \log(n!)$$

επειδή η $\log$ είναι μονότονα αύξουσα.
Αλλά από την προσέγγιση Stirling έχουμε

$$n! \geqslant \left(\frac{n}{e}\right)^n, e = 2.71828$$

συνεπώς

$$h \geqslant \log\left(\frac{n}{e}\right)^n = n\log n - n\log e$$

ή

$$h \geqslant \Omega(n\log n).$$

Άρα ένας αλγόριθμος ταξινόμησης απαιτεί $\Omega(n\log n)$ πολυπλοκότητα στη χειρότερη περίπτωση.
Συνεπώς για ένα παράλληλο αλγόριθμο ταξινόμησης που χρησιμοποιεί N επεξεργαστές έχουμε $\Omega((n\log n)/N), N \leqslant n\log n$. ■

4.2 Ένα Δίκτυο για Ταξινόμηση

Για ταξινόμηση μέσω δικτύου μιας ακολουθίας $S = \{s_1, s_2, \cdots, s_n\}$, όπου n είναι μια δύναμη του 2 ακολουθούμε τα εξής βήματα. Στο πρώτο βήμα χρησιμοποιούνται $n/2$ συγκριτές για τη δημιουργία $n/2$ ταξινομημένων ακολουθιών μήκους 2 η κάθε μία. Στο δεύτερο βήμα ζεύγη αυτών συγχωνεύονται σε ταξινομημένες ακολουθίες μήκους με τη χρήση μιας στήλης από (2,2) - δίκτυα συγχώνευσης. Στο τρίτο βήμα, ζεύγη ακολουθιών μήκους 4 συγχωνεύονται με τη χρήση (4,4) δικτύων συγχώνευσης σε ακολουθίες μήκους 8. Η εργασία αυτή συνεχίζεται μέχρις ότου δύο ακολουθίες μήκους $n/2$ συγχωνευθούν.

Ανάλυση

Επειδή το μέγεθος των ακολουθιών που συγχωνεύονται διπλασιάζεται σε κάθε βήμα, συνολικά απαιτούνται $\log n$ βήματα.

Χρόνος Εκτέλεσης
Αν $s(2^i)$ συμβολίζει το χρόνο συγχώνευσης δύο ακολουθιών στο $i-$ οστό βήμα με 2^{i-1} στοιχεία η κάθε μία, τότε

$$s(2) = 1, \quad i = 1$$

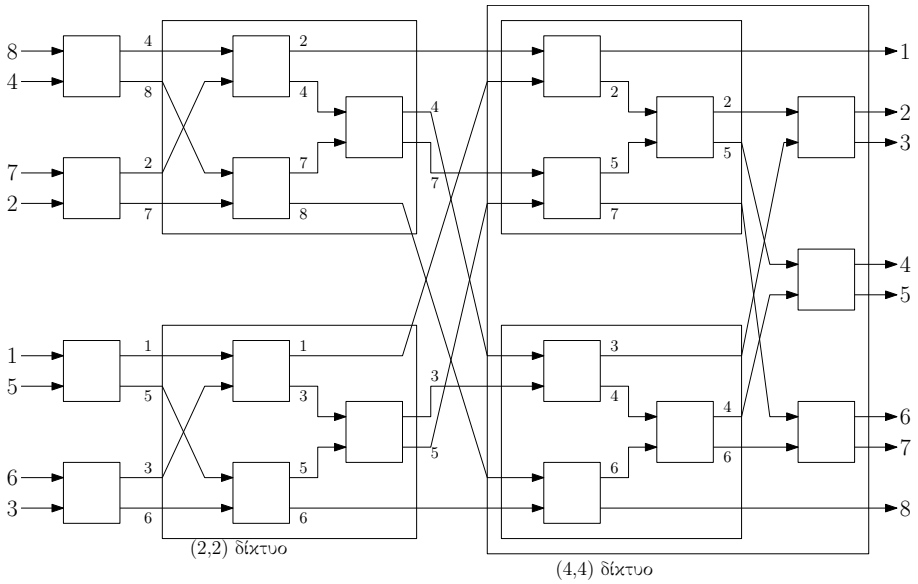
$$s(2^i) = s(2^{i-1}) + 1, \quad i > 1$$

άρα $s(2^i) = i$ και

$$t(n) = \sum_{i=1}^{\log n} s(2^i) = \sum_{i=1}^{\log n} i = O(\log^2 n)$$

Αριθμός επεξεργαστών

$$q(2) = 1, \quad i = 1$$



Σχήμα 4.1: $(2,2)$ και $(4,4)$ δίκτυα

$$q(2^i) = 2q(2^{i-1}) + 2^{i-1} - 1, \quad i > 1$$

άρα $q(2^i) = (i-1)2^{i-1} + 1$ και

$$p(n) = \sum_{i=1}^{\log n} 2^{(\log n)-i} \cdot q(2^i) = O(n \log^2 n)$$

Κόστος

$$c(n) = p(n) \cdot t(n) = O(n \log^4 n)$$

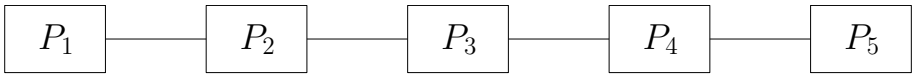
Συνεπώς το δίκτυο ταξινόμησης δεν είναι βέλτιστο.

Παρατηρήσεις

- Το δίκτυο είναι πολύ γρήγορο.
- Ο αριθμός των συγκριτών είναι πολύ μεγάλος.
- Η σύνδεση των συγκριτών παρουσιάζει πρακτικές δυσκολίες.

4.3 Ταξινόμηση σε ένα μονοδιάστατο πίνακα επεξεργαστών

Υποθέτουμε ότι έχουμε ένα SIMD υπολογιστή όπου οι επεξεργαστές είναι συνδεδεμένοι όπως στο σχήμα 4.2:



Σχήμα 4.2: Σύνδεση επεξεργαστών

Ο αλγόριθμος που θα αναπτυχθεί για την ταξινόμηση της ακολουθίας $S = \{s_1, s_2, \cdots, s_n\}$ χρησιμοποιεί n επεξεργαστές $P_1, P_2, \cdots, P_N$.

Διαδικασία 17: QUICKSORT (S)

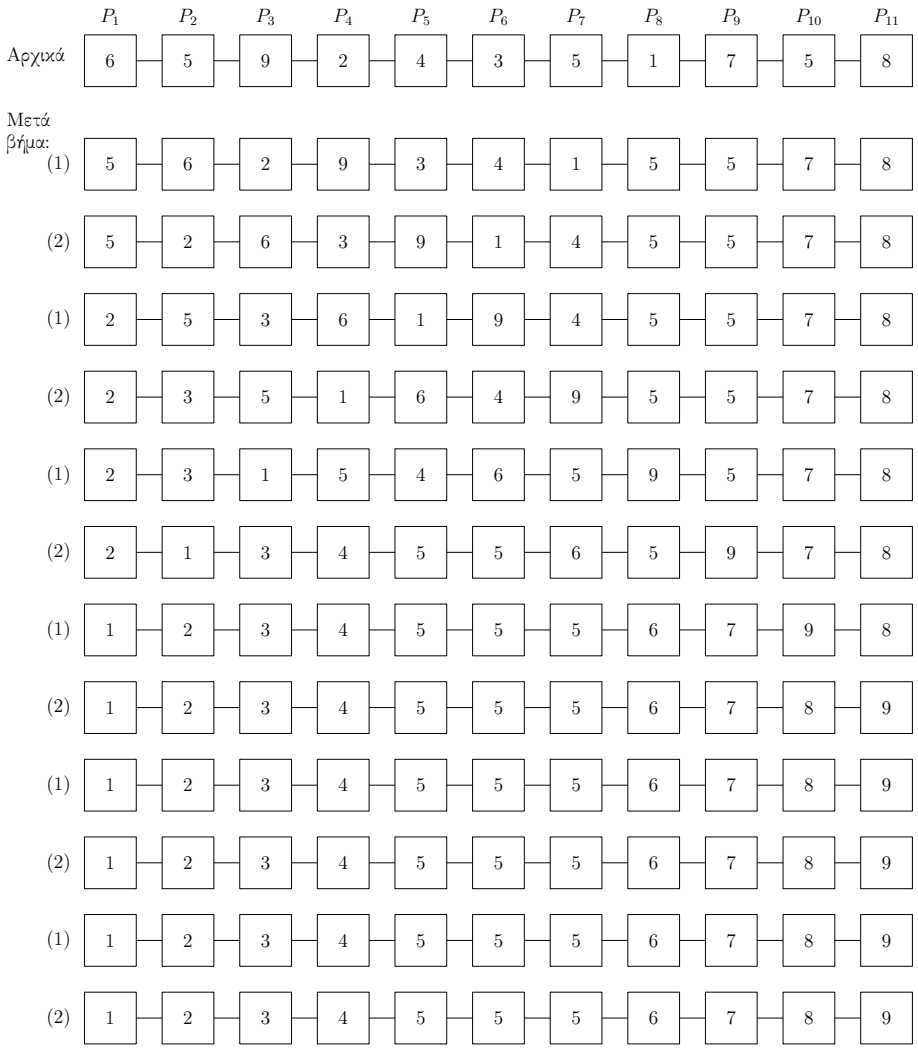
αν $|S| = 2$ ***και*** $S_2 < S_1$ **τότε**
  $S_1 \leftrightarrow S_2$
αλλιώς
  αν $|S| > 2$ **τότε**
    (1) {Προσδιορισμός του m, μέσου στοιχείου της S}
    SEQUENTIAL SELECT $(S, \lceil |S|/2 \rceil)$
    (2) {Χωρισμός της S σε δύο υποακολουθίες S_1 και S_2}
      (2.1) $S_1 \leftarrow \{s_i : s_i \leqslant m\}$ και $|S_1| = \lceil |S|/2 \rceil$
      (2.2) $S_2 \leftarrow \{s_i : s_i \geqslant m\}$ και $|S_2| = \lceil |S|/2 \rceil$
    (3) QUICKSORT (S_1)
    (4) QUICKSORT (S_2)
  τέλος
τέλος

$t(n) = cn + 2t(n/2), \quad c =$ σταθερά
$t(n) = O(n \log n)$ βέλτιστος χρόνος

Κατά τη διάρκεια εκτέλεσης του αλγορίθμου ο επεξεργαστής P_i έχει ένα στοιχείο x_i, $i = 1, 2, \cdots, n$. Αρχικά $x_i = s_i$. Ο αλγόριθμος επαναλαμβάνει στην ουσία δύο βήματα. Στο πρώτο βήμα όλοι οι περιττοί επεξεργαστές P_i λαμβάνουν το x_{i+1} από τον P_{i+1}. Αν $x_i > x_{i+1}$ τότε οι P_i και P_{i+1} ανταλλάσουν τα στοιχεία τους. Στο δεύτερο βήμα όλοι οι άρτιοι επεξεργαστές εκτελούν την ίδια εργασία όπως οι περιττοί στο πρώτο βήμα. Μετά από $\lceil n/2 \rceil$ επαναλήψεις αυτών των δύο βημάτων θα έχουν τελειώσει όλες οι ανταλλαγές των στοιχείων.

Ανάλυση

$$t(n) = O(n), \quad p(n) = n, \quad c(n) = O(n^2)$$



Σχήμα 4.3: Ακολουθία ταξινόμησης 11 στοιχείων με τη διαδικασία ODD-EVEN TRANSPOSITION

Διαδικασία 18: ODD-EVEN TRANSPOSITION (S)

για $j = 1$ **έως** $\lceil n/2 \rceil$ **κάνε**
  (1) **για** $i = 1, 3, \cdots,$ **έως** $2\lfloor n/2 \rfloor - 1$ **κάνε παράλληλα**
    αν $x_i > x_{i+1}$ **τότε**
      $x_i \leftrightarrow x_{i+1}$
    τέλος
  τέλος
  (2) **για** $i = 2, 4, \cdots,$ **έως** $2\lfloor (n-1)/2 \rfloor - 1$ **κάνε παράλληλα**
    αν $x_i > x_{i+1}$ **τότε**
      $x_i \leftrightarrow x_{i+1}$
    τέλος
  τέλος
τέλος

άρα δεν είναι βέλτιστος.

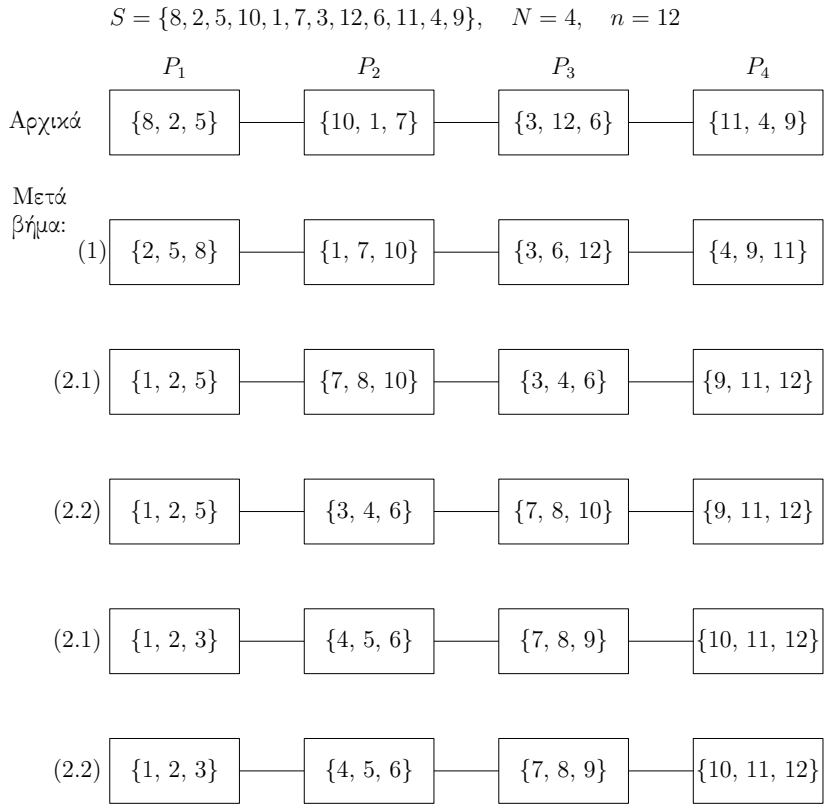
Παρατηρήσεις

- Πετυχαίνει σε σχέση με την Quicksort μια ταχύτητα $O(\log n)$
- Χρησιμοποιεί μεγάλο αριθμό επεξεργαστών
- Δεν είναι βέλτιστου κόστους

Νέος Αλγόριθμος

Υποθέτουμε ότι κάθε ένας P_i από τους $N < n$ επεξεργαστές έχει μία υποακολουθία s_i της S μήκους n/N (προσθέτουμε εικονικά στοιχεία αν το n δεν είναι πολ/σιο του N). Στον νέο αλγόριθμο οι συγκρίσεις - ανταλλαγές αντικαθίστανται με συγχώνευση - χωρισμό στις υποακολουθίες. Στο βήμα 1 κάθε επεξεργαστής P_i ταξινομεί την S_i με την QUICKSORT. Στο βήμα 2.1 κάθε περιττός επεξεργαστής συγχωνεύει τις δύο υποακολουθίες S_i και S_{i+1} σε μία ταξινομημένη ακολουθία $S'_i = \{s'_1, s'_2, \cdots, s'_{2b/N}\}$. Διατηρεί το πρώτο μέρος της S'_i και καταχωρεί στον γειτονικό του επεξεργαστή P_{i+1} το δεύτερο τμήμα. Το βήμα 2.2 είναι το ίδιο με το 2.1 εκτός ότι αυτή τη φορά χρησιμοποιούνται οι άρτιοι επεξεργαστές. Μετά από $\lceil N/2 \rceil$ επαναλήψεις έχουν τελειώσει όλες οι ανταλλαγές των στοιχείων μεταξύ δύο επεξεργαστών. Στο τέλος η ακολουθία $S = \{s_1, s_2, \cdots, s_N\}$ είναι ταξινομημένη.

Ανάλυση



Σχήμα 4.4: Ακολουθία ταξινόμησης 12 στοιχείων με τη διαδικασία MERGE SPLIT

Το Βήμα 1 απαιτεί $O((n/N)\log(n/N))$ βήματα. Μεταφορά της S_{i+1} στον P_i, συγχώνευση με SEQUENTIAL MERGE, και επιστροφή της S_{i+1} στον P_{i+1} απαιτούν $O(n/N)$ χρόνο. Συνεπώς

$$t(n) = O((n/N)\log(n/N)) + \lceil N/2 \rceil \cdot O(n/N)$$

$$= O((n \log n)/N) + O(n)$$

$$c(n) = O(n \log n) + O(nN)$$

το οποίο είναι βέλτιστο όταν $N \leqslant \log n$.

$$T = \Theta(\frac{n}{p} \log \frac{n}{p}) + \Theta(n) + \Theta(n).$$

$$S = \frac{\Theta(n \log n)}{\Theta((n/p)\log(n/p)) + \Theta(n)}$$

Διαδικασία 19: MERGE SPLIT (S)

Βήμα 1
για $i = 1$ **έως** N **κάνε παράλληλα**
  QUICKSORT (S_i)
τέλος

Βήμα 2
για $j = 1$ **έως** $\lceil N/2 \rceil$ **κάνε**
  (2.1) **για** $i = 1, 3, \cdots,$ **έως** $2\lceil N/2 \rceil - 1$ **κάνε παράλληλα**
    (i) SEQUENTIAL MERGE (S_i, S_{i+1}, S'_i)
    (ii) $S_i \leftarrow \{s'_1, s'_2, \cdots, s'_{n/N}\}$
    (iii) $S_{i+1} \leftarrow \{S'_{(n/N)+1}, S'_{(n/N)+2}, \cdots, S'_{2n/N}\}$
  τέλος
  (2.2) **για** $i = 2, 4, \cdots,$ **έως** $2\lfloor (N-1)/2 \rfloor$ **κάνε παράλληλα**
    (i) SEQUENTIAL MERGE (S_i, S_{i+1}, S'_i)
    (ii) $S_{i+1} \leftarrow \{s'_1, s'_2, \cdots, s'_{n/N}\}$
    (iii) $S_{i+1} \leftarrow \{S'_{(n/N)+1}, S'_{(n/N)+2}, \cdots, S'_{2n/N}\}$
  τέλος
τέλος

Διαδικασία 20: BITONIC SORT (ετικέτα, d)

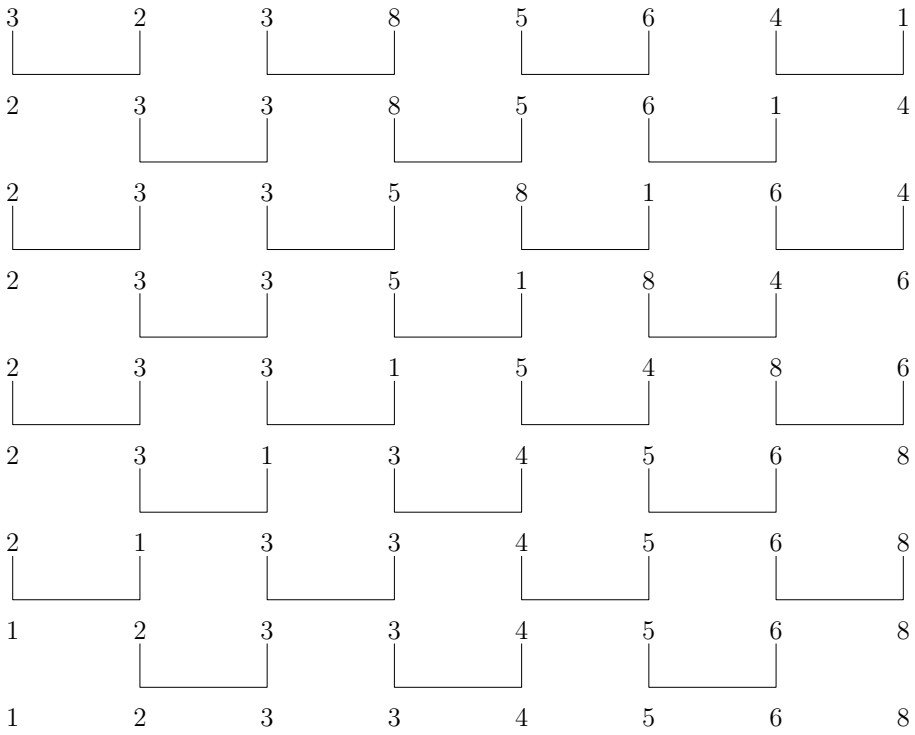
για $i = 0$ **έως** $d - 1$ **κάνε**
  για $j = i$ **έως** 0 **κάνε**
    αν $(i+1)^{\text{οστό}}$ *bit της ετικέτας* $\neq j^{\text{οστού}}$ *bit της ετικέτας* **τότε**
      comp_exchange_max (j)
    αλλιώς
      comp_exchange_min (j)
    τέλος
  τέλος
τέλος

Διαδικασία 21: BUBBLE SORT (n)

για $i = n - 1$ **έως** 1 **κάνε**
  για $j = 1$ **έως** i **κάνε**
    comp_exchange (α_j, α_{j+1})
  τέλος
τέλος

Διαδικασία 22: ODD-EVEN (n)

```
για i = 1 έως n κάνε
    αν i περιττός τότε
        για j = 0 έως n/2 − 1 κάνε
            comp_exchange (α_{2j+1}, α_{2j+2})
        τέλος
    τέλος
    αν i άρτιος τότε
        για j = 1 έως n/2 − 1 κάνε
            comp_exchange (α_{2j}, α_{2j+1})
        τέλος
    τέλος
τέλος
```



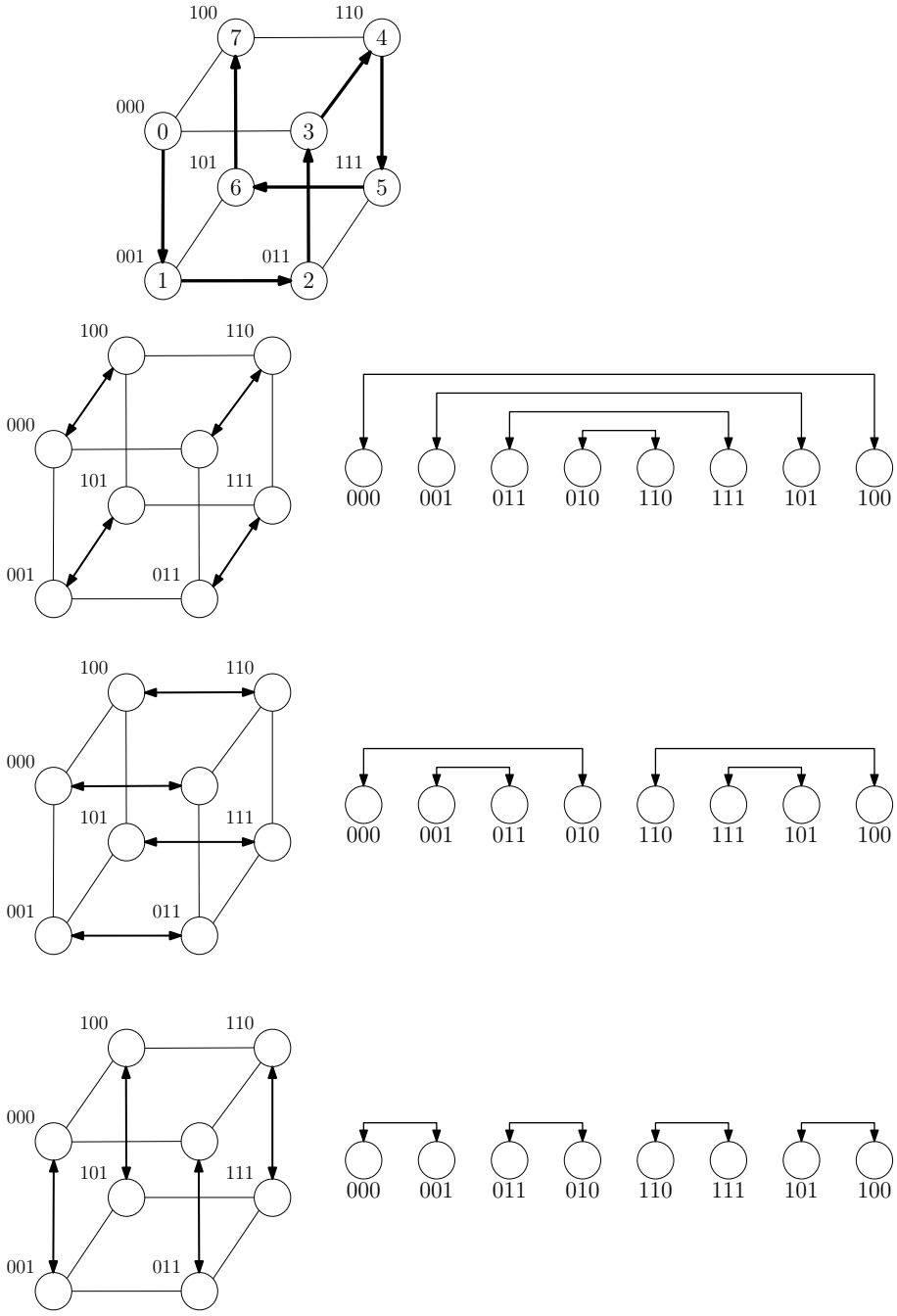
Σχήμα 4.5: Ιδέα της διαδικασίας ODD-EVEN

$$E = \frac{1}{1 - \Theta((\log p)/(\log n)) + \Theta(p/\log n)}$$

Διαδικασία 23: ODD-EVEN PAR (n)

```
id = ετικέτα επεξεργαστή
για i = 1 έως n κάνε
    αν i περιττός τότε
        αν id περιττός τότε
            comp_exchange_min (id + 1)
        αλλιώς
            comp_exchange_max (id − 1)
        τέλος
    τέλος
    αν i άρτιος τότε
        αν id άρτιος τότε
            comp_exchange_min (id + 1)
        αλλιώς
            comp_exchange_max (id − 1)
        τέλος
    τέλος
τέλος
```

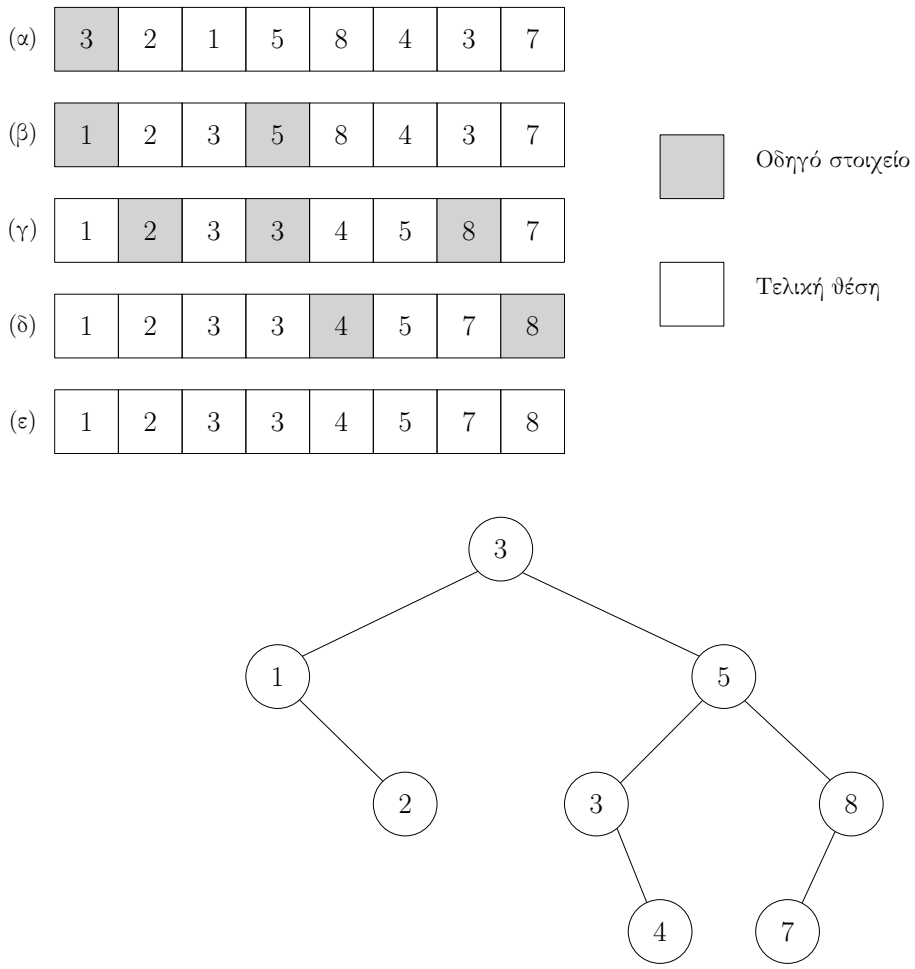
$$T_p = \Theta(\frac{n}{p} \log \frac{n}{p}) + \Theta(\frac{n}{p} \log p) + \Theta(l\frac{n}{p})$$



Σχήμα 4.6: Ιδέα της ODD-EVEN PAR

Διαδικασία 24: QUICKSORT (A, q, r)

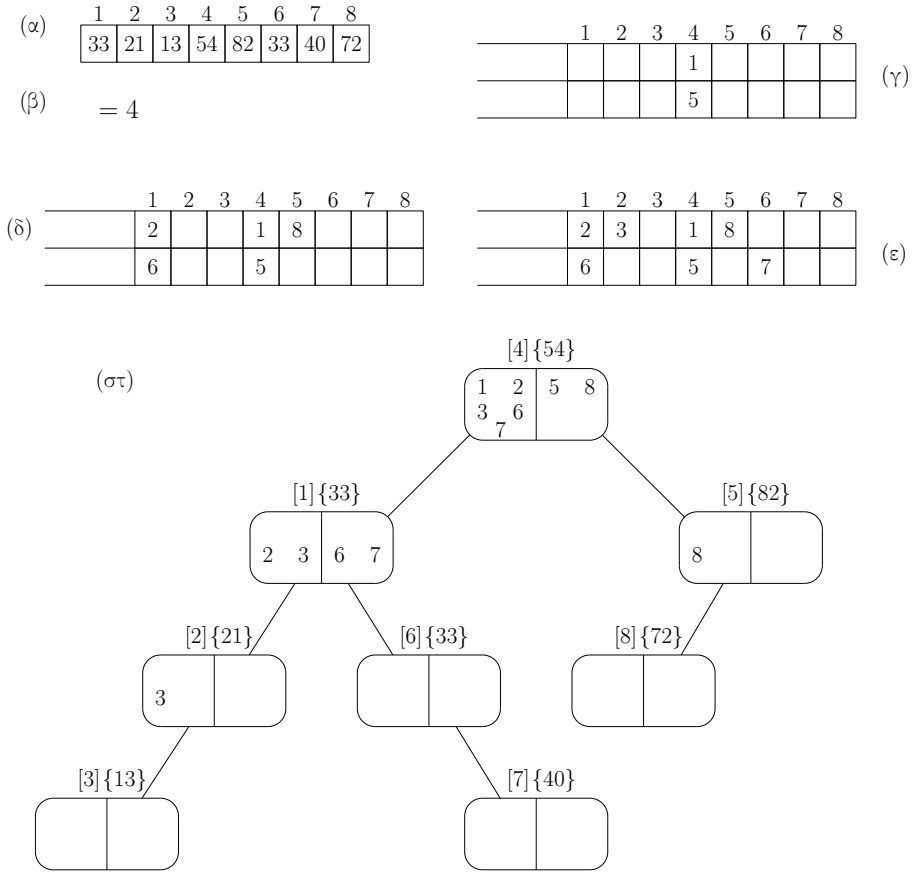
```
αν q < r τότε
    x = A[q]
    s = q
    για i = q + 1 έως r κάνε
        αν A[i] ⩽ x τότε
            s = s + 1
            swap (A[s], A[i])
        τέλος
        swap (A[q], A[s]);
        QUICKSORT (A, q, s)
        QUICKSORT (A, s + 1, r)
    τέλος
τέλος
```



Σχήμα 4.7: Ιδέα της QUICKSORT

Διαδικασία 25: BUILD TREE $(A[1 \cdots n])$

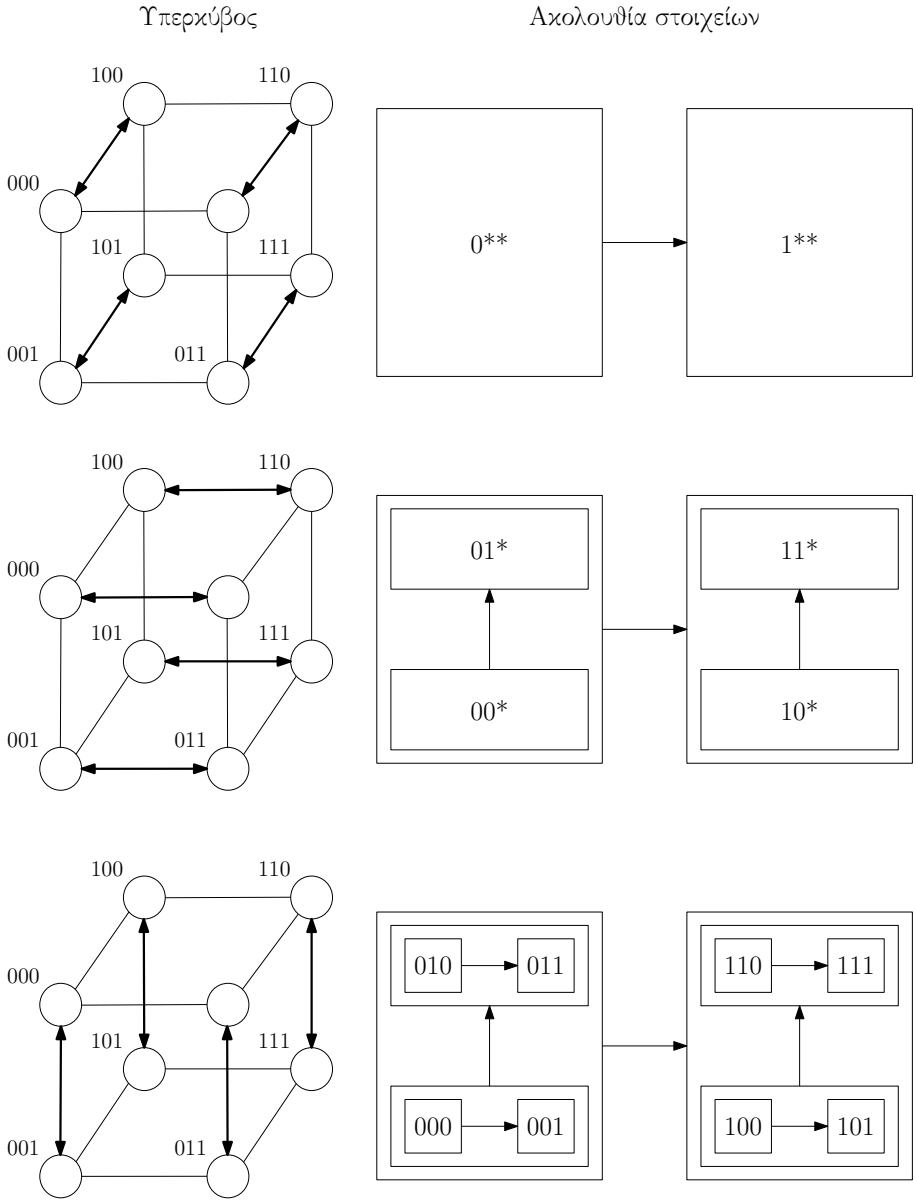
```
για κάθε επεξεργαστή i κάνε
    ρίζα = i
    γονέας_i = ρίζα
    αριστερό_παιδί[i] = δεξιό_παιδί[i] = n + 1
τέλος
επανάλαβε
    αν (A[i] < A[γονέας_i]) ή (A[i] = A[γονέας_i]) και
     (i < γονέας_i) τότε
        αριστερό παιδί[γονέας_i] = i
        αν i = αριστερό παιδί[γονέας_i] τότε
            έξοδος
        αλλιώς
            γονέας_i = αριστερό παιδί[γονέας_i]
        τέλος
    αλλιώς
        δεξιό παιδί[γονέας_i] = i
        αν i = δεξιό παιδί[γονέας_i] τότε
            έξοδος
        αλλιώς
            γονέας_i = δεξιό παιδί[γονέας_i]
        τέλος
    τέλος
μέχρι για κάθε επεξεργαστή i ≠ ρίζα
```



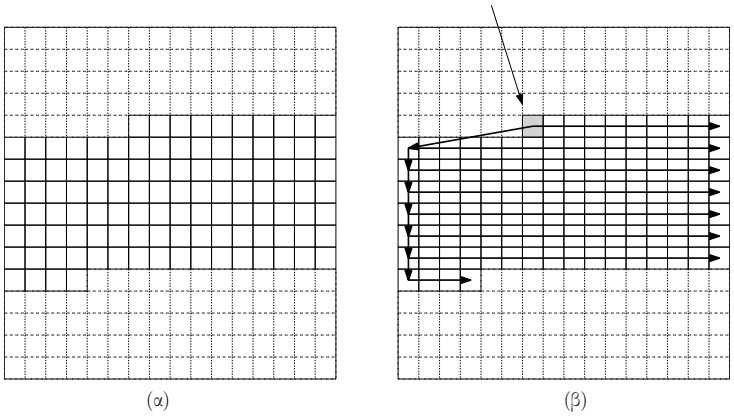
Σχήμα 4.8: Ιδέα της BUILD TREE

Διαδικασία 26: HYPERCUBE QUICKSORT (B, n)

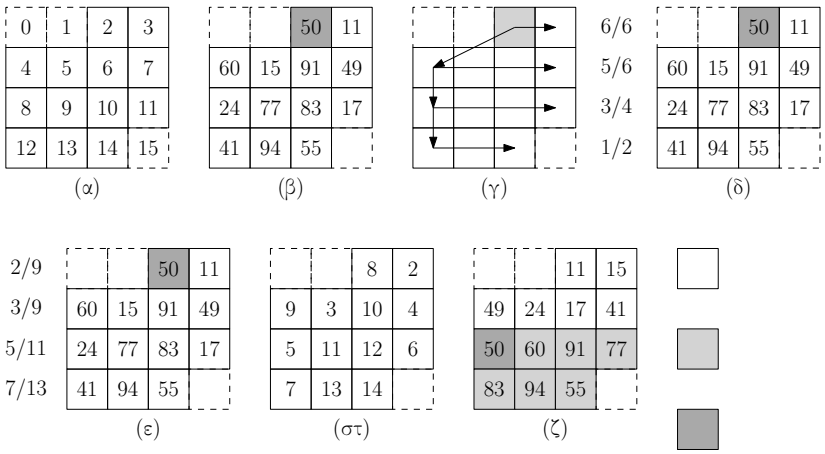
```
id = ετικέτα επεξεργαστή
για i = 1 έως d κάνε
    x = οδηγός
    χώρισε το B σε B_1 και B_2 έτσι ώστε B_1 ⩽ x < B_2
    αν i^{οστό} bit = 0 τότε
        στείλε B_2 στον επεξεργαστή στο i^{οστό} κανάλι επικοινωνίας
        C = υποακολουθία λαμβάνεται στο i^{οστό} κανάλι
         επικοινωνίας
        B = B_1 ⋃ C
    αλλιώς
        στείλε B_1 στον επεξεργαστή στο i^{οστό} κανάλι επικοινωνίας
        C = υποακολουθία λαμβάνεται στο i^{οστό} κανάλι
         επικοινωνίας
        B = B_2 ⋃ C
    τέλος
τέλος
Ταξινόμησε B με την SEQUENTIAL QUICKSORT
```



Σχήμα 4.9: Ιδέα της HYPERCUBE SORT



Σχήμα 4.10



Σχήμα 4.11

$$T_p = \Theta(\frac{n}{p}\log\frac{n}{p}) + \Theta(\frac{n}{p}\log p) + \Theta(\log^2 p).$$

$$S = \frac{\Theta(n\log n)}{\Theta((n/p)\log(n/p)) + \Theta((n/p)\log p) + \Theta(\log^2 p)}$$

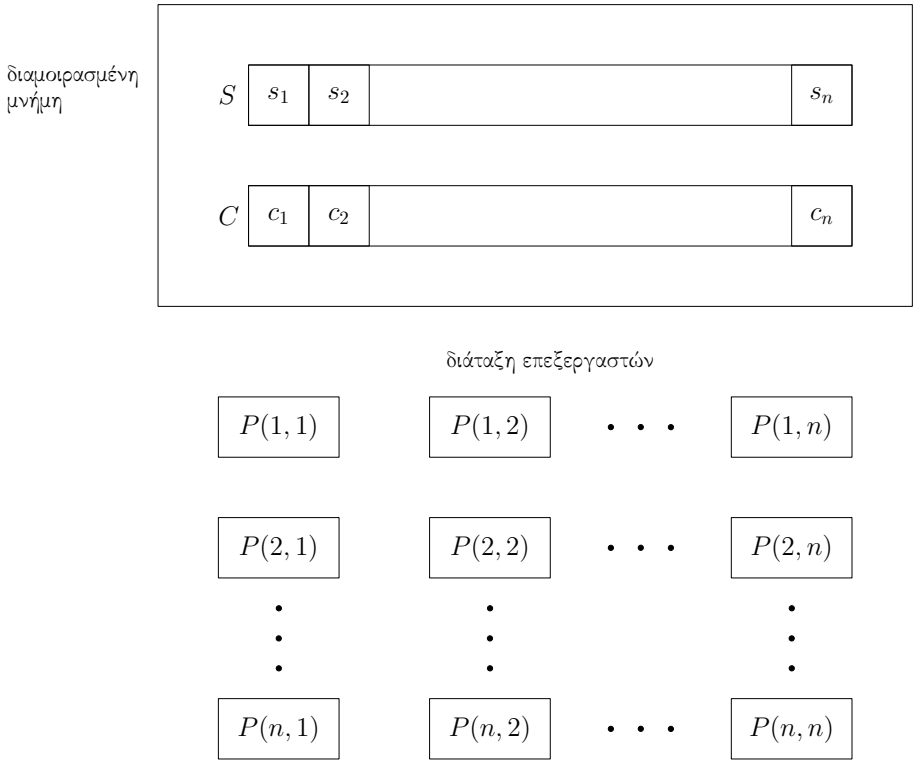
$$E = \frac{1}{1 + \Theta((\log p)/(\log n)) + \Theta((p\log^2 p)/(n\log n))}$$

Διαδικασία 27: ENUM SORT (n)

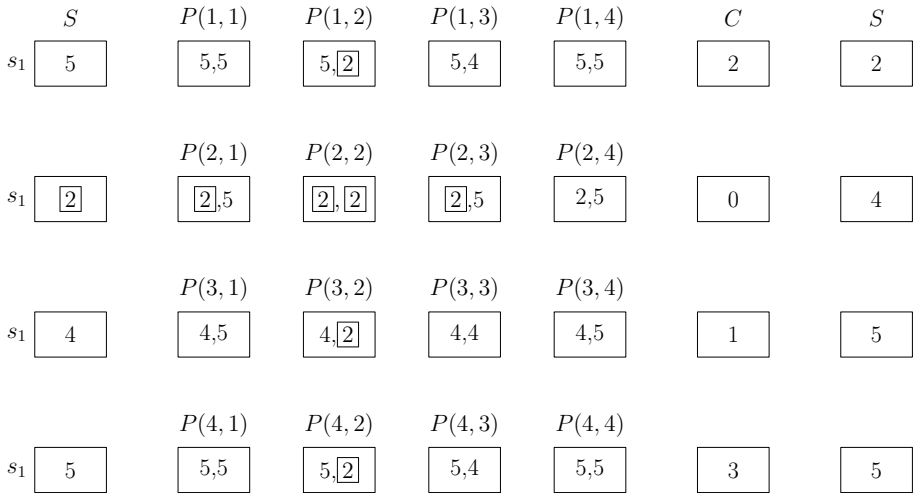
για *κάθε επεξεργαστή* $P_{1,j}$ **κάνε**
| $C[j] = 0$
τέλος
για *κάθε επεξεργαστή* $P_{i,j}$ **κάνε**
| **αν** $(A[i] < A[j])$ ***ή*** $(A[i] = A[j])$ ***και*** $(i < j)$ **τότε**
| | $C[j] = 1$
| **αλλιώς**
| | $C[j] = 0$
| **τέλος**
τέλος
για *κάθε επεξεργαστή* $P_{1,j}$ **κάνε**
| $A[C[j]] = A[j]$
τέλος

4.4 Ταξινόμηση στο CRCW Μοντέλο

Υποθέτουμε ότι έχουμε το SM SIMD CRCW μοντέλο, όπου υποθέτουμε ότι οι συγκρούσεις γραφής (προσπάθεια γραφής διαφορετικών δεδομένων στην ίδια θέση μνήμης) επιλύονται με την αποθήκευση του αθροίσματος των ακεραίων στη διεύθυνση αυτή.
Υποθέτουμε επίσης ότι n^2 επεξεργαστές είναι διαθέσιμοι στο CRCW μοντέλο για την ταξινόμηση της ακολουθίας $S = \{s_1, s_2, ..., s_n\}$. Ο αλγόριθμος της ταξινόμησης βασίζεται στην ιδέα της ταξινόμησης με απαρίθμηση. Η θέση του κάθε στοιχείου s_i της S στην ταξινομημένη ακολουθία προσδιορίζεται με τον υπολογισμό του c_i που συμβολίζει το πλήθος των στοιχείων μικρότερα από αυτό. Αν $s_i = s_j$ τότε σαν μεγαλύτερο λαμβάνεται το s_i αν $i > j$, διαφορετικά λαμβάνεται το s_j. Από τη στιγμή που έχουν υπολογιστεί όλα τα c_i, το s_i τοποθετείται στη θέση $1 + c_i$ της ταξινομημένης ακολουθίας. Για ευκολία υποθέτουμε ότι οι επεξεργαστές αποτελούν ένα διδιάστατο πίνακα όπως φαίνεται στο σχήμα.



Σχήμα 4.12: Διαμοιραζόμενη μνήμη και διάταξη επεξεργαστών στο CRCW SM SIMD μοντέλο



Σχήμα 4.13: Παράδειγμα για $S = \{5, 1, 4, 5\}$

Παράδειγμα

$$S = \{5, 1, 4, 5\}$$

- κάθε στοιχείο s_i διαβάζεται ταυτόχρονα από όλους τους επεξεργαστές στην i γραμμή και i στήλη
- όλοι οι επεξεργαστές σε μια δεδομένη γραμμή γράφουν στην ίδια θέση μνήμης.

Η διαμοιραζόμενη μνήμη περιέχει δύο πίνακες: Η αρχική ακολουθία έχει καταχωρηθεί στον πίναξα S, ενώ το πλήθος των στοιχείων που είναι μικρότερα του s_i, c_i, έχουν καταχωρηθεί στον πίνακα C. Η τελική ταξινομημένη ακολουθία καταχωρείται στον πίνακα S. Η $i-$οστή σειρά των επεξεργαστών ¨είναι υπεύθυνη¨ για το στοιχείο s_i: οι επεξεργαστές $P(i, 1), P(i, 2), ..., P(i, n)$ υπολογίζουν το c_i και αποθηκεύουν το s_i στη θέση $c_i + 1$.

$$t(n) = O(1)!$$

$$p(n) = n^2$$

$$c(n) = c(n^2): \text{ όχι βέλτιστο κόστος}$$

Διαδικασία 28: CRCW SORT (S)

Βήμα 1
για $i = 1$ **έως** n **κάνε παράλληλα**
  για $j = 1$ **έως** n **κάνε παράλληλα**
    αν $(s_i > s_j)$ ***ή*** $(s_i = s_j$ ***και*** $i > j)$ **τότε**
      ο $P(i,j)$ γράφει 1 στην c_i
    αλλιώς
      ο $P(i,j)$ γράφει 0 στην c_i
    τέλος
  τέλος
τέλος

Βήμα 2
για $i = 1$ **έως** n **κάνε παράλληλα**
  ο $P(i,1)$ αποθηκεύει το s_i στη θέση $1 + c_i$ του S
τέλος

4.5 Ταξινόμηση στο CREW Μοντέλο

Σκοπός μας είναι να σχεδιάσουμε έναν αλγόριθμο, ο οποίος δεν απαιτεί ταυτόχρονη γραφή και χρησιμοποιεί ένα μικρό αριθμό αριθμό επεξεργαστών. Στη συνέχεια θα χρησιμοποιηθεί πάλι ένας αλγόριθμος συγχώνευσης για την ταξινόμηση (CREW MERGE).
Υποθέτουμε ότι έχουμε ένα (CREW SM SIMD) υπολογιστή με N επεξεργαστές $P_1, P_2, ..., P_N$ για την ταξινόμηση της ακολουθίας $S = s_1, s_2, ..., s_N$, όπου $N \leqslant n$. Τα στοιχεία της S κατανέμονται ομοιόμορφα μεταξύ των N επεξεργαστών $\left(\lceil \frac{n}{N} \rceil\right)$ στοιχεία ανά επεξεργαστή). Κάθε επεξεργαστής ταξινομεί την ακολουθία του με τη χρήση της QUICKSORT. Οι N ταξινομημένες υποακολουθίες συγχωνεύονται ανά δύο, ταυτόχρονα, με τη χρήση της CREW MERGE για κάθε ζευγάρι. Οι προκύπτουσες ακολουθίες συγχωνεύονται ξανά κατά ζεύγη μέχρις ότου ληφθεί μια ακολουθία με n στοιχεία. Στη συνέχεια, δίνεται ο αλγόριθμος, όπου S_j^k συμβολίζει τη συγχωνευμένη ακολουθία και P_j^k τους επεξεργαστές που έκαναν τη συγχώνευση.

Διαδικασία 29: CREW SORT (S)

Βήμα 1

για $i = 1$ **έως** N **κάνε παράλληλα**

Ο επεξεργαστής P_i

(1.1) διαβάζει μια διακεκριμένη υποακολουθία S_i του S μεγέθους n/N

(1.2) QUICKSORT(S_i)

(1.3) $S_i^1 \leftarrow S_i$

(1.4) $P_i^1 \leftarrow \{P_i\}$

τέλος

Βήμα 2

(2.1) $u \leftarrow 1$

(2.2) $v \leftarrow N$

(2.3) **όσο** $v > 1$ **κάνε**

(2.3.1) **για** $m = 1$ **έως** $\lfloor v/2 \rfloor$ **κάνε παράλληλα**

(i) $P_m^{u+1} \leftarrow P_{2m-1}^u \cup P_{2m}^u$

(ii) Οι επεξεργαστές του συνόλου P_m^{u+1} εκτελούν την CREW MERGE $(S_{2m-1}^u, S_{2m}^u, S_m^{u+1})$

τέλος

(2.3.2) **αν** v *είναι περιττό* **τότε**

(i) $P_{\lceil v/2 \rceil}^{u+1} \leftarrow P_v^u$

(ii) $S_{\lceil v/2 \rceil}^{u+1} \leftarrow S_v^u$

τέλος

(2.3.3) $u \leftarrow u + 1$

(2.3.4) $v \leftarrow \lceil v/2 \rceil$

τέλος

Ανάλυση

Η QUICKSORT απαιτεί $O((\frac{n}{N})\log(\frac{n}{N}))$ χρόνο. Σε κάθε επανάληψη του βήματος 2.3 συγχωνεύονται $\lfloor\frac{\nu}{2}\rfloor$ ζεύγη υπακολουθιών με $\frac{n}{\lfloor\frac{\nu}{2}\rfloor}$ στοιχεία για κάθε ζεύγος με τη χρήση $\frac{N}{\lfloor\frac{\nu}{2}\rfloor}$ επεξεργαστών για κάθε ζεύγος. Η CREW MERGE απαιτεί $O\left([(n/\lfloor\nu/2\rfloor)/(N/\lfloor\nu/2\rfloor)] + \log(n/\lfloor\nu/2\rfloor)\right)$, δηλαδή $O(n/N + \log n)$ χρόνο. Επειδή το βήμα 2.3 επαναλαμβάνεται $\lfloor\log N\rfloor$ φορές, ο συνολικός χρόνος της CREW SORT είναι

$$t(n) = \underbrace{O\left(\left(\frac{n}{N}\right)\log\left(\frac{n}{N}\right)\right)}_{\text{QUICKSORT}} + O\left(\left(\frac{n}{N}\right)\log N + \log n \log N\right) =$$
$$= O\left(\left(\frac{n}{N}\right)\log n + \log^2 n\right)$$

Επειδή $p(n) = N$, το κόστος είναι $c(n) = O(n\log n + N\log^2 n)$ το οποίο είναι βέλτιστο για $N \leqslant n/\log n$.

Παράδειγμα

Έστω $S = \{2, 8, 5, 10, 15, 1, 12, 6, 14, 3, 11, 7, 9, 4, 13, 16\}$ και $N = 4$. Έχουμε $n = 16$ και $\frac{n}{N} = 4$, άρα στο Βήμα 1 οι επεξεργαστές P_1, P_2, P_3 και P_4 λαμβάνουν τις υπακολουθίες:

$$S_1 = \{2, 8, 5, 10\}, \quad S_2 = \{15, 1, 12, 6\}, \quad S_3 = \{14, 3, 11, 7\} \quad \text{και}$$
$$S_4 = \{9, 4, 13, 16\}$$

αντίστοιχα, τις οποίες ταξινομούν τοπικά. Στο τέλος του βήματος 1, θα έχουμε:

$$S_1^1 = \{2, 5, 8, 10\},\ S_2^1 = \{1, 6, 12, 15\},\ S_3^1 = \{3, 7, 11, 14\},\ S_4^1 = \{4, 9, 13, 16\}$$

$$P_1^1 = \{P_1\}, \quad P_2^1 = \{P_2\}, \quad P_3^1 = \{P_3\}, \quad P_4^1 = \{P_4\}$$

Στην πρώτη επανάληψη του βήματος 2.3 οι επεξεργαστές στο $P_1^2 = P_1^1 \cup P_2^1 = \{P_1, P_2\}$ συγχωνεύουν τα στοιχεία των S_1^1 και S_2^1 για το σχηματισμό της

$$S_1^2 = \{1, 2, 5, 6, 8, 10, 12, 15\}.$$

Ταυτόχρονα οι επεξεργαστές του $P_2^2 = P_3^1 \cup P_4^1 = \{P_3, P_4\}$ συγχωνεύουν τις S_3^1 και S_4^1 στην

$$S_2^2 = \{3, 4, 7, 9, 11, 13, 14, 16\}.$$

Στη δεύτερη επανάληψη οι επεξεργαστές στο $P_1^3 = P_1^2 \cup P_2^2 = \{P_1, P_2, P_3, P_4\}$ συγχωνεύουν τις S_1^2 και S_2^2 στην

$$S_1^3 = \{1, 2, ..., 16\}.$$

4.6 Ταξινόμηση στο EREW Μοντέλο

Υποθέτουμε ότι έχουμε στη διάθεσή μας N επεξεργαστές $P_1, P_2, ..., P_N$ σε ένα EREW SM SIMD υπολογιστή για την ταξινόμηση της ακολουθίας $S = \{s_1, s_2, ..., s_n\}$, όπου $N < n$.

4.6.1 Προσομοίωση της CREW SORT

Ο απλούστερος τρόπος προκειμένου να αποφευχθεί το ταυτόχρονο διάβασμα στην CREW SORT είναι να χρησιμοποιηθεί η MULTIPLE BROADCAST. Η προσομοίωση αυτή της CREW SORT στο EREW μοντέλο απαιτεί

$$t(n) = O((n/N)\log n + \log n \log N) \times \underbrace{\log N}_{\text{BROADCAST}} = \\ = O(((n/N) + \log N)\log n \log N)$$

με κόστος

$$c(n) = O((n + N\log N)\log n \log N)$$

το οποίο δεν είναι βέλτιστο.

4.6.2 Ταξινόμηση χωρίς συγκρούσεις διαβάσματος

Η αντικατάσταση της CREW MERGE με την EREW MERGE εξαλείφει το πρόβλημα της σύγκρουσης του διαβάσματος. Το βήμα αυτό απαιτεί $O((n/N) + \log n \log N)$ και επειδή υπάρχουν $\log N$ επαναλήψεις, ο συνολικός χρόνος θα είναι:

$$t(n) = O((n/N)\log(n/N)) + O((n/N)\log N + \log n \log^2 N) = \\ = O(((n/N) + \log^2 n)\log n)$$

με κόστος

$$c(n) = O((n + N\log^2 n)\log n)$$

το οποίο είναι βέλτιστο όταν $N \leqslant n\log^2 n$.

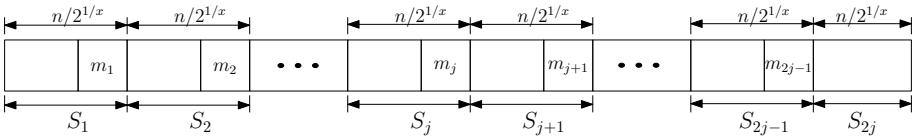
4.6.3 Ταξινόμηση με επιλογή

Στον αλγόριθμο αυτό χρησιμοποιείται πάλι η QUICKSORT. Σημειώνουμε ότι $N < n$ και μπορούμε να γράψουμε $N = n^{1-x}$, όπου $0 < x < 1$.
Έστω m_i συμβολίζει το $\lceil i(n/2^{1/x})\rceil$-οστό μικρότερο στοιχείο της S για

$1 \leqslant i \leqslant 2^{\frac{1}{x}} - 1$. Τα m_i διαιρούν την S σε $2^{\frac{1}{x}}$ υποακολουθίες μήκους $\frac{n}{2^{\frac{1}{x}}}$ η κάθε μία. Αυτές οι υποακολουθίες συμβολίζονται με

$$S_1, S_2, ..., S_j, S_{j+1}, S_{j+2}, ..., S_{2j}$$

όπου $j = 2^{(\frac{1}{x})-1}$ και ικανοποιούν την ιδιότητα: Κάθε στοιχείο της S_i είναι μικρότερο ή ίσο με κάθε στοιχείο της S_{i+1} για $1 \leqslant i \leqslant 2j-1$ (βλ. σχήμα). Η εργασία της υποδιαίρεσης μπορεί τώρα να εκτελεστεί αναδρομικά σε κάθε μία από τις υποακολουθίες S_i μέχρις ότου όλη η υποακολουθία S ταξινομηθεί.



Σχήμα 4.14: Διαίρεση ακολουθίας για ταξινόμηση με επιλογή

Είναι $2j = 2^{\frac{1}{x}}$, $j = 2^{\frac{1}{x}-1}$ και κάθε στοιχείο της S_i είναι μικρότερο ή ίσο από κάθε στοιχείο της S_{i+1} για $1 \leqslant i \leqslant 2j-1$.
Επίσης $N < n, N = n^{1-x}, 0 < x < 1, m_i \left\lceil i\left(\frac{n}{2^{\frac{1}{x}}}\right)\right\rceil$-οστό μικρότερο στοιχείο της $S, 1 \leqslant i \leqslant 2^{\frac{1}{x}} - 1$. Τα m_i διαιρούν την S σε $2^{\frac{1}{x}}$ υποακολουθίες μεγέθους $\frac{n}{2^{\frac{1}{x}}}$ η κάθε μία.
Ο αλγόριθμος εκτελείται παράλληλα καλώντας αρχικά την PARALLEL SELECT για τον προσδιορισμό των στοιχείων m_i και στη συνέχεια της δημιουργίας των υποακολουθιών S_i. Ο αλγόριθμος εφαρμόζεται παράλληλα στις υποακολουθίες

$$S_1, S_2, ..., S_j$$

χρησιμοποιώντας N/j επεξεργαστές για κάθε υποακολουθία. Στη συνέχεια γίνεται το ίδιο για κάθε υποακολουθία από τις

$$S_{j+1}, S_{j+2}, ..., S_{2j}$$

Ας σημειωθεί ότι ο αριθμός των επεξεργαστών για την ταξινόμηση κάθε υποακολουθίας μεγέθους $n/2^{1/x}$, είναι $n^{1-x}/2^{1/x-1}$, δηλαδή ακριβώς ίσος με όσο απαιτεί μια κανονική αναδρομή $(n/2^{1-x})^{1-x}$. Θα πρέπει να τονιστεί ότι το $2^{1/x}$ θα είναι ένας ακέραιος πεπερασμένου μεγέθους, πράγμα που εξασφαλίζει την ύπαρξη ενός ορίου στον εκτελέσιμο χρόνο καθώς και ότι όλα τα m_i υπάρχουν. Αρχικά οι N διαθέσιμοι επεξεργαστές υπολογίζουν το x από την $N = n^{1-x}$. Αν το x δεν ικανοποιεί τις συνθήκες:

$$\text{(i)}\ \left\lceil \frac{1}{x} \right\rceil \leqslant 10^{(1)}\ \ \text{(π.χ.) και}\ \ \text{(ii)}\ n \geqslant 2^{\lceil \frac{1}{x} \rceil (2)}$$

τότε ο μικρότερος πραγματικός αριθμός μεγαλύτερος του x που ικανοποιεί τα (1) και (2) λαμβάνεται ο x. Έστω $k = 2^{\lceil \frac{1}{x} \rceil}$, τότε ο αλγόριθμος δίνεται από την EREW SORT.

Διαδικασία 30: EREW SORT (S)

αν $|S| \leqslant k$ **τότε**
| QUICKSORT(S)
αλλιώς
| (1) **για** $i = 1$ **έως** $k - 1$ **κάνε**
| | PARALLEL SELECT$(S, \lceil \frac{i|S|}{k} \rceil)$ {βρίσκει το m_i}
| **τέλος**
| (2) $S_1 \leftarrow \{s \in S : s \leqslant m_1\}$
| (3) **για** $i = 2$ **έως** $k - 1$ **κάνε**
| | $S_i \leftarrow \{s \in S : m_{i-1} \leqslant s \leqslant m_i\}$
| **τέλος**
| (4) $S_k \leftarrow \{s \in S : s \geqslant m_{k-1}\}$
| (5) **για** $i = 1$ **έως** $\frac{k}{2}$ **κάνε παράλληλα**
| | EREW SORT(S_i)
| **τέλος**
| (6) **για** $i = (\frac{k}{2} + 1)$ **έως** k **κάνε παράλληλα**
| | EREW SORT(S_i)
| **τέλος**
τέλος

Σημειώνεται ότι η ακολουθία S_i (βλ.βήματα 2-4) δημιουργείται με τη μέθοδο που σχετίζεται με την PARALLEL SELECT. Επίσης, στο βήμα 3 τα στοιχεία της S μικρότερα από το m_i και μεγαλύτερα ή ίσα του m_{i-1} τοποθετουνται πρώτα στην S_i. Αν $|S_i| < \lceil \frac{|S|}{k} \rceil$, τότε στοιχεία ίσα με το m_i προστίθενται στο S_i έτσι ώστε είτε $|S_i| = \lceil \frac{|S|}{k} \rceil$ ή δεν υπάρχουν (έχουν απομείνει) για να προστεθούν στην S_i. Τα βήματα 2 και 4 εκτελούνται ανάλογα.

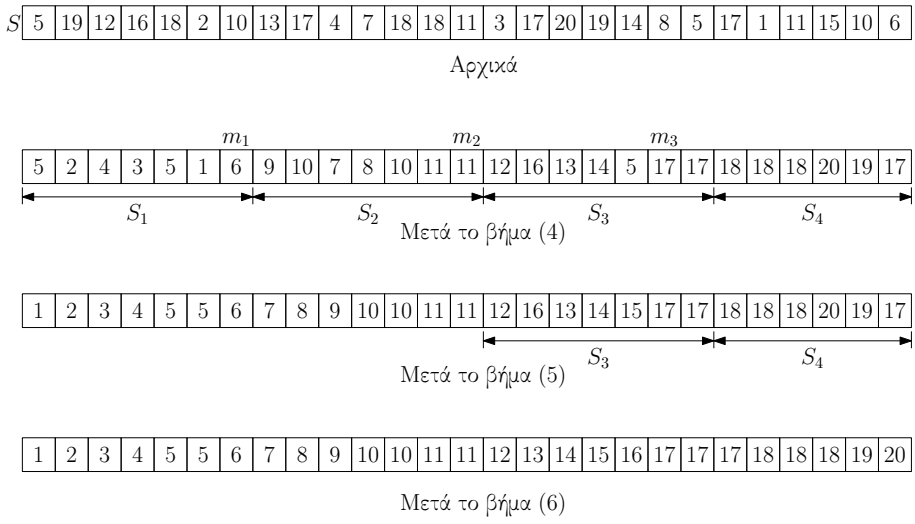
(1) Εξασφαλίζει ότι ο $2^{\frac{1}{x}}$ είναι ακέραιος πεπερασμένου μεγέθους.
(2) Εξασφαλίζει ότι τα m_i θα βρεθούν.

Παράδειγμα

Έστω

$$S = \{5, 9, 12, 16, 18, 2, 10, 13, 17, 4, 7, 18, 18, 11, 3, 17, 20, 19, 14, 8, 5, 17, 1, 11, 15, 10, 6\}$$

δηλαδή $n = 27$ και έχουμε 5 επεξεργαστές P_1, P_2, P_3, P_4 και P_5, δηλαδή $N = 5$, συνεπώς $5 = 27^{1-x} \Rightarrow x \simeq 0.5, k = 2^{\lceil \frac{1}{x} \rceil} = 4$. Άρα, $m_1 = 6, m_2 = 11$ και $m_3 = 17$. Στο βήμα 5 εφαρμόζεται αναδρομικά ο αλγόριθμος και ταυτόχρονα στις S_1 και S_2. Επειδή $|S_1| = |S_2| = 7$ και $\lfloor 7^{1-x} \rfloor = 2$ άρα 2 επεξεργαστές χρησιμοποιούνται για την ταξινόμηση κάθε υποακολουθίας S_1 και S_2 ο πέμπτος παραμένει ανενεργός. Για την S_1 οι επεξεργαστές P_1 και P_2 υπολογίζουν τα $m_1 = 2, m_2 = 4$ και $m_3 = 5$ και δημιουργούνται οι τέσσερεις υποακολουθίες $\{1, 2\}$, $\{3, 4\}$, $\{5, 5\}$ και $\{6\}$ κάθε μία από τις οποίες είναι ήδη ταξινομημένη. Για την S_2, οι επεξεργαστές P_3 και P_4 υπολογίζουν τα $m_1 = 8, m_2 = 10$ και $m_3 = 11$ και τις υποακολουθίες $\{7, 8\}, \{9, 10\}, \{10, 11\}$ και $\{11\}$ που ήδη είναι ταξινομημένες. Στο βήμα 6, ο αλγόριθμος εφαρμόζεται αναδρομικά και ταυτόχρονα στις S_3 και S_4.



Σχήμα 4.15: Ακολουθία S μετά το βήμα 5

Επειδή $|S_3| = 7$ και $|S_4| = 6$, τα 7^{1-x} και 6^{1-x} στρογγυλεύονται στο 2 και έτσι δύο επεξεργαστές χρησιμοποιούνται για την ταξινόμηση κάθε μιας από τις δύο υποακολουθίες S_3 και S_4. Για την $S_3, m_1 = 13, m_2 = 15$ και $m_3 = 17$ και δημιουργούνται οι τέσσερεις υποακολουθίες $\{12, 13\}, \{14, 15\}, \{16, 17\}$ και $\{17\}$ κάθε μία από τις οποίες είναι ήδη ταξινομημένη. Για

την S_4, έχουμε $m_1 = 18, m_2 = 18$ και $m_3 = 20$, οπότε προκύπτουν οι τέσσερεις υποακολουθίες $\{17, 18\}, \{18, 18\}, \{19, 20\}$ και μια κενή. Η ακολουθία S μετά το βήμα 5 παρουσιάζεται στο Σχήμα 4.15.

Ανάλυση

Ο χρόνος για την EREW SORT είναι:

$$t(n) = \underbrace{cn^x}_{\substack{\text{PARALLEL} \\ \text{SELECT}}} + 2t\left(\frac{n}{k}\right) = O(n^x \log n)$$

Επειδή $p(n) = n^{1-x}$,

$$c(n) = p(n)t(n) = O(n \log n)$$

το οποίο είναι βέλτιστο.

Κεφάλαιο 5

ΑΝΑΖΗΤΗΣΗ (Searching)

Η αναζήτηση είναι μια από τις πλέον βασικές πράξεις στην περιοχή των υπολογισμών. Χρησιμοποιείται σε οποιαδήποτε εφαρμογή απαιτείται να βρεθεί αν ένα στοιχείο ανήκει σε μια λίστα ή πιο γενικά ανάκληση πληροφοριών από ένα αρχείο που σχετίζονται με αυτό το στοιχείο.
Το πρόβλημα της αναζήτησης διατυπώνεται ως εξής:
Δίνεται μια ακολουθία $S = \{s_1, s_2, ..., s_n\}$ ακεραίων αριθμών και ένας ακέραιος αριθμός x. Ζητείται να βρεθεί αν $x = s_k$ για κάποιο s_k στην S.
Στους ακολουθιακούς υπολογισμούς το πρόβλημα λύνεται με τη σάρωση της ακολουθίας S και τη σύγκρισή του x με τα διαδοχικά στοιχεία της μέχρις ότου είτε ένας ακέραιος είναι ίσος με x ή τελειώσουν τα στοιχεία της ακολουθίας. Η μέθοδος αυτή είναι η σειριακή ή ακολουθιακή αναζήτηση και δίνεται από την procedure SEQUENTIAL SEARCH. Στη χειρότερη περίπτωση ο υπολογιστικός χρόνος του υποπρογράμματος είναι $O(n)$, το οποίο είναι βέλτιστο, αφού

Διαδικασία 31: SEQUENTIAL SEARCH (S, x, k)

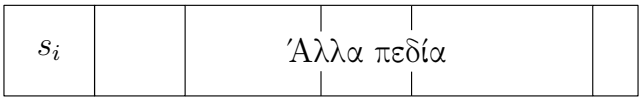
Βήμα 1
(1.1) $i \leftarrow 1$
(1.2) $k \leftarrow 0$

Βήμα 2
όσο $i \leqslant n$ *και* $k = 0$ **κάνε**
  αν $s_i = x$ **τότε**
    $k \leftarrow i$
  τέλος
  $i \leftarrow i + 1$
τέλος

κάθε στοιχείο της S πρέπει να ελεγχθεί (όταν το x δε βρίσκεται στην S). Στην περίπτωση που η S είναι ταξινομημένη σε μη φθίνουσα διάταξη, τότε η BINARYSEARCH επιστρέφει τη θέση ενός στοιχείου της S που είναι ίσο με το x (ή 0 αν δεν υπάρχει τέτοιο στοιχείο σε $O(\log n)$ χρόνο.

5.1 Αναζήτηση σε ταξινομημένη ακολουθία

Υποθέτουμε ότι η ακολουθία $S = s_1, s_2, ..., s_n$ είναι ταξινομημένη σε μη φθίνουσα σειρά, δηλαδή, $s_1 \leqslant s_2 \leqslant ... \leqslant s_n$. Συνήθως, ένα αρχείο με n εγγραφές, οι οποίες είναι ταξινομημένες με βάση το κλειδί s (πεδίο) είναι αυτό το πρόβλημα που συναντάται στις εφαρμογές. Χάριν απλότητας υποθέτουμε ότι τα s_i είναι διαφορετικά.



Σχήμα 5.1: Δομή εγγραφής σε αρχείο προς αναζήτηση

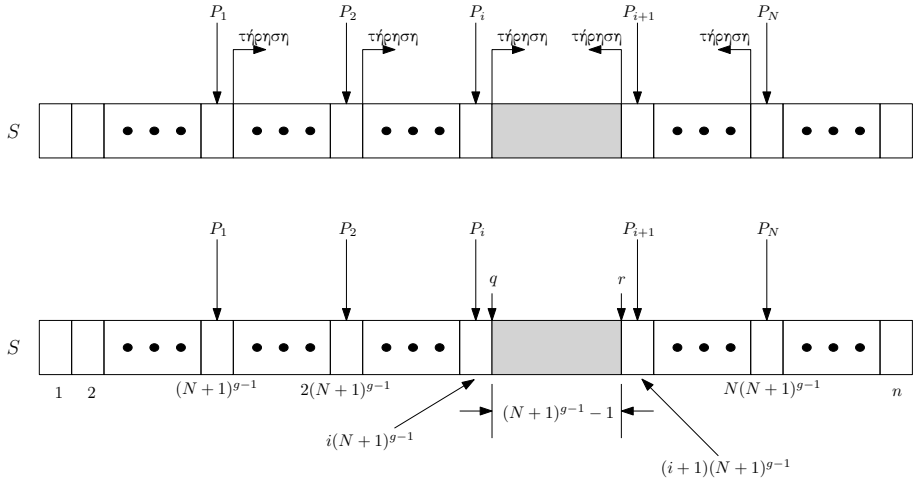
5.1.1 EREW Αναζήτηση

Υποθέτουμε ότι έχουμε N επεξεργαστές σε ένα EREW SM SIMD υπολογιστή για την αναζήτηση ενός δεδομένου στοιχείου του x.
Κατάρχήν η τιμή του x θα πρέπει να γίνει γνωστή σε όλους τους επεξεργαστές. Αυτό μπορεί να γίνει με την BROADCAST σε $O(\log N)$ χρόνο. Στη συνέχεια η S χωρίζεται σε N υποακολουθίες με $\frac{n}{N}$ στοιχεία η κάθε μια και στον P_i επεξεργαστή καταχωρούνται τα στοιχεία $\{s_{(i-1)(\frac{n}{N})+1}, s_{(i-1)(\frac{n}{N})+2}, ..., s_{i(\frac{n}{N})}\}$. Όλοι οι επεξεργαστές εκτελούν την BINARY SEARCH στις υποακολουθίες τους, που συμβαίνει $O(\log(\frac{n}{N}))$ χρόνο στη χειρότερη περίπτωση. Επειδή τα στοιχεία της S είναι όλα διαφορετικά, το πολύ ένας επεξεργαστής θα εντοπίσει ένα s_k ίσο με το x και επιστρέφει το k. Ο συνολικός χρόνος που απαιτείται από αυτόν τον EREW αλγόριθμο αναζήτησης είναι $O(\log N) + O(\log(\frac{n}{N}))$, το οποίο είναι $O(\log n)$. Επειδή ο χρόνος αυτός είναι ίσος με αυτόν της ακολουθιακής BINARY SEARCH δεν επιτυγχάνεται αύξηση της ταχύτητας με τη μέθοδο αυτή!

5.1.2 CREW Αναζήτηση

Ο αλγόριθμος για τον EREW υπολογιστή μπορεί να χρησιμοποιηθεί και στην παρούσα περίπτωση με τη διαφορά ότι τώρα όλοι οι επεξεργαστές

μπορούν να διαβάσουν το x ταυτόχρονα σε σταθερό χρόνο και στη συνέχεια προχωρούν στην εκτέλεση της BINARY SEARCH για τις υπακολουθίες τους. Ο απαιτούμενος χρόνος τώρα είναι $O(\log(\frac{n}{N}))$ στη χειρότερη περίπτωση και είναι μικρότερος από τον ακολουθιακό χρόνο της BINARY SEARCH. Είναι δυνατόν όμως να επιτύχουμε καλύτερο χρόνο με την παραλληλοποίηση της δυαδικής αναζήτησης. Στην παράλληλη αναζήτηση υπάρχουν διαθέσιμοι N επεξεργαστές και συνεπώς χρησιμοποιείται N+1-ιοστή αναζήτηση. Σε κάθε φάση, η ακολουθία χωρίζεται σε N+1 υποακολουθίες με ίσο πλήθος στοιχείων και οι N επεξεργαστές ελέγχουν ταυτόχρονα τα στοιχεία που βρίσκονται στο σύνορο μεταξύ διαδοχικών υποακολουθιών (βλ. Σχήμα 5.2). Κάθε επεξεργαστής συγκρίνει το στοιχείο s του S με το x και:



Σχήμα 5.2: Εξαγωγή του αριθμού σταδίων που απαιτούνται για αναζήτηση ακολουθίας

1. Αν $s > x$, τότε απορρίπτονται όλα τα στοιχεία μεγαλύτερα του s.
2. Διαφορετικά, αν $s < x$, τότε απορρίπτονται όλα τα στοιχεία μικρότερα του s.

Συνεπώς, κάθε επεξεργαστής χωρίζει την ακολουθία σε δύο τμήματα. Το ένα τμήμα περιέχει όλα τα στοιχεία τα οποία απορρίπτονται καθώς σίγουρα δεν περιέχουν ένα στοιχείο ίσο με x και το άλλο τμήμα που περιλαμβάνει εκείνα τα στοιχεία τα οποία πιθανά να περιέχουν ένα στοιχείο ίσο με x. Το πρώτο τμήμα απορρίπτεται ενώ το δεύτερο διατηρείται. Η

διαδικασία αυτή έχει σαν αποτέλεσμα να συρρικνώνεται η αρχική ακολουθία σε μια υποακολουθία που είναι η τομή όλων των τμημάτων που διατηρούνται, δηλαδή η υποακολουθία που βρίσκεται μεταξύ δύο στοιχείων που εξετάζονται σε αυτή τη φάση. Αυτή η υποακολουθία (φαίνεται γραμμοσκιασμένη στο Σχήμα 5.2) είναι εκείνη στην οποία εφαρμόζεται η ίδια διαδικασία στην επόμενη φάση. Η αναζήτηση συνεχίζεται μέχρις ότου είτε ένα στοιχείο βρεθεί ίσο με x ή απορριφθούν όλα τα στοιχεία της S. Επειδή κάθε φάση χρησιμοποιεί μια ακολουθία της οποίας το μήκος είναι $\frac{\text{μήκος προηγούμενης}}{N+1} - 1$, απαιτούνται $O(\log_{N+1}(n+1))$ φάσεις. Στη συνέχεια αναπτύσσουμε πιο αναλυτικά τη μέθοδο.

Έστω g ο μικρότερος ακέραιος τέτοιος ώστε $n \leqslant (N+g)-1$, δηλαδή $g = \lceil \log(n+1)/\log(N+1) \rceil$.
Στη συνέχεια θα δείξουμε με επαγωγή ότι <u>g φάσεις είναι ικανές για την αναζήτηση μιας ακολουθίας μήκους n.</u> Πράγματι, η ανωτέρω πρόταση είναι αληθής για $g = 0$. Υποθέτουμε ότι η πρόταση είναι αληθής για την αναζήτηση μιας ακολουθίας μήκους $(N+1)^{g-1} - 1$. Για την αναζήτηση μιας ακολουθίας μήκους $(N+1)^g - 1$ ο επεξεργαστής $P_i, i = 1, 2, ..., N$ συγκρίνει το x με το s_j, όπου $j = i(N+1)^{g-1}$ (βλ. Σχήμα 5.2). Μετά τη σύγκριση, η ακολουθία για αναζήτηση έχει μήκος $(N+1)^{g-1}-1$ και είναι η γραμμοσκιασμένη, πράγμα που αποδεικνύει την πρόταση. Η υποακολουθία αυτή προσδιορίζεται ως εξής: Κάθε επεξεργαστής P_i χρησιμοποιεί μια μεταβλητή c_i, η οποία λαμβάνει την τιμή `left` ή `right` ανάλογα με το τμήμα της ακολουθίας που αποφασίζει ο P_i να διατηρήσει βρίσκεται στα αριστερά ή δεξιά του στοιχείου που συγκρίθηκε με το x κατά τη διάρκεια αυτής της φάσης. Αρχικά, η τιμή κάθε c_i είναι αυθαίρετη, με σταθερές τις τιμές των $c_0 = \texttt{right}$ και $c_{N+1} = \texttt{left}$. Μετά τη σύγκριση μεταξύ του x και ενός στοιχείου s_{j_i} της S, ο P_i καταχωρεί μια τιμή στο c_i (εκτός εαν $s_{j_i} = x$). Αν $c_i \neq c_{i-1}$ για κάποιο $i, 1 \leqslant i \leqslant N$, τότε η ακολουθία που πρόκειται να αναζητηθεί θα αρχίζει από το s_q και θα τελειώνει στο s_r, όπου:

$$q = (i-1)(N+1)^{g-1} - 1 \quad \text{και} \quad r = i(N+1)^{g-1} - 1$$

Ένας μόνον επεξεργαστής ενημερώνει τα q και r στην κοινή μνήμη και όλοι οι υπόλοιποι επεξεργαστές μπορούν ταυτόχρονα να διαβάσουν τις ενημερωμένες τιμές σε σταθερό χρόνο.

Διαδικασία 32: CREW SEARCH (S, x, k)

Βήμα 1 {Αρχικές τιμές στους δείκτες των ακολουθιών που θα σαρωθούν}
(1.1) $q \leftarrow 1$
(1.2) $r \leftarrow n$

Βήμα 2 {Αρχικές τιμές για τα αποτελέσματα και το μέγιστο αριθμό σταδίων}
(2.1) $k \leftarrow 0$
(2.2) $g \leftarrow \left\lceil \frac{\log(n+1)}{\log(N+1)} \right\rceil$
(2.3) $c_0 = \mathtt{right}$
(2.4) $c_{NH} = \mathtt{left}$

Βήμα 3
όσο $(q \leqslant r$ ***και*** $k = 0)$ **κάνε**
  (3.1) $j_0 \leftarrow q - 1$
  (3.2) **για** $i = 1$ **έως** N **κάνε παράλληλα**
    (i) $j_i \leftarrow (q-1) + i(N+1)^{g-1}$
    {Ο επεξεργαστής P_i συγκρίνει το x με το s_j και προσδιορίζει το μέρος της ακολουθίας που θα διατηρηθεί}
    (ii) **αν** $j_i \leqslant r$ **τότε**
      αν $s_{ji} = x$ **τότε**
        $k \leftarrow j_i$
      αλλιώς
        αν $s_{j_i} > x$ **τότε**
          $c_i \leftarrow \mathtt{left}$
        αλλιώς
          $c_i \leftarrow \mathtt{right}$
        τέλος
      τέλος
    αλλιώς
      (a) $j_i \leftarrow r + 1$
      (b) $c_i \leftarrow \mathtt{left}$
      {Σημαίνει ότι $j_i > r$ δείχνει εκτός της ακολουθίας και γίνεται διορθωτική ενέργεια}
    τέλος
    {υπολογίζονται οι δείκτες της ακολουθίας που θα σαρωθεί στην επόμενη επανάληψη}
    (iii) **αν** $c_i \neq c_{i-1}$ **τότε**
      (a) $q \leftarrow j_{i-1} + 1$
      (b) $r \leftarrow j_i - 1$
    τέλος
    (iv) **αν** $(i = N$ ***και*** $c_i \neq c_{i+1})$ **τότε**
      $q \leftarrow j_i + 1$
    τέλος
  τέλος
  (3.3) $g \leftarrow g - 1$
τέλος

Ανάλυση
Τα βήματα 1, 2, 3.1 και 3.2 εκτελούνται από έναν επεξεργαστή, έστω τον

P_1 σε σταθερό χρόνο. Το βήμα 3.2 εκτελείται επίσης σε σταθερό χρόνο. Όπως αποδείχθηκε, το βήμα 3 εκτελείται το πολύ σε g επαναλήψεις. Συνεπώς η CREW SEARCH εκτελείται σε $O\left(\frac{\log(n+1)}{\log(N+1)}\right)$ χρόνο, δηλαδή $t(n) = O(\log_{N+1}(n+1))$ συνεπώς $c(n) = O(\log_{N+1}(n+1))$ το οποίο δεν είναι βέλτιστο.

Παράδειγμα
Έστω $S = \{1, 4, 6, 9, 10, 11, 13, 14, 15, 18, 20, 23, 32, 45, 51\}$ η ακολουθία,

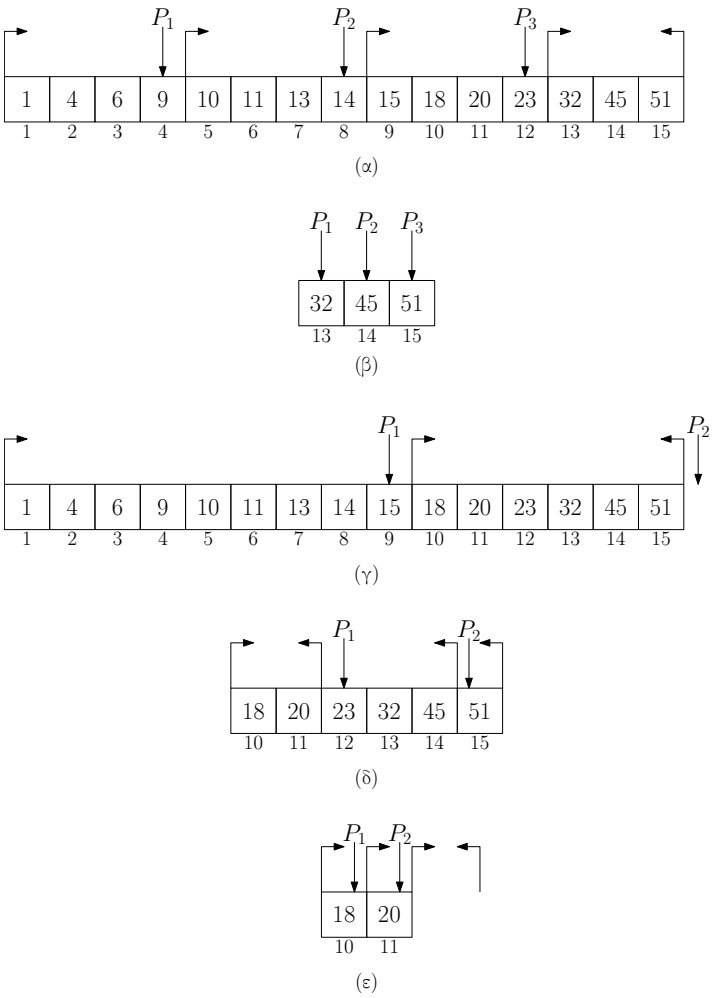
η οποία πρόκειται να αναζητηθεί χρησιμοποιώντας ένα CREW SM SIMD υπολογιστή.

1. Υποθέτουμε ότι $N = 3, x = 45$. Αρχικά $q = 1, r = 15, k = 0$ και $g = 2$. Στην πρώτη επανάληψη του βήματος 3, ο P_1 υπολογίζει $j_1 = 4$ και συγκρίνει το s_4 με το x. Επειδή $9 < 45$, $c_1 = \texttt{right}$. Ταυτόχρονα οι P_2 και P_3 συγκρίνουν τα s_8 και s_{12} με το x, αντίστοιχα. Επειδή $14 < 45$ και $23 < 45$, $c_2 = \texttt{right}$ και $c_3 = \texttt{right}$. Αλλά $c_3 \neq c_4$, συνεπώς $q = 13$ και το r παραμένει αμετάβλητο. Η νέα ακολουθία που θα αναζητηθεί ξεκινά από το s_{13} μέχρι το s_{15} (βλ. Σχήμα 5.3 (α)) με $g = 1$. Στη δεύτερη επανάληψη (Σχήμα 5.3 (β)), ο P_1 υπολογίζει $j_1 = 12 + 1$ και συγκρίνει το s_{13} με το x. Επειδή $32 < 45$, $c_1 = \texttt{right}$. Ταυτόχρονα, ο P_2 συγκρίνει το s_{14} με το x και επειδή είναι ίσα, θέτει $k = 14$. Επίσης, ο P_3 συγκρίνει το s_{15} με το x. Επειδή $51 > 45$, $c_3 = \texttt{left}$. Τώρα, $c_3 \neq c_2$. Συνεπώς, $q = 12 + 2 + 1 = 15$ και $r = 12 + 3 - 1 = 14$ και η διαδικασία σταματά με $k = 14$.

2. Έστω ότι $x = q$ με $N = 3$. Στην πρώτη επανάληψη, ο P_1 συγκρίνει το s_4 με το x και επειδή είναι ίσα, θέτει $k = 4$.

3. Έστω ότι $N = 2$ και $x = 21$. Αρχικά, $g = 3$. Στην πρώτη επανάληψη ο P_1 υπολογίζει $j_1 = 9$ και συγκρίνει το s_9 με το x, επειδή $15 < 21, c_1 = \texttt{right}$. Ταυτόχρονα ο P_2 υπολογίζει $j_2 = 16$ και $c_2 = \texttt{left}$. Τώρα, $c_2 \neq c_1$, συνεπώς $q = 10$ και $r = 15$ έτσι η ακολουθία που θα αναζητηθεί θα είναι από s_{10} μέχρι s_{15} και $g = 2$ (βλ. Σχήμα 5.3 (γ)).
   Στη δεύτερη επανάληψη, ο P_1 υπολογίζει $j_1 = q + 3$ και συγκρίνει το s_{12} με το x, επειδή $51 > 21, c_2 = \texttt{left}$. Τώρα $c_1 \neq c_0$ και συνεπώς r_{11} και το q παραμένει αμετάβλητο (βλ. Σχήμα 5.3 (δ)).

Στην τελική επανάληψη, $q = 1$ και ο P_1 υπολογίζει $j_1 = q + 1$, συγκρίνει το s_{10} με το x. Επειδή $18 < 21$, $c_1 = \texttt{right}$. Ταυτόχρονα, ο P_2 υπολογίζει $j_2 = q + 2$ και συγκρίνει το s_{11} με το x, επειδή $20 < 21, c_2 = \texttt{right}$. Τώρα, $c_2 \neq c_3$ και $q = 12$. Επειδή $q > r$ η διαδικασία σταματά ανεπιτυχώς και επιστρέφει $k = 0$.



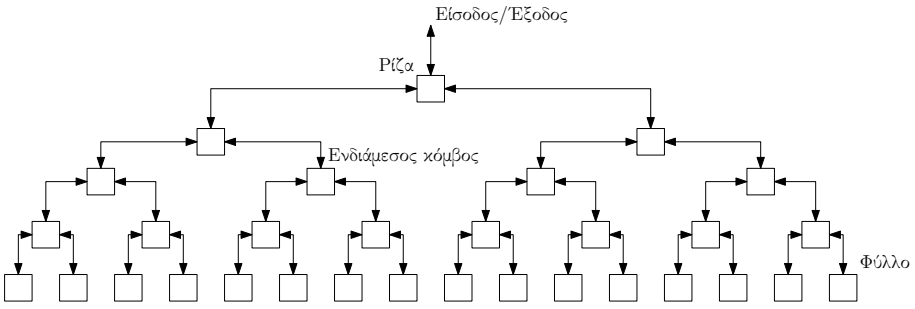
Σχήμα 5.3: Ακολουθία αναζήτησης 18 στοιχείων με τη διαδικασία CREW SEARCH

Διαδικασία 33: SM SEARCH (S, x, k)

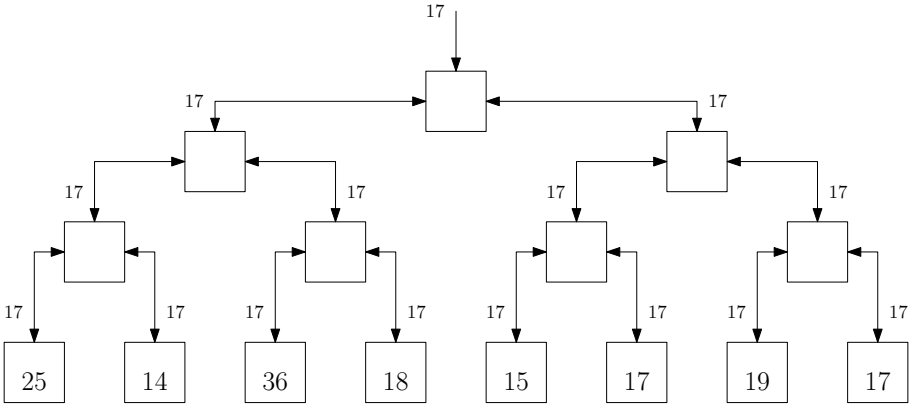
Βήμα 1
για $i = 1$ **έως** N **κάνε παράλληλα**
| διάβασε το x
τέλος

Βήμα 2
για $i = 1$ **έως** N **κάνε παράλληλα**
| (2.1) $S_i \leftarrow \{s_{(i-1)(\frac{n}{N})+1}, s_{(i-1)(\frac{n}{N})+2}, ..., s_{i(\frac{n}{N})}\}$
| (2.2) SEQUENTIAL SEARCH(S_i, x, k_i)
τέλος

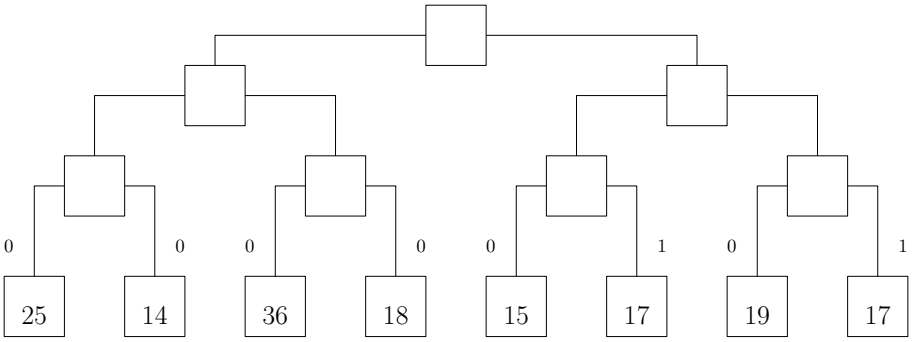
Βήμα 3
για $i = 1$ **έως** N **κάνε παράλληλα**
| **αν** $k_i > 0$ **τότε**
| | $k \leftarrow k_i$
| **τέλος**
τέλος



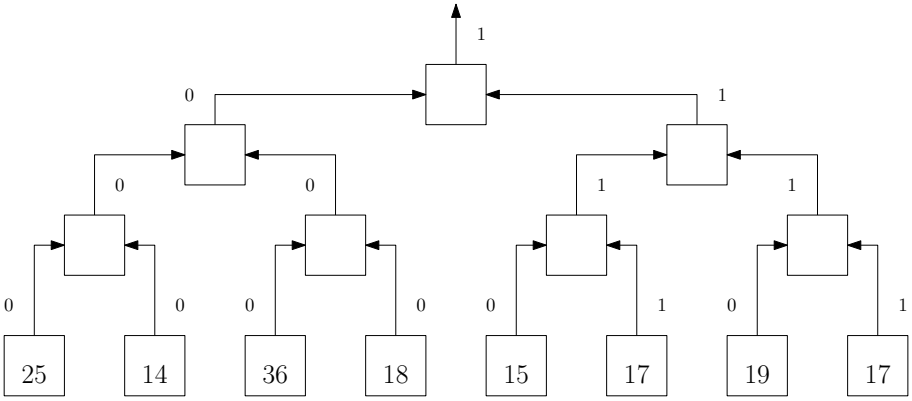
Σχήμα 5.4: Υπολογιστής προς αναζήτηση, συνδεδεμένος σε δέντρο



(α) Στάδιο 1

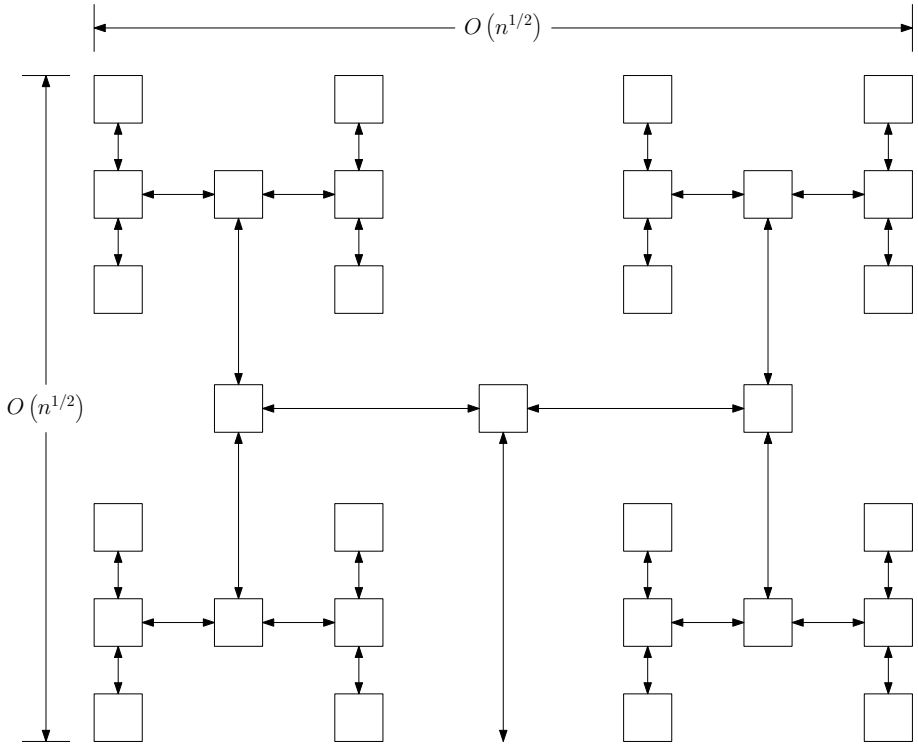


(β) Στάδιο 2

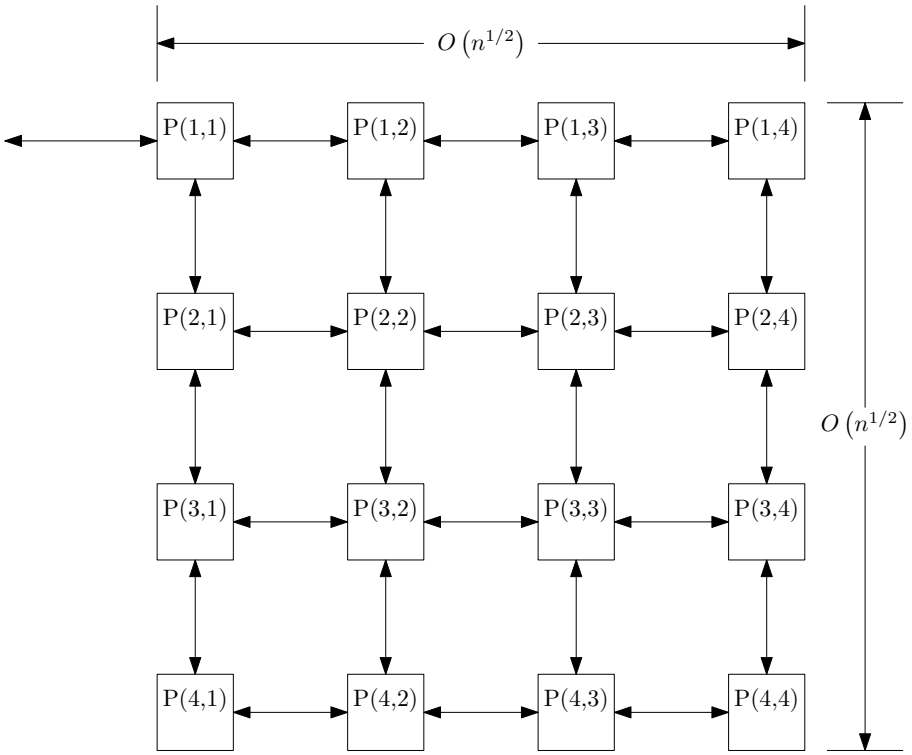


(γ) Στάδιο 3

Σχήμα 5.5: Ακολουθία αναζήτησης οκτώ στοιχείων με χρήση δέντρου



Σχήμα 5.6: Υπολογιστής συνδεδεμένος σε δέντρο ενσωματωμένος σε πλέγμα



Σχήμα 5.7: Υπολογιστής προς αναζήτηση, συνδεδεμένος σε πλέγμα

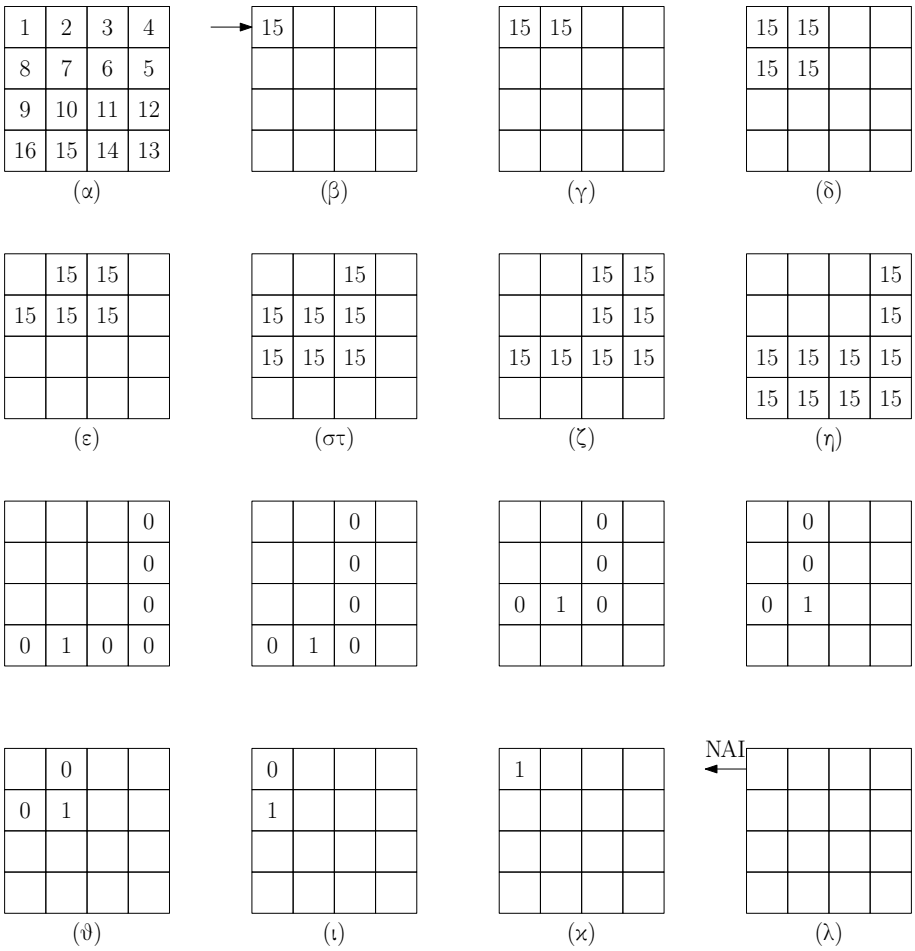
Διαδικασία 34: MESH SEARCH $(S, x, \text{απάντηση})$

Βήμα 1 { Ο $P(1,1)$ διαβάζει τα δεδομένα}
αν $x = s_{1,1}$ **τότε**
| $b_{1,1} \leftarrow 1$
αλλιώς
| $b_{1,1} \leftarrow 0$
τέλος

Βήμα 2 { ξεδίπλωμα}
για $i = 1$ **έως** $n^{\frac{1}{2}} - 1$ **κάνε**
| (2.1) **για** $j = 1$ **έως** i **κάνε παράλληλα**
| | (i) Ο $P(j,i)$ μεταδίδει το $(b_{j,i}, x)$ στο $P(j, i+1)$
| | (ii) **αν** $(x = s_{j,i+1}$ ***ή*** $b_{j,i} = 1)$ **τότε**
| | | $b_{j,i+1} \leftarrow 1$
| | **αλλιώς**
| | | $b_{j,i+1} \leftarrow 0$
| | **τέλος**
| **τέλος** 2.2
| **για** $j = 1$ **έως** $i + 1$ **κάνε παράλληλα**
| | (i) Ο $P(i,j)$ μεταδίδει το $(b_{i,j}, x)$ στο $P(i+1, j)$
| | (ii) **αν** $(x = s_{i+1,j}$ ***ή*** $b_{i,j} = 1)$ **τότε**
| | | $b_{i+1,j} \leftarrow 1$
| | **αλλιώς**
| | | $b_{i+1,j} \leftarrow 0$
| | **τέλος**
| **τέλος**
τέλος

Βήμα 3 {δίπλωμα}
για $i = n^{\frac{1}{2}}$ **έως** 2 **κάνε**
| (3.1) **για** $j = 1$ **έως** i **κάνε παράλληλα**
| | $P(j, i)$ μεταδίδει το $b_{j,i}$ στο $P(j, i-1)$
| **τέλος**
| (3.2) **για** $j = 1$ **έως** $i-1$ **κάνε παράλληλα**
| | $P(j, i)$ μεταδίδει το $b_{j,i}$ στο $P(j, i-1)$
| **τέλος**
| (3.3) **αν** $(b_{i,i-1} = 1$ ***ή*** $b_{i,i} = 1)$ **τότε**
| | $b_{i,i-1} \leftarrow 1$
| **αλλιώς**
| | $b_{i,i-1} \leftarrow 0$
| **τέλος**
| (3.4) **για** $j = 1$ **έως** $i-1$ **κάνε παράλληλα**
| | Ο $P(i, j)$ μεταδίδει το $b_{i,j}$ στο $P(i-1, j)$
| **τέλος**
| (3.5) **για** $j = 1$ **έως** $i-2$ **κάνε παράλληλα**
| | $b_{i-1,j} \leftarrow b_{i,j}$
| **τέλος**
| (3.6) **αν** $(b_{i-1,i-1} = 1$ **έως** $b_{i,i-1} = 1)$ **τότε**
| | $b_{i-1,i-1} \leftarrow 1$
| **αλλιώς**
| | $b_{i-1,i-1} \leftarrow 0$
| **τέλος**
τέλος

Βήμα 4 { Ο $P(1, 1)$ δημιουργεί τα αποτελέσματα}
αν $b_{1,1} = 1$ **τότε**
| απάντηση $\leftarrow$ ναι
αλλιώς
| απάντηση $\leftarrow$ όχι
τέλος



Σχήμα 5.8: Ακολουθία αναζήτησης 16 στοιχείων με τη διαδικασία MESH SEARCH

Κεφάλαιο 6

ΑΛΓΟΡΙΘΜΟΙ ΓΡΑΦΗΜΑΤΩΝ (Graph Algorithms)

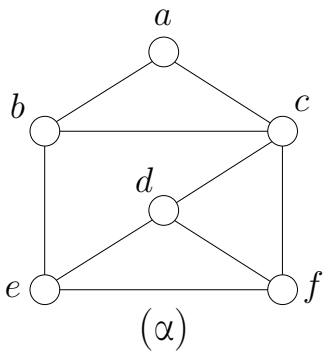
6.1 Εισαγωγή

Σε κάθε περιοχή της επιστήμης των υπολογιστών οι γράφοι χρησιμοποιούνται για να οργανώσουν δεδομένα, να μοντελοποιήσουν αλγορίθμους και γενικά είναι ένα ισχυρό εργαλείο για την αναπαράσταση υπολογιστικών εννοιών. Ιδιαίτερα, τα δέντρα, εμφανίζονται παντού. Πολλοί κλάδοι της μηχανολογίας και των επιστημών βασίζονται στους γράφους για την αναπαράσταση μιας μεγάλης ποικιλίας αντικειμένων από ηλεκτρονικά κυκλώματα, χημικές ενώσεις και κρυστάλλους γενετικών διαδικασιών, κοινωνικές δομές και οικονομικά συστήματα. Το ίδιο ισχύει για κάθε ερευνητική δραστηριότητα όπου οι γράφοι παίζουν σημαντικό ρόλο στη μοντελοποίηση και επίλυση πολυάριθμων προβλημάτων βελτιστοποίησης όπως είναι τα προβλήματα χρονοδρομολόγησης, δεομολόγησης, μεταφοράς και ροής δικτύου. Επομένως είναι πολύ σημαντικό για αυτές τις εφαρμογές να αναπτυχθούν αποτελεσματικοί αλγόριθμοι γράφων.
Κατά συνέπεια, ένα μεγάλο μέρος της υπάρχουσας λογοτεχνίας αφορά υπολογιστικά προβλήματα θεωρίας γράφων και λύσεις τους. Το παρόν κεφάλαιο αφορά τους παράλληλους αλγόριθμους γράφων.

6.2 Ορισμοί

Ένας γράφος αποτελείται από ένα πεπερασμένο σύνολο κόμβων και ένα πεπερασμένο σύνολο ακμών που συνδέουν ζεύγη κόμβων.

Στο σχήμα 6.1(α) φαίνεται ένας γράφος 6 κόμβων και 9 ακμών.

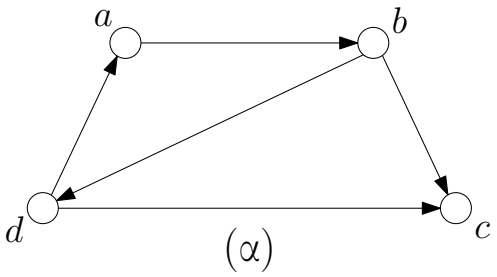


$$
\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array}
\begin{array}{c}
\begin{array}{cccccc} a & b & c & d & e & f \end{array} \\
\left[\begin{array}{cccccc}
0 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 & 0
\end{array}\right] \\
(\beta)
\end{array}
$$

Σχήμα 6.1: Γράφος με 6 κόμβους και ο πίνακας γειτνίασής του

Εδώ οι κόμβοι (καλούνται επίσης κορυφές) ονομάζονται a, b, c, d, e και f. Οι ακμές είναι (a, b), (a, c), (b, c), (b, e), (c, d), (c, f), (d, c), (d, f) και (e, f).

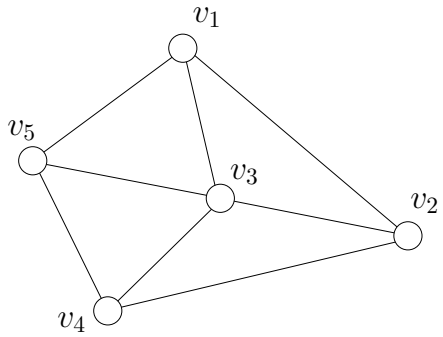
Ένας γράφος είναι κατευθυνόμενος όταν οι ακμές (καλούνται επίσης τόξα) έχουν κάποιο προσανατολισμό, παρέχοντας έτσι μια μονόδρομη σύνδεση όπως φαίνεται από τις κεφαλές κάθε βέλους στο Σχήμα 6.2(α).



$$
\begin{array}{c} \\ a \\ b \\ c \\ d \end{array}
\begin{array}{c}
\begin{array}{cccc} a & b & c & d \end{array} \\
\left[\begin{array}{cccc}
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1
\end{array}\right] \\
(\beta)
\end{array}
$$

Σχήμα 6.2: Κατευθυνόμενος γράφος και ο πίνακας γειτνίασής του

Εδώ, ο κόμβος a συνδέεται με τον κόμβο b, ο κόμβος b συνδέεται με τον κόμβο c και τον d και ο κόμβος d συνδέεται με τον c. Ο συμβολισμός $Q(V, E)$ χρησιμοποιείται για να αναπαραστήσει ένα γράφο G του οποίου το σύνολο κορυφών συμβολίζεται με V και το σύνολο των ακμών με E.
Η αναπαράσταση με πίνακα μπορεί να χρησιμοποιηθεί για αποθήκευση



(α)

	v_1	v_2	v_3	v_4	v_5
v_1	0	6	0	8	1
v_2	6	0	5	3	0
v_3	0	5	0	4	1
v_4	8	5	4	0	3
v_5	1	0	7	9	0

(β)

Σχήμα 6.3: Βεβαρυμένος γράφος και ο πίνακας βαρών

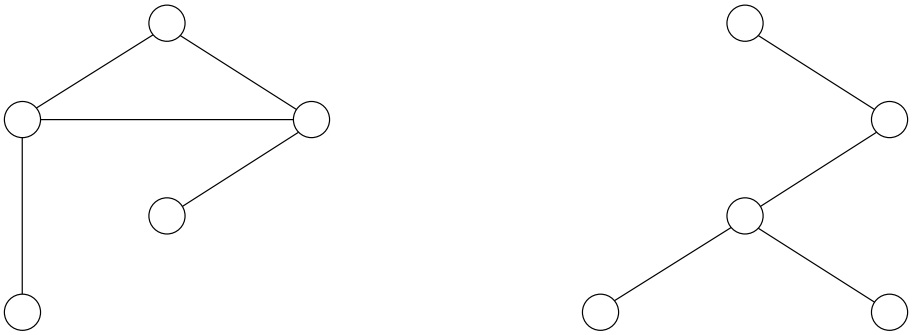
υπολογισμών και χειρισμό του γράφου. Έστω G ο γράφος του οποίου το σύνολο κορυφών είναι $V = v_0, v_1, v_2, \ldots, v_{n-1}$. Ο γράφος αυτός μπορεί να αναπαρασταθεί μοναδικά με έναν $n \times n$ πίνακα γειτνίασης A, του οποίου τα στοιχεία $a_{ij},\ 0 \leqslant i \leqslant j \leqslant n-1$ ορίζονται ως εξής:

$$a_{ij} = \begin{cases} 1, & \text{αν} \quad v_i \text{ συνδέεται με το} \quad v_j \\ 0, & \text{διαφορετικά} \end{cases}$$

Οι πίνακες γειτνίασης των γράφων στα σχήματα 6.1(α) και 6.2(α), φαίνονται στα σχήματα 6.1(β) και 6.2(β), αντίστοιχα, όπου $v_0 = a$, $v_1 = b$, κ.λπ.

Να σημειωθεί ότι τη στιγμή που ο γράφος του Σχήματος 6.1(α) είναι μη κατευθυνόμενος, ο πίνακας του σχήματος 6.1(β) είναι συμμετρικός.
Όταν κάθε ακμή του γράφου σχετίζεται με ένα πραγματικό αριθμό, ο οποίος λέγεται βάρος, ο γράφος λέγεται βεβαρυμένος. Ένας βεβαρυμένος γράφος μπορεί να είναι κατευθυνόμενος ή μη κατευθυνόμενος. Το Σχήμα 6.3(α) δείχνει έναν μη κατευθυνόμενο βεβαρυμένο γράφο.
Η έννοια του βάρους ποικίλει από τη μία εφαρμογή στην άλλη, μπορεί να αναπαριστά απόσταση, κόστος, χρόνο, πιθανότητα κ.λπ. Ένας πίνακας βαρών W χρησιμοποιείται για την αναπαράσταση του βεβαρυμένου γράφου, όπως φαίνεται στο Σχήμα 6.3(α). Εδώ κάθε στοιχείο εισόδου w_{ij} του W αναπαριστά το βάρος της ακμής (v_i, v_j). Αν τα v_i και v_j δεν είναι συνδεδεμένα με μία ακμή, τότε το w_{ij} μπορεί να είναι ίσο με το μηδέν ή το άπειρο ή οποιαδήποτε κατάλληλη τιμή, ανάλογα με την εφαρμογή.

Ένα μονοπάτι από την αρχική κορυφή v_i στην κορυφή προορισμού v_j στο γράφο $G(V, E)$ είναι μια ακολουθία ακμών (v_i, v_k), $(v_k, v_e), \ldots,$ $(v_m,$



Σχήμα 6.4: Δύο υπογράφοι του γράφου του Σχήματος 6.1

v_j) από το E, όπου καμμία κορυφή δεν εμφανίζεται περισσότερο από μία φορά. Στο Σχήμα 6.1, για παράδειγμα, (a, c), (c, d), (d, e) είναι ένα μονοπάτι από το a στο e. Ένας κύκλος είναι ένα μονοπάτι του οποίου η αρχή και ο προορισμός είναι ίδια. Η ακολουθία (a, b), (b, d), (d, a) του Σχήματος 6.2 σχηματίζει ένα κύκλο. Σε ένα βεβαρυμένο γράφο, το μήκος του μονοπατιού ή του κύκλου είναι ίσο με τον αριθμό των ακμών που το αποτελούν.

Ένας υπογράφος $G'(V', E')$ του γράφου $G(V, E)$ είναι ένας γράφος τέτοιος ώστε $V' \subseteq V$ και $E' \subseteq E$, δηλαδή ένας γράφος του οποίου οι κορυφές και οι ακμές είναι στο G. Στο Σχήμα 6.4 φαίνονται δύο υπογράφοι του γράφου του Σχήματος 6.1.

6.3 Υπολογίζοντας τον πίνακα συνεκτικότητας

Ο πίνακας συνεκτικότητας ενός γράφου G n-κόμβων είναι ένας $n \times n$ πίνακας C του οποίου τα στοιχεία ορίζονται ως εξής:

$$C_{ij} = \begin{cases} 1, & \text{αν υπάρχει μονοπάτι μήκους 0 ή μετακίνηση} \\ & \text{από τον } v_j \text{ στον } v_k \\ 0, & \text{διαφορετικά} \end{cases}$$

για $j, k = 0, 1, \ldots, n-1$. Να σημειωθεί ότι ένα μονοπάτι μήκους 0 αρχίζει και τελειώνει σε μία κορυφή, χωρίς να χρησιμοποιηθούν ακμές, ενώ ένα μονοπάτι μήκους 1 αποτελείται από μία ακμή. Ο πίνακας C είναι επίσης γνωστός σαν ανακλαστική και μεταβατική κλειστότητα του G.

Όταν δίνεται ο πίνακας γειτνίασης A του γράφου G, απαιτείται να υπολογιστεί ο C. Η προσέγγιση που θα χρησιμοποιήσουμε κάνει χρήση

πολλαπλασιασμού Boolean πινάκων, ο οποίος διαφέρει από τον κανονικό πολλαπλασιασμό πινάκων στο ότι:

(i) Οι πίνακες που πρόκειται να πολλαπλασιαστούν καθώς και το γινόμενο των πινάκων είναι όλα δυαδικοί, δηλαδή κάθε ένα από τα στοιχεία τους είναι 0 ή 1

(ii) Ο τελεστής Boolean (ή αλλιώς ο λογικός τελεστής) and αντικαθιστά τον κανονικό πολλαπλασιασμό, δηλαδή

$$0 \textbf{ and } 0 = 0$$
$$0 \textbf{ and } 1 = 0$$
$$1 \textbf{ and } 0 = 0$$
$$1 \textbf{ and } 1 = 1$$

(iii) Ο τελεστής Boolean (ή αλλιώς ο λογικός τελεστής) or αντικαθιστά την κανονική πρόσθεση, δηλαδή

$$0 \textbf{ or } 0 = 0$$
$$0 \textbf{ or } 1 = 1$$
$$1 \textbf{ or } 0 = 1$$
$$1 \textbf{ or } 1 = 1$$

Επομένως, αν X, Y και Z είναι οι $n \times n$ Boolean πίνακες, όπου Z είναι το Boolean ηινόμενο των X και Y, τότε κάθε στοιχείο του πίνακα Z υπολογίζεται από το:

$$z_{ij} = (x_{i1} \textbf{ and } y_{1j}) \textbf{ or } (x_{i2} \textbf{ and } y_{2j}) \textbf{ or } \ldots \textbf{ or } (x_{in} \textbf{ and } y_{nj}),$$
$$\text{για } i, j = 0, 1, \ldots, n-1$$

Το πρώτο βήμα για τον υπολογισμό του πίνακα συνεκτικότητας C, είναι να πάρουμε τον $n \times n$ πίνακα B από τον A ως εξής:

$$b_{jk} = a_{jk} \text{ (για } j \neq k) \quad \text{και } b_{jj} = 1,\ j, k = 0, 1, \ldots, n-1$$

Ο πίνακας B επομένως αναπαριστά όλα τα μονοπάτια στο G μήκους μικρότερου του 2, δηλαδή:

$$b_{jk} = \begin{cases} 1, & \text{αν υπάρχει μονοπάτι μήκους 0 ή 1} \\ & \text{από το } v_j \text{ στο } v_k \\ 0, & \text{διαφορετικά} \end{cases}$$

Όμοια, ο B^2 (για παράδειγμα το Boolean γινόμενο του B με τον εαυτό του) αναπαριστά μονοπάτια μήκους μικρότερου ή ίσου του n.

Παρατηρούμε τώρα ότι αν υπάρχει ένα μονοπάτι από το v_i στο v_j, αυτό δεν μπορεί να έχει μήκος μεγαλύτερο από $n-1$. Κατά συνέπεια, $C = B^{n-1}$, δηλαδή ο πίνακας συνεκτικότητας λαμβάνεται έπειτα από $\lceil \log n - 1 \rceil$ πολλαπλασιασμούς Boolean πινάκων.
Να σημειωθεί ότι όταν το $n-1$ δεν είναι δύναμη του 2, ο C λαμβάνεται από τον B^m, όπου $m = 2^{\lceil \log n - 1 \rceil}$. Αυτό είναι σωστό, αφού $B^m = B^{n-1}$ για $m > n-1$.

Προκειμένου να υλοποιήσουμε αυτόν τον αλγόριθμο παράλληλα, κάνουμε χρήση οποιουδήποτε αλγόριθμου πολλαπλασιασμού πινάκων, προσαρμοσμένου στο να εκτελεί πολλαπλασιασμό Boolean πινάκων.
Ειδικότερα, η συνάρτηση CUBE MATRIX MULTIPLICATION μπορεί να χρησιμοποιηθεί. Ο αλγόριθμος ο οποίος προκύπτει μας δίνει τη συνάρτηση CUBE CONNECTIVITY. Η συνάρτηση αυτή δέχεται σαν είσοδο τον πίνακα γειτνίασης A και επιστρέφει σαν έξοδο τον πίνακα συνεκτικότητας C. Εκτελείται σε έναν cube-connected SIMD υπολογιστή με $N = n^3$ επεξεργαστές $P_1, P_2, \ldots, P_N$. Οι επεξεργαστές αυτοί μπορεί να θεωρηθεί ότι έχουν τακτοποιηθεί σ' ένα $n \times n \times n$ πρότυπο πίνακα. Σε αυτόν τον πίνακα, ο P_r κατέχει τη θέση (i, j, k), όπου $r = in^2 + jn + k$ και $0 \leqslant i, j, k \leqslant n-1$. Έχει τρεις καταχωρητές $A(i, j, k)$, $B(i, j, k)$ και $C(i, j, k)$. Αρχικά, οι επεξεργαστές στις θέσεις $(0, j, k)$, $0 \leqslant j, k \leqslant n-1$, περιέχουν τον πίνακα γειτνίασης, δηλαδή $A(0, j, k) = a_{jk}$. Στο τέλος των υπολογισμών αυτοί οι επεξεργαστές περιέχουν τον πίνακα συνεκτικότητας, δηλαδή $C(0, j, k) = c_{jk}$, $0 \leqslant j, k \leqslant n-1$.

Διαδικασία 35: CUBE CONNECTIVITY (A, C)

Βήμα 1

{Τα διαγώνια στοιχεία του πίνακα γειτνίασης γίνονται ίσα με 1}

για $j = 0$ **έως** $n - 1$ **κάνε παράλληλα**
  $A(0, j, j) \leftarrow 1$
τέλος

Βήμα 2

{Οι καταχωρητές A αντιγράφονται στους καταχωρητές B}

για $j = 0$ **έως** $n - 1$ **κάνε παράλληλα**
  για $k = 0$ **έως** $n - 1$ **κάνε παράλληλα**
    $B(0, j, k) \leftarrow A(0, j, k)$
  τέλος
τέλος

Βήμα 3

{Ο πίνακας συνεκτικότητας λαμβάνεται μέσω επαναλαμβανόμενων πολλαπλασιασμών Boolean}

για $i = 1$ **έως** $\lceil \log n - 1 \rceil$ **κάνε**
  (3.1) CUBE MATRIX MULTIPLICATION (A, B, C)
  (3.2)
  για $j = 0$ **έως** $n - 1$ **κάνε παράλληλα**
    για $k = 0$ **έως** $n - 1$ **κάνε παράλληλα**
      (ι) $A(0, j, k) \leftarrow C(0, j, k)$
      (ιι) $B(0, j, k) \leftarrow C(0, j, k)$
    τέλος
  τέλος
τέλος

Ανάλυση

Τα βήματα 1, 2 και 3.2 απαιτούν σταθερό χρόνο. Στο βήμα 3.1, η συνάρτηση CUBE MATRIX MULTIPLICATION απαιτεί $\mathcal{O}(\log n)$ χρόνο. Το βήμα αυτό επαναλαμβάνεται $\log n$ φορές. Προκύπτει ότι ο συνολικός χρόνος εκτέλεσης αυτής της συνάρτησης είναι $t(n) = \mathcal{O}(\log^2 n)$.
Αφού $p(n) = n^3$,

$$c(n) = \mathcal{O}(n^3 \log^2 n)$$

Παράδειγμα

Θεωρούμε τον πίνακα γειτνίασης του Σχήματος 6.2(β). Μετά την εφαρμογή των βημάτων 1 και 2 της συνάρτησης CUBE CONNECTIVITY έχει υπολογιστεί ο

$$B = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Η πρώτη επανάληψη του βήματος 3 παράγει τον

$$B^2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Ενώ η δεύτερη επανάληψη παράγει τον $B^4 = B^2$.

Διαδικασία 36: CUBE MATRIX MULTIPLICATION (A, B, C)

Βήμα 1
για $m = 3q - 1$ **έως** $2q$ **κάνε**
  για *όλα τα* r *στο* $\{N, r_m = 0\}$ **κάνε παράλληλα**
    (1.1) $A_r(m) = A_r$
    (1.2) $B_r(m) = B_r$
  τέλος
τέλος

Βήμα 2
για $m = q - 1$ **έως** $n - 1$ **κάνε**
  για *όλα τα* r *στο* $\{N, r_m = r_{2q+m}\}$ **κάνε παράλληλα**
    $A_r(m) \leftarrow A_r$
  τέλος
τέλος

Βήμα 3
για $m = 2q - 1$ **έως** q **κάνε**
  για *όλα τα* r *στο* $\{N, r_m = r_{q+m}\}$ **κάνε παράλληλα**
    $B_r(m) \leftarrow B_r$
  τέλος
τέλος

Βήμα 4
για $r = 1$ **έως** N **κάνε παράλληλα**
  $C_r \leftarrow A_r \times B_r$
τέλος

Βήμα 5
για $m = 2q$ **έως** $3q - 1$ **κάνε**
  για $r = 1$ **έως** N **κάνε παράλληλα**
    $C_r \leftarrow C_r + C_r(m)$
  τέλος
τέλος

6.4 Εύρεση συνεκτικών συνιστωσών

Ένας μη κατευθυνόμενος γράφος λέγεται συνεκτικός αν για κάθε ζευγάρι v_i και v_j των κορυφών του υπάρχει ένα μονοπάτι από το v_i στο v_j.
Μια συνεκτική συνιστώσα του γράφου G είναι ένας υπογράφος G' του G, ο οποίος είναι συνεκτικός.
Το πρόβλημα που θεωρούμε για αυτή την ενότητα είναι το ακόλουθο. Δίνεται ένας μη κατευθυνομενος $n-$κόμβων γράφος G με τον πίνακα γειτνίασής του και ζητείται να αναλυθεί ο G στο μικρότερο δυνατό αριθμό συνεκτικών συνιστωσών.
Μπορούμε να λύσουμε το πρόβλημα υπολογίζοντας πρώτα τον πίνακα συνεκτικότητας C του G. Χρησιμοποιώντας τον C, μπορούμε τώρα να κατασκευάσουμε έναν $n \times n$ πίνακα D, του οποίου τα στοιχεία ορίζονται από

$$d_{jk} = \begin{cases} v_k, & \text{αν } c_{jk} = 1 \\ 0, & \text{διαφορετικά} \end{cases}$$

για $0 \leqslant j, k \leqslant n-1$. Με άλλα λόγια, η γραμμή j του D περιέχει τα ονόματα των κορυφών στις οποίες συνδέεται ο v_j με ένα μονοπάτι, δηλαδή τις κορυφές της ίδιας συνεκτικής συνιστώσας του v_j. Τελικά ο γράφος μπορεί να χωριστεί στον μικρότερο δυνατό αριθμό συνεκτικών συνιστωσών εκχωρώντας κάθε κορυφή σε μια συνιστώσα ως εξής:
Ο v_j εκχωρείται στη συνιστώσα l, αν η l έχει το μικρότερο δείκτη για τον οποίο $d_{ij} \neq 0$.
Η παράλληλη υλοποίηση αυτής της προσέγγισης κάνει χρήση της συνάρτησης CUBE CONNECTIVITY, που αναπτύχθηκε στην προηγούμενη ενότητα, προκειμένου να υπολογίσει τον πίνακα συνεκτικότητας C. Ο αλγόριθμος αυτός δίνεται με τη συνάρτηση CUBE COMPONENTS. Η συνάρτηση αυτήτ τρέχει σε έναν cube-connected SIMD υπολογιστή με $N = n^3$ επεξεργαστές καθένας με 3 καταχωρητές A, B και C. Οι επεξεργαστές είναι διατεταγμένοι σε ένα $n \times n \times n$ πρότυπο πίνακα όπως εξηγήσαμε νωρίτερα. Αρχικά $A(0, j, k) = a_{jk}$ για $0 \leqslant j, k \leqslant n-1$, δηλαδή οι επεξεργαστές στις θέσεις $(0, j, k)$ περιέχουν τον πίνακα γειτνίασης του G. Όταν η συνάρτηση σταματήσει, ο $C(0, j, 0)$ περιέχει τον αριθμό της συνιστώσας για την κορυφή v_j, όπου $j = 0, 1, \ldots, n-1$.

Διαδικασία 37: CUBE COMPONENTS (A, C)

Βήμα 1
{Υπολογισμός του πίνακα συνεκτικότητας}
CUBE CONNECTIVITY (A,C)

Βήμα 2
{Κατασκευή του πίνακα D}
για $j = 0$ **έως** $n - 1$ **κάνε παράλληλα**
  για $k = 0$ **έως** $n - 1$ **κάνε παράλληλα**
    αν $C(0, j, k) = 1$ **τότε**
      $C(0, j, k) = \nu_k$
    τέλος
  τέλος
τέλος

Βήμα 3
{Προσδιορισμός μιας συνεκτικής συνιστώσας για κάθε κορυφή}
για $j = 0$ **έως** $n - 1$ **κάνε παράλληλα**
  (3.1) Οι n επεξεργαστές στη γραμμή j (σχηματίζοντας έναν $\log n -$ διάστασης κύβο) βρίσκουν το μικρότερο l για το οποίο $C(0, j, l) \neq 0$
  (3.2) $C(0, j, 0) \leftarrow l$
τέλος

Ανάλυση

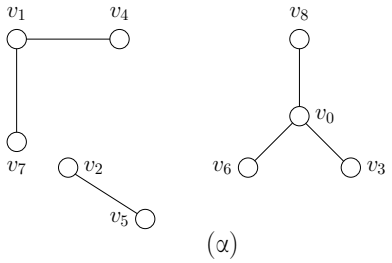
Όπως δείξαμε στην προηγούμενη ενότητα, το βήμα 1 απαιτεί $\mathcal{O}(\log^2 n)$ χρόνο. Τα βήματα 2 και 3.2 απαιτούν σταθερό χρόνο. Γνωρίζουμε ότι το βήμα 1 απαιτεί $\mathcal{O}(\log n)$ χρόνο. Ο συνολικός χρόνος εκτέλεσης της συνάρτησης CUBE COMPONENTS είναι $t(n) = \mathcal{O}(\log^2 n)$.
Αφού $p(n) = n^3$,

$$c(n) = \mathcal{O}(n^3 \log^2 n)$$

Παράδειγμα

Θεωρούμε το γράφο του Σχήματος 6.5(α) του οποίου οι πίνακες γειτνίασης και συνεκτικότητας δίνονται από τα Σχήματα 6.5(β) και (γ) αντίστοιχα. Ο πίνακας D φαίνεται στο Σχήμα 6.5(δ). Επομένως, η εκχώρηση σε κάθε συνιστώσα έχει ως εξής:

Συνιστώσα 0: $\nu_0, \nu_3, \nu_6, \nu_8$
Συνιστώσα 1: ν_1, ν_4, ν_7
Συνιστώσα 2: ν_2, ν_5



(α)

(β)

	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
v_0	0	0	0	1	0	0	1	0	1
v_1	0	0	0	0	1	0	0	1	0
v_2	0	0	0	0	0	1	0	0	0
v_3	1	0	0	0	0	0	0	0	0
v_4	0	1	0	0	0	0	0	0	0
v_5	0	0	1	0	0	0	0	0	0
v_6	1	0	0	0	0	0	0	0	0
v_7	0	1	0	0	0	0	0	0	0
v_8	1	0	0	0	0	0	0	0	0

(γ)

	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
v_0	1	0	0	1	0	0	1	0	1
v_1	0	1	0	0	1	0	0	1	0
v_2	0	0	1	0	0	1	0	0	0
v_3	1	0	0	1	0	0	1	0	1
v_4	0	1	0	0	1	0	0	1	0
v_5	0	0	1	0	0	1	0	0	0
v_6	1	0	0	1	0	0	1	0	1
v_7	0	1	0	0	1	0	0	1	0
v_8	1	0	0	1	0	0	1	0	1

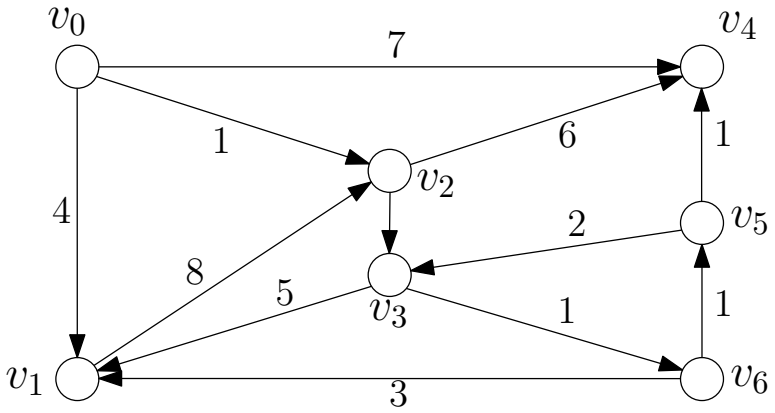
(δ)

	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
v_0	v_0	0	0	v_3	0	0	v_6	0	v_8
v_1	0	v_1	0	0	v_4	0	0	v_7	0
v_2	0	0	v_2	0	0	v_5	0	0	0
v_3	v_0	0	0	v_3	0	0	v_6	0	v_8
v_4	0	v_1	0	0	v_4	0	0	v_7	0
v_5	0	0	v_2	0	0	v_5	0	0	0
v_6	v_0	0	0	v_3	0	0	v_6	0	v_8
v_7	0	v_1	0	0	v_4	0	0	v_7	0
v_8	v_0	0	0	v_3	0	0	v_6	0	v_8

Σχήμα 6.5: Υπολογισμός συνεκτικών συνιστωσών γράφου

6.5 Εύρεση Ελάχιστων Μονοπατιών για όλα τα ζεύγη κόμβων (All pairs Shortest Paths)

Δίνεται ένας κατευθυνόμενος και βεβαρυμένος γράφος $G(V, E)$, όπως φαίνεται, για παράδειγμα, στο Σχήμα 6.6. Για ευκολία σε αυτή την ενότητα, όταν αναφερόμαστε στο βάρος της ακμής (v_i, v_j) θα λέμε μήκος.



Σχήμα 6.6: Κατευθυνόμενος και βεβαρυμένος γράφος

Για κάθε ζευγάρι κορυφών v_i και v_j του V, απαιτείται να βρούμε το συντομότερο μονοπάτι από το v_i στο v_j κατά μήκος των ακμών του E. Εδώ, το μήκος του μονοπατιού ή του κύκλου είναι το άθροισμα των μηκών των ακμών που το αποτελούν. Στο Σχήμα 6.6, το συντομότερο μονοπάτι από το v_0 στο v_4 είναι κατά μήκος των ακμών (v_0, v_2), (v_2, v_3), (v_3, v_6), (v_6, v_5), και (v_5, v_4) και έχει μήκος 6. Τυπικά, το πρόβλημα όλων των ζευγών των συντομότερων μονοπατιών αναλύεται ως εξής:
Δίνεται ένας γράφος $n-$κορυφών με τον $n \times n$ πίνακα βαρών W.
Κατασκευάζουμε έναν $n \times n$ πίνακα βαρών D τέτοιο ώστε d_{ij} να είναι το μήκος του συντομότερου μονοπατιού από το v_i στο v_j του G_r, για όλα τα i και j. Θα υποθέσουμε ότι ο W έχει θετικά, μηδέν ή αρνητικά στοιχεία όσο δεν υπάρχει κύκλος αρνητικού μήκους στο G. Έστω ότι με d_{ij}^k συμβολίζουμε το μήκος του συντομότερου μονοπατιού από το v_i στο v_j που διέρχεται το πολύ από $k-1$ ενδιάμεσες κορυφές. Έτσι, $d_{ij}^1 = w_{ij}$, δηλαδή το βάρος της ακμής από το v_i στο v_j. Ειδικότερα, αν δεν υπάρχει ακμή από το v_i στο v_j, όπου i και j είναι διακριτά, $d_{ij}^1 = \infty$. Επίσης, $d_{ii}^1 = 0$. Δεδομένου ότι ο G δεν έχει κύκλους ή αρνητικά μήκη, δεν υπάρχει πλεονέκτημα στην επίσκεψη κάποιας κορυφής περισσότερο από μία φορές στο συντομότερο μονοπάτι από το v_i στο v_j (ακόμα κι αν ο ορισμός του μονοπατιού επιτρέπει σε μία κορυφή να εμφανίζεται περισσότερες από μία φορές σε αυτό το μονοπάτι).

Προκύπτει ότι $d_{ij} = d_{ij}^{n-1}$.

Προκειμένου να υπολογίσουμε το d_{ij}^k, για $k > 1$, μπορούμε να χρησιμοποιήσουμε το γεγονός ότι

$$d_{ij}^{k} = \min_{l}\{d_{il}^{n/2} + d_{lj}^{n/2}\}$$

δηλαδή d_{ij}^{k} ισούται με το συντομότερο $d_{il}^{k/2} + d_{lj}^{k/2}$, πάνω από όλες τις τιμές του l. Επομένως, ο πίνακας D μπορεί να παραχθεί από τον D^1 υπολογίζοντας τα $D^2, D^4, \ldots, D^{n-1}$ και τέλος παίρνοντας $D = D^{n-1}$. Προκειμένου να λάβουμε το D^k από το $D^{k/2}$ μέσω της προηγούμενης έκφρασης, μπορούμε να χρησιμοποιήσουμε μια ειδική μορφή πολλαπλασιασμού πινάκων με την οποία οι πρότυπες λειτουργίες του πολλαπλασιασμού πινάκων, δηλαδή $\times$ και $+$ να αντικατασταθούν από $+$ και $\min$, αντίστοιχα.

Έτσι, αν είναι διαθέσιμη μια συνάρτηση πολλαπλασιασμού πινάκων, μπορεί να τροποποιηθεί ώστε να παράγει τον D^{n-1} από τον D^1. Απαιτούνται ακριβώς $\lceil \log(n-1) \rceil$ τέτοια γινόμενα πινάκων. Ο αλγόριθμος υλοποιείται παράλληλα χρησιμοποιώντας οποιαδήποτε συνάρτηση πολλαπλασιασμού πινάκων προσαρμοσμένης να εκτελέσει $(+, \min)$ πολλαπλασιασμό.

Για άλλη μια φορά θα επικαλεστούμε τη συνάρτηση CUBE MATRIX MULTIPLICATION. Ο προκύπτων αλγόριθμος καλείται από τη συνάρτηση CUBE SHORTEST PATHS. Η συνάρτηση τρέχει σε έναν cube-connected SIMD υπολογιστή με $N = n^3$ επεξεργαστές, ο καθένας με τρεις καταχωρητές A, B και C. Όπως και πριν, οι επεξεργαστές μπορεί να θεωρηθεί ότι είναι διατεταγμένοι σε έναν $n \times n \times n$ πρότυπο πίνακα. Αρχικά $A(0, j, k) = w_{jk}$ για $0 \leqslant j, k \leqslant n-1$, δηλαδή οι επεξεργαστές στις θέσεις $(0, j, k)$ περιέχουν τον βεβαρυμένο πίνακα του G.

Αν ο v_j δε συνδέεται με τον v_k ή αν $j = k$, τότε $w_{jk} = 0$. Όταν η συνάρτηση τελειώσει, ο $C(0, j, k)$ περιέχει το μήκος του συντομότερου μονοπατιού από τον v_j στον v_k, για $0 \leqslant j, k \leqslant n-1$.

Διαδικασία 38: CUBE SHORTEST PATHS (A, C)

Βήμα 1
{Κατασκευή του πίνακα D^1 και αποθήκευση στους καταχωρητές A και B}
για $j = 0$ **έως** $n-1$ **κάνε παράλληλα**
  για $k = 0$ **έως** $n-1$ **κάνε παράλληλα**
    (1.1)
    αν $j \neq k$ ***και*** $A(0,j,k) = 0$ **τότε**
      $A(0,j,k) \leftarrow \infty$
    τέλος 1.2
    $B(0,j,k) \leftarrow A(0,j,k)$
  τέλος
τέλος

Βήμα 2
{Κατασκευή των πινάκων $D^2, D^4, \ldots, D^{n-1}$ μέσω επαναλαμβανόμενων πολλαπλασιασμών πινάκων}
για $i = 1$ **έως** $\lceil \log(n-1) \rceil$ **κάνε**
  (2.1) CUBE MATRIX MULTIPLICATION (A,B,C)
  (2.2)
  για $j = 0$ **έως** $n-1$ **κάνε παράλληλα**
    για $k = 0$ **έως** $n-1$ **κάνε παράλληλα**
      (i) $A(0,j,k) \leftarrow C(0,j,k)$
      (ii) $B(0,j,k) \leftarrow C(0,j,k)$
    τέλος
  τέλος
τέλος

Ανάλυση

Τα βήματα 1 και 2.2 απαιτούν σταθερό χρόνο. Υπάρχουν $\lceil \log(n-1) \rceil$ επαναλήψεις του βήματος 2.1, η καθεμία από αυτές απαιτεί $\mathcal{O}(\log n)$ χρόνο. Ο συνολικός χρόνος εκτέλεσης της συνάρτησης CUBE SHORTEST PATHS είναι τελικά $t(n) = \mathcal{O}(\log^2 n)$.
Αφού $p(n) = n^3$,

$$c(n) = \mathcal{O}(n^3 \log^2 n)$$

Παράδειγμα

Οι πίνακες D^1, D^2, D^4 και D^8 του γράφου του Σχήματος 6.6 φαίνονται στο Σχήμα 6.7

$$
\begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array}
\begin{array}{c}
\begin{array}{ccccccc} v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\
\left[\begin{array}{ccccccc}
0 & 4 & 1 & \infty & 7 & \infty & \infty \\
\infty & 0 & 8 & \infty & \infty & \infty & \infty \\
\infty & \infty & 0 & 2 & 6 & \infty & \infty \\
\infty & 5 & \infty & 0 & \infty & \infty & 1 \\
\infty & \infty & \infty & \infty & 0 & \infty & \infty \\
\infty & \infty & \infty & 2 & 1 & 0 & \infty \\
\infty & 3 & \infty & \infty & \infty & 1 & 0
\end{array}\right] \\
(\alpha)
\end{array}
\qquad
\begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array}
\begin{array}{c}
\begin{array}{ccccccc} v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\
\left[\begin{array}{ccccccc}
0 & 4 & 1 & 3 & 7 & \infty & \infty \\
\infty & 0 & 8 & 10 & 14 & \infty & \infty \\
\infty & 7 & 0 & 2 & 6 & \infty & 3 \\
\infty & 4 & 13 & 0 & \infty & 2 & 1 \\
\infty & \infty & \infty & \infty & 0 & \infty & \infty \\
\infty & 7 & \infty & 2 & 1 & 0 & 3 \\
\infty & 3 & 11 & 3 & 2 & 1 & 0
\end{array}\right] \\
(\beta)
\end{array}
$$

$$
\begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array}
\begin{array}{c}
\begin{array}{ccccccc} v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\
\left[\begin{array}{ccccccc}
0 & 4 & 1 & 3 & 7 & 5 & 4 \\
\infty & 0 & 8 & 10 & 14 & 12 & 11 \\
\infty & 6 & 0 & 2 & 5 & 4 & 3 \\
\infty & 4 & 12 & 0 & 3 & 2 & 1 \\
\infty & \infty & \infty & \infty & 0 & \infty & \infty \\
\infty & 6 & 14 & 2 & 1 & 0 & 3 \\
\infty & 3 & 11 & 3 & 2 & 1 & 0
\end{array}\right] \\
(\gamma)
\end{array}
\qquad
\begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array}
\begin{array}{c}
\begin{array}{ccccccc} v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\
\left[\begin{array}{ccccccc}
0 & 4 & 1 & 3 & 6 & 5 & 4 \\
\infty & 0 & 8 & 10 & 13 & 12 & 11 \\
\infty & 6 & 0 & 2 & 5 & 4 & 3 \\
\infty & 4 & 12 & 0 & 3 & 2 & 1 \\
\infty & \infty & \infty & \infty & 0 & \infty & \infty \\
\infty & 6 & 14 & 2 & 1 & 0 & 3 \\
\infty & 3 & 11 & 3 & 2 & 1 & 0
\end{array}\right] \\
(\delta)
\end{array}
$$

Σχήμα 6.7: Εύρεση ελάχιστων μονοπατιών για όλα τα ζεύγη κόμβων του Σχήματος 6.6

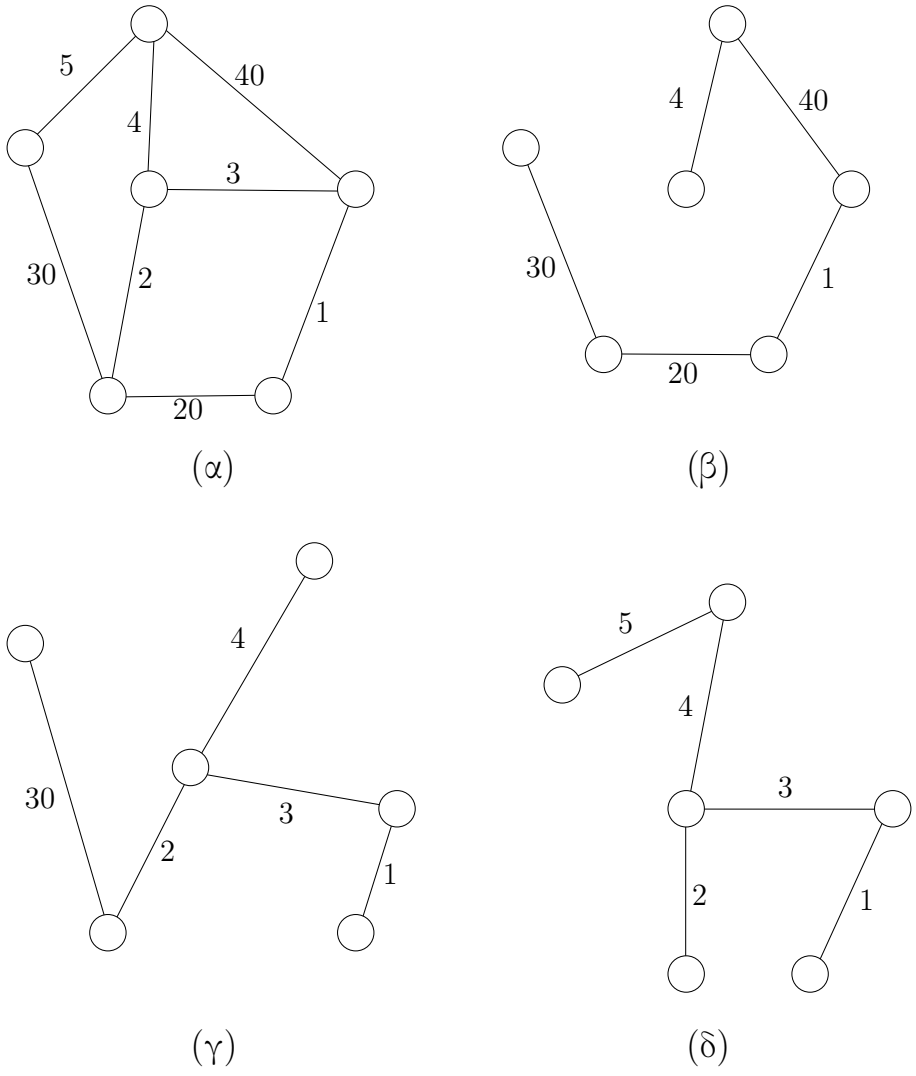
6.6 Υπολογίζοντας το Minimum Spanning Tree (Ελάχιστο δέντρο επικάλυψης)

Ένα δέντρο είναι ένας συνδεδεμένος (μη κατευθυνόμενος) γράφος χωρίς κύκλους.
Δίνεται ένας μη κατευθυνόμενος και συνδεδεμένος γράφος $G(V, E)$, το δέντρο επικάλυψής του G είναι ένας υπογράφος $G'(V', E')$ του G τέτοιος ώστε

1. (i) G' είναι ένα δέντρο
2. (ii) $V' = V$

Αν ο γράφος G είναι βεβαρυμένος, τότε το ελάχιστο δέντρο επικάλυψης (MST) του G είναι αυτό με το μικρότερο συνολικό βάρος, ανάμεσα σε όλα τα δέντρα επικάλυψης του G. Αυτοί οι ορισμοί φαίνονται στο Σχήμα 6.8. Τρία δέντρα επικάλυψης του βεβαρυμένου γράφου του Σχήματος 6.8(α)

φαίνονται στα Σχήματα 6.8(β-δ).



Σχήμα 6.8: Βεβαρυμένος γράφος και τρία δέντρα επικάλυψης του

Το δέντρο το Σχήματος 6.8(δ) έχει ελάχιστο βάρος. Να σημειωθεί ότι όταν όλες οι ακμές του γράφου έχουν διακριτά μέρη, το MST είναι μοναδικό. Αν $V = \nu_0, \nu_1, \ldots, \nu_{n-1}$, τότε το MST έχει $n-1$ ακμές. Αυτές οι ακμές πρέπει ενδεχομένως να επιλεγούν ανάμεσα από $\frac{n(n-1)}{2}$ υποψήφιες. Αυτό δίνει ένα $\Omega(n^2)$ κάτω φράγμα του αριθμού των λειτουργιών που απαιτούνται για τον υπολογισμό του MST αφού κάθε ακμή πρέπει να εξεταστεί τουλάχιστον μία φορά.

Για ευκολία, από δώ και στο εξής θα αναφερόμαστε στο βάρος της ακμής (v_i, v_j) σαν την απόσταση που χωρίζει το v_i και το v_j και το συμβολίζουμε με $\mathtt{dist}(v_i, v_j)$.
Ένας ακολουθιακός αλγόριθμος για τον υπολογισμό του MST βασίζεται στην greedy προσέγγιση του προβλήματος, επιλύοντάς το σταδιακά. Αρχίζοντας με την τυχαία επιλογή μιας κορυφής, σε κάθε φάση προστίθεται μία κορυφή και η σχετική ακμή του δέντρου. Αν v_i είναι μια κορυφή η οποία δεν είναι ακόμη στο δέντρο, τότε με $c(v_i)$ συμβολίζουμε μια κορυφή που είναι ήδη στο δέντρο και είναι η πιο κοντινή στη v_i. Ο αλγόριθμος αποτελείται από δύο βήματα:

Βήμα 1: Περιέλαβε την κορυφή v_0 στο MST και θέσε $c(v_i) = v_0$ για $i = 1, 2, \ldots, n-1$

Βήμα 2: Το βήμα αυτό επαναλαμβάνεται όσο υπάρχουν κορυφές που δεν ανήκουν ακόμα στο MST:
(2.1) Περιέλαβε στο δέντρο την πιο κοντινή κορυφή η οποία δεν είναι ακόμα στο δέντρο, δηλαδή για όλες τις v_i που δεν είναι στο MST να βρεθεί ακμή $(v_i, c(v_i))$ για την οποία η $\mathtt{dist}(v_i, c(v_i))$ είναι η μικρότερη και πρόσθεσέ τη στο δέντρο
(2.2) Για όλες τις v_i που δεν βρίσκονται στο MST, ενημέρωσε το $c(v_i)$, δηλαδή, υποθέτοντας ότι η v_j ήταν η πιο πρόσφατη κορυφή που προστέθηκε στο δέντρο, τότε το $c(v_i)$ μπορεί να ενημερωθεί προσδιορίζοντας το μικρότερο από τα $\mathtt{dist}(v_i, c(v_i))$ και $\mathtt{dist}(v_i, v_j)$

Το βήμα 1 απαιτεί n σταθερού χρόνου λειτουργίες. Το βήμα 2 εκτελείται μια φορά για κάθε μία από τις $n-1$ κορυφές. Αν υπάρχουν ήδη k κορυφές στο δέντρο, τότε τα βήματα 2.1 και 2.2 αποτελούνται από $n-k-1$ και $n-k$ συγκρίσεις, αντίστοιχα. Έτσι, το βήμα 2 και επομένως ο αλγόριθμος, απαιτεί χρόνο ανάλογο του $\sum_{k=1}^{n-1}(n-k)$, ο οποίος είναι $\mathcal{O}(n^2)$. Αυτός ο ακολουθιακός χρόνος εκτέλεσης είνα επομένως βέλτιστος από την άποψη του χαμηλότερου φράγματος που αναφέρθηκε προηγουμένως.
Θα δείξουμε τώρα πως αυτός ο αλγόριθμος μπορεί να προσαρμοστεί για να τρέξει παράλληλα σε έναν EREW SM SIMD υπολογιστή. Η παράλληλη υλοποίηση χρησιμοποιεί N επεξεργαστές $P_0, P_1, \ldots, P_{N-1}$. Ο αριθμός των επεξεργαστών είναι ανεξάρτητος του αριθμού των κορυφών του G εκτός από το ότι υποθέτουμε ότι $1 < N < n$.
Όπως κάναμε και σε προηγούμενα κεφάλαια, είνα πιο βολικό να γράψουμε $N = n^{1-x}$, όπου $0 < x < 1$. Σε κάθε επεξεργαστή P_i εκχωρείται μία διακριτή υποακολουθία V_i του V μεγέθους n^x. Με άλλα λόγια, ο P_i

είναι «υπεύθυνος» των κορυφών V_i. Να σημειωθεί ότι ο P_i χρειάζεται μόνο να αποθηκεύσει τους δείκτες της πρώτης και τελευταίας κορυφής του V_i. Κατά τη διάρκεια της διεργασίας κατασκευής του MST και για κάθε κορυφή v_p του V_i η οποία δεν είναι ακόμα στο δέντρο, ο P_i επίσης σημειώνει (αποθηκεύει) την κοντινότερη κορυφή στο δέντρο, και τη συμβολίζει με $c(v_p)$. Ο πίνακας βάρους W του G αποθηκεύεται στη διαμοιραζόμενη μνήμη, όπου $w_{ij} = dist(v_i, v_j)$, για $i, j = 0, 1, \ldots, n-1$. Αν $i = j$ ή αν v_i και v_j δεν είναι απ' ευθείας συνδεδεμένοι μέσω μίας ακμής, τότε $w_{ij} = \infty$. Ο αλγόριθμος αρχικά περιλαμβάνει μία τυχαία κορυφή του δέντρου. Ο υπολογισμός του MST εκτελείται σε $n-1$ φάσεις. Κατά τη διάρκεια κάθε φάσης, μία νέα κορυφή και έτσι μία νέα ακμή προστίθενται στο υπάρχον μερικό δέντρο.
Αυτό γίνεται ως εξής:

Με όλους τους επεξεργαστές να λειτουργούν παράλληλα, κάθε επεξεργαστής βρίσκει ανάμεσα στις κορυφές που δεν ανήκουν στο δέντρο την πιο κοντινή κορυφή (έσω κορυφή) στο δέντρο. Ανάμεσα στις n^{1-x} κορυφές που βρέθηκαν, η πιο κοντινή κορυφή (έσω κορυφή) στο δέντρο βρέθηκε και προστέθηκε στο δέντρο μαζί με την αντίστοιχη ακμή. Αυτή η κορυφή, την οποία λέμε v_h, την κάνουμε τώρα γνωστή σε όλους τους επεξεργαστές. Το ακόλουθο βήμα είναι για την παράλληλη εκτέλεση από όλους τους επεξεργαστές, για καθεμιά από τις n^x κορυφές.
Για κάθε κορυφή v_p η οποία δεν είναι ακόμα στο δέντρο, αν $dist(v_p, v_h) < dist(v_p, c(v_p))$, τότε η $c(v_p)$ γίνεται ίση με την v_h. Ο αλγόριθμος δίνεται παρακάτω σαν συνάρτηση EREW MST.
Η συνάρτηση χρησιμοποιεί τη συνάρτηση BROADCAST που περιγράψαμε στην ενότητα 2 και τη συνάρτηση MINIMUM $(x_1, x_2, \ldots, x_m)$, η οποία χρησιμοποιεί m επεξεργαστές για να βρει το μικρότερο στοιχείο του πίνακα $x_1, x_2, \ldots, x_m$ και το επιστρέφει στη x_1.

Διαδικασία 39: MINIMUM $(x_1, x_2, \ldots, x_m)$

```
για j = 0 έως (log m − 1) κάνε
    για i = 1 έως m με βήμα 2^(j+1) κάνε παράλληλα
        (1) Ο P_i λαμβάνει x_(i+2^j), μέσω διαμοιραζόμενης μνήμης
        (2)
        αν x_(i+2^j) < x_i τότε
            x_i ← x_(i+2^j)
        τέλος
    τέλος
τέλος
```

Διαδικασία 40: EREW MST (W, TREE)

Βήμα 1

(1.1) Ονόμασε την κορυφή v_0 του V_0 σαν κορυφή που βρίσκεται ήδη στο δέντρο

(1.2) **για** $i = 0$ **έως** $N - 1$ **κάνε παράλληλα**

  για *κάθε κορυφή* v_j *του* V_i **κάνε**

    $c(v_j) \leftarrow v_0$

  τέλος

τέλος

Βήμα 2

για $i = 1$ **έως** $n - 1$ **κάνε**

  (2.1) **για** $j = 0$ **έως** $N - 1$ **κάνε παράλληλα**

    (i) ο P_j βρίσκει τη μικρότερη από τις ποσότητες $\mathrm{dist}(v_p, c(v_p))$, όπου v_p είναι η κορυφή του V_j που δεν είναι ακόμα στο δέντρο

    (ii) Έστω ότι η μικρότερη ποσότητα που βρέθηκε στο (i) είναι $\mathrm{dist}(v_r, v_t)$: ο P_j παραδίδει μια τριάδα (d_j, a_j, b_j) όπου
    $d_j = \mathrm{dist}(v_r, v_t)$
    $a_j = v_r$ και
    $b_j = v_t$

  τέλος

  (2.2) Χρησιμοποιώντας τη συνάρτηση MINIMUM η μικρότερη των αποστάσεων d_j και οι συναφείς κορυφές a_j και b_j, για $0 \leqslant j \leqslant N - 1$, βρίσκονται. Έστω ότι αυτή η τριάδα είναι η (d_s, a_s, b_s), όπου a_s είναι κάποια κορυφή v_h που δεν βρίσκεται στο δεύτερο και b_s είναι κάποια κορυφή v_k που βρίσκεται ήδη στο δέντρο

  (2.3) Ο P_0 εκχωρεί το (v_h, v_k) στο $\mathrm{TREE}(i)$, το i-οστό στοιχείο του πίνακα TREE

  (2.4) Χρησιμοποιώντας τη BROADCAST, η v_h γίνεται γνωστή στους N επεξεργαστές

  (2.5) **για** $j = 0$ **έως** $N - 1$ **κάνε παράλληλα**

    (i) **αν** v_h *είναι στο* V_j **τότε**

      Ο P_j ονομάζει τη v_h σαν κορυφή ήδη μεσα στο δέντρο

    τέλος

    (ii) **για** *κάθε κορυφή* v_p *του* V_j *η οποία δεν βρίσκεται ακόμα στο δέντρο* **κάνε**

      αν $\mathrm{dist}(v_p, v_h) < \mathrm{dist}(v_p, c(v_p))$ **τότε**

        $c(v_p) \leftarrow v_h$

      τέλος

    τέλος

  τέλος

τέλος

Παράγει έναν πίνακα TREE στη διαμοιραζόμενη μνήμη που περιέχει τις $n-1$ ακμές του MST. Όταν δύο αποστάσεις είναι ίσες, η συνάρτηση κάνει την επιλογή τυχαία.

Ανάλυση

Το βήμα 1.1 γίνεται σε σταθερό χρόνο. Αφού κάθε επεξεργαστής είναι υπεύθυνος για n^x κορυφές, το βήμα 1.2 απαιτεί n^x αναθέσεις. Επομένως, το βήμα 1 τρέχει σε $\mathcal{O}(n^x)$ χρόνο. Στο βήμα 2.1, ένας επεξεργαστής βρίσκει τη μικρότερη από n^x ποσότητες (ακολουθιακά) κάνοντας $n^x - 1$ συγκρίσεις. Οι συναρτήσεις MINIMUM και BROADCAST εμπλέκουν και οι δύο $\mathcal{O}(\log N)$ σταθερού χρόνου λειτουργίες. Αφού $N = n^{1-x}$, τα βήματα 2.2 και 2.4 πραγματοποιούνται σε $\mathcal{O}(\log n)$ χρόνο. Είναι φανερό ότι τα βήματα 2.3 και 2.5 απαιτούν σταθερό χρόνο και $\mathcal{O}(n^x)$ χρόνο, αντίστοιχα.Έτσι, κάθε επανάληψη του βήματος 2 απαιτεί $\mathcal{O}(n^x)$ χρόνο. Αφού αυτό το βήμα επαναλαμβάνεται $n+1$ φορές, συμπληρώνεται σε $\mathcal{O}(n^{1+x})$ χρόνο. Κατά συνέπεια, ο συνολικός χρόνος εκτέλεσης της συνάρτησης είναι $\mathcal{O}(n^{1+x})$. Η συνάρτηση είναι τότε προσαρμοστική (adaptive). Το κόστος της είναι

$$c(n) = p(n) \times t(n) = n^{1-x} \times \mathcal{O}(n^{1+x}) = \mathcal{O}(n^2)$$

Αυτό σημαίνει ότι η συνάρτηση είναι βέλτιστου κόστους. Να σημειωθεί ότι για αρκετά μεγάλο n, $n^x > \log n$ για κάθε x και $N = n^{1-x} < n/\log n$. Η βελτίωση της συνάρτησης επομένως προσδιορίζεται στην περιοχή όπου $N < n/\log n$.

Παράδειγμα

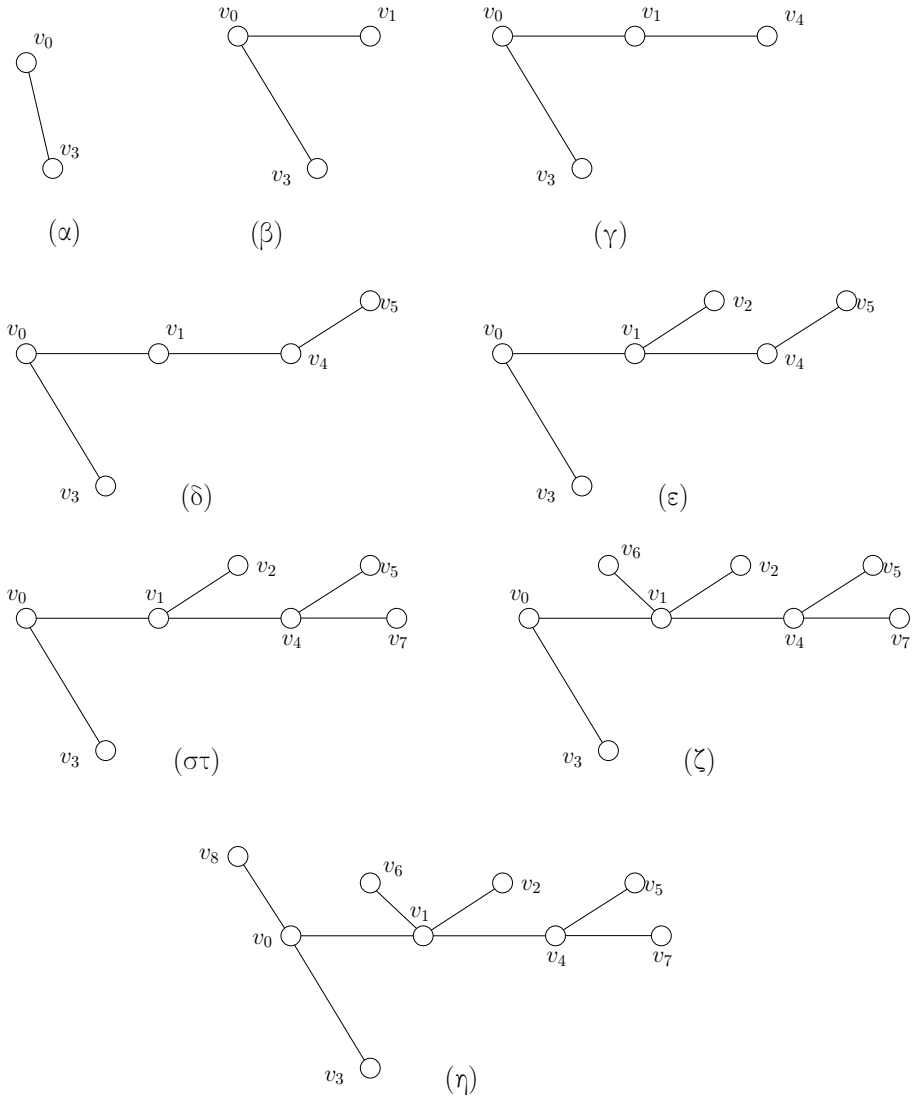
Έστω G ένας βεβαρυμένος 9-κόμβων γράφος του οποίου ο πίνακας βαρών φαίνεται στο Σχήμα 6.9.

Επίσης υποθέτουμε ότι ένας EREW SM SIMD υπολογιστής με τρεις επεξεργαστές είναι διαθέσιμος. Έτσι $3 = 9^{1-x}$, δηλαδή $x = 0.5$. Στους επεξεργαστές P_0, P_1 και P_2 ανατίθενται ακολουθίες $V_0 = \{v_0, v_1, v_2\}$, $V_1 = \{v_3, v_4, v_5\}$ και $V_2 = \{v_6, v_7, v_8\}$. Στο βήμα 1.1, η v_0 περιλαμβάνεται στο δέντρο και θεωρείται η πιο κοντινή κορυφή του δέντρου από όλες τις εναπομείναντες κορυφές.
Κατά τη διάρκεια της $1^{ης}$ επανάληψης του βήματος 2, ο P_0 καθορίζει ότι $\mathrm{dist}(v_1, v_0) < \mathrm{dist}(v_2, v_0)$ και επιστρέφει την τριάδα $(5, v_1, v_0)$. Όμοια, οι P_1 και P_2 επιστρέφουν $(5, v_3, v_0)$ και $(5, v_8, v_0)$ αντίστοιχα.
Η συνάρτηση MINIMUM χρησιμοποιείται τότε για να καθορίσει $v_h = v_3$

$$
\begin{array}{c}
 \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \\ v_8
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc} v_0 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \end{array} \\
\left[\begin{array}{ccccccccc}
\infty & 5 & 6 & 1 & \infty & 6 & 10 & \infty & 5 \\
5 & \infty & 3 & 9 & 2 & 5 & 4 & 12 & \infty \\
6 & 3 & \infty & 7 & 3 & 9 & 11 & \infty & 14 \\
1 & 9 & 7 & \infty & 10 & \infty & \infty & 9 & 8 \\
\infty & 2 & 3 & 10 & \infty & 1 & 5 & 3 & 15 \\
6 & 5 & 9 & \infty & 1 & \infty & 6 & 13 & \infty \\
10 & 4 & 11 & \infty & 5 & 6 & \infty & 4 & 16 \\
\infty & 12 & \infty & 9 & 3 & 13 & 4 & \infty & 7 \\
5 & \infty & 14 & 8 & 15 & \infty & 16 & 7 & \infty
\end{array}\right]
\end{array}
$$

Σχήμα 6.9: Πίνακας με βάρη γράφου 9 κόμβων

και έτσι $\mathsf{TREE}(1) = (\nu_3, \nu_0)$. Τώρα, η ν_3 γίνεται γνωστή σε όλους τους επεξεργαστές με την κλήση της BROADCAST και ο P_1 την ονομάζει σαν κορυφή μέσα στο δέντρο. Στο βήμα 2.5, ο P_0 διατηρεί το $\mathsf{c}(\nu_1)$ και θέτει το $\mathsf{c}(\nu_2)$ ίσο με το ν_0, ο P_2 ενημερώνει το $\mathsf{c}(\nu_4)$ με το ν_3 αλλά διατηρεί το $\mathsf{c}(\nu_5) = \nu_0$ και ο P_3 διατηρεί $\mathsf{c}(\nu_6) = \nu_0$ και $\mathsf{c}(\nu_8) = 0$ ενώ ενημερώνει το $\mathsf{c}(\nu_7) = \nu_3$. Η διαδικασία συνεχίζεται μέχρι όλα τα δέντρα $(\nu_3, \nu_0), (\nu_1, \nu_0), (\nu_4, \nu_1), (\nu_5, \nu_4), (\nu_2, \nu_1), (\nu_7, \nu_4), (\nu_6, \nu_1), (\nu_8, \nu_0)$ να παραχθούν.
Η διαδικασία αυτή φαίνεται στο Σχήμα 6.10.



Σχήμα 6.10: Υπολογισμός ελάχιστου δέντρου επικάλυψης με χρήση της συνάρτησης EREW MST