ΠΑΡΑΛΛΗΛΑ ΣΥΣΤΗΜΑΤΑ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΙΑ 1

OpenMP

ΣΤΟΙΧΕΙΑ ΦΟΙΤΗΤΗ / ΕΡΓΑΣΙΑΣ

ΟΝΟΜΑΤΕΠΩΝΥΜΟ : ΑΘΑΝΑΣΙΟΥ ΒΑΣΙΛΕΙΟΣ ΕΥΑΓΓΕΛΟΣ
ΑΡΙΘΜΟΣ ΜΗΤΡΩΟΥ : 19390005
ΕΞΑΜΗΝΟ ΦΟΙΤΗΤΗ : 11
ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ : ΠΑΔΑ

ΥΠΕΥΘΥΝΟΣ ΕΡΓΑΣΤΗΡΙΟΥ: ΙΟΡΔΑΝΑΚΗΣ ΜΙΧΑΛΗΣ
ΥΠΕΥΘΥΝΟΣ ΘΕΩΡΙΑΣ: ΜΑΜΑΛΗΣ ΒΑΣΙΛΕΙΟΣ

ΠΕΡΙΕΧΟΜΕΝΑ

1. Εισαγωγή 3
1.1 Σκοπός της άσκησης 3
1.2 Συνοπτική περιγραφή του προβλήματος που επιλύεται 3
2. Σχεδιασμός 4
2.1 Περιγραφή της προσέγγισης που ακολουθήθηκε 4
2.2 Ανάλυση της λογικής και των μεθοδολογιών 4
2.3 Περιγραφή των δομών δεδομένων και των αλγορίθμων 5
2.3.1 Δομές δεδομένων και μεταβλητές 5
2.3.2 Πρόγραμμα γεννήτρια για παραγωγή 2Δ πινάκων 6
2.3.3 Αλγόριθμος δυαδικού δένδρου 8
3. Υλοποίηση 9
3.1 Αναφορά στις βασικές λειτουργίες του κώδικα 9
3.2 Επεξήγηση παράλληλων τμημάτων του κώδικα 10
3.3 Περιγραφή της επικοινωνίας και του συγχρονισμού μεταξύ νημάτων 14
4. Δοκιμές και Αποτελέσματα 17
4.1 Αναφορά των συνθηκών εκτέλεσης 17
4.2 Παρουσίαση των αποτελεσμάτων σε μορφή κειμένου 17
4.2.1 Output_no_args.txt 17
4.2.2 Output_no_strict_diagonial.txt 17
4.2.3 Output_T1_N10_CZ2.txt 18
4.2.4 Output_T2_N16_CZ3.txt 19
4.2.5 Output_T4_N25_CZ5.txt 20
4.2.6 Output_T8_N400_CZ20.txt 21
4.2.7 Output_T12_N1000_CZ100.txt 23
4.2.8 Output_T8_N10000_CZ1000.txt 24
4.3 Ανάλυση της αποδοτικότητας 25
4.3.1 Χρόνοι εκτέλεσης των παράλληλων εργασιών 25
4.3.2 Επιταχύνσεις 28
4.3.3 Παρατηρήσεις 32
5. Προβλήματα και Αντιμετώπιση 32
5.1 Αναφορά προβλημάτων 32
5.2 Λύσεις που δοκιμάστηκαν και εφαρμόστηκαν 32
6. Συμπεράσματα 34
6.1 Ανακεφαλαίωση 34

1. Εισαγωγή

1.1 Σκοπός της άσκησης

Ο σκοπός της άσκησης είναι η υλοποίηση και η αξιολόγηση ενός παράλληλου προγράμματος με χρήση του OpenMP που αποτελεί σε χαμηλό επίπεδο το πολυνηματικό (multithreaded) πρότυπο παράλληλου προγραμματισμού, με τη λογική του μοντέλου παράλληλης εκτέλεσης fork-join.

Συγκεκριμένα, ο προγραμματιστής εξοικειώνεται με τις high level τεχνικές που του παρέχει το OpenMP, όπου διακρίνει το τμήμα κώδικα που θα εκτελεστεί παράλληλα και κατανοεί τους διαμοιρασμούς των εργασιών στα αντίστοιχα νήματα (threads) και την συγχρονισμένη επικοινωνία μεταξύ νημάτων

Τέλος, η άσκηση αποσκοπεί στο να αναδείξει και τους χρόνους εκτέλεσης των παράλληλων εργασιών, ώστε να υπολογιστούν και να συγκριθούν οι επιταχύνσεις (speed-up) τόσο κατά την περίπτωση που το πρόγραμμα εκτελείται ακολουθιακά (με 1 νήμα), όσο παράλληλα (> 1 νήματα).

1.2 Συνοπτική περιγραφή του προβλήματος που επιλύεται

Το πρόβλημα αφορά τις εξής εργασίες:

a) την επεξεργασία ενός τετραγωνικού πίνακα A (N x N) με στόχο τον έλεγχο διαγώνιας κυριαρχίας (ελέγχεται παράλληλα αν είναι αυστηρά διαγώνια δεσπόζων ή όχι), για κάθε γραμμή να ελέγχεται αν ισχύει η ιδιότητα

$$|A_{ii}| > \sum |A_{ij}| \text{ όπου } j = 0 \dots N-1 \text{ για } i \neq j$$

b) τον υπολογισμό του μέγιστου κατά απόλυτη τιμή στοιχείου της διαγωνίου του A

$$m = \max(|A_{ii}|), \qquad i = 0 \dots N-1$$

c) τη δημιουργία ενός νέου πίνακα B επίσης τετραγωνικού (N x N) με στοιχεία

$$B_{ij} = m - |A_{ij}| \text{ για } i \neq j \text{ και } B_{ij} = m \text{ για } i = j$$

d) τον υπολογισμό όπως του ελάχιστου κατά απόλυτη τιμή στοιχείου του B με διάφορες μεθόδους όπως είναι ο αλγόριθμος δυαδικού δένδρου

$$\text{min\_val} = \min(|B_{ij}|), \qquad \text{για } i,j = 0 \dots N-1$$

e) την ανάλυση της απόδοσης για διάφορες τιμές παράμετρων

Όσον αφορά την απόδοση, λαμβάνονται υπόψιν οι εξής παράμετροι:

- Μέγεθος πίνακα N
- Αριθμός νημάτων T
- Αριθμός στοιχείων CZ που θα διαμοιραστούν στα νήματα

2. Σχεδιασμός

2.1 Περιγραφή της προσέγγισης που ακολουθήθηκε

Η υλοποίηση βασίζεται στην κατανομή των υπολογισμών σε πολλαπλά νήματα (threads) χρησιμοποιώντας το OpenMP για την επίτευξη παράλληλης επεξεργασίας. Η προσέγγιση διαχωρίζεται σε διακριτές φάσεις:

- **Ανάθεση εργασιών στα νήματα:** Με χρήση OpenMP, κάθε νήμα εκτελεί συγκεκριμένα κομμάτια των υπολογισμών.
- **Αξιοποίηση της δυνατότητας του OpenMP:** Στην κατανομή εργασιών, στον συγχρονισμό και στις οδηγίες reduction.
- **Ανάλυση της δομής του προβλήματος:** Τα δεδομένα οργανώνονται ώστε να διευκολύνουν την παράλληλη επεξεργασία, ελαχιστοποιώντας την επικοινωνία μεταξύ νημάτων.

Για κάθε εργασία που αναφέρθηκε στο κεφάλαιο 1.2 Συνοπτική περιγραφή του προβλήματος που επιλύεται ορίζεται ξεχωριστή περιοχή παράλληλης επεξεργασίας, ώστε να διακριθούν οι ακολουθιακές εργασίες που πρέπει να γίνονται πριν και μετά την παράλληλη επεξεργασία κάθε εργασίας.

2.2 Ανάλυση της λογικής και των μεθοδολογιών

Η λογική βασίζεται στα εξής:

a) **Έλεγχος του πίνακα Α για το αν είναι αυστηρά διαγώνια δεσπόζων:** Οι γραμμές του πίνακα διαμοιράζονται στα νήματα ανάλογα με την παράμετρο CZ και ελέγχεται παράλληλα αν ο Α είναι αυστηρά διαγώνια δεσπόζων μέσω μίας κοινής μεταβλητής

b) **Υπολογισμός μέγιστου στοιχείου της διαγωνίου του Α:** Το κάθε νήμα υπολογίζει τοπικά το μέγιστο στοιχείο της διαγωνίου και τα αποτελέσματα συγκεντρώνονται σε μια μεταβλητή (οδηγία reduction)

c) **Δημιουργία νέου πίνακα Β με τιμές Bij = m – |Aij| για i ≠ j και Bij = m για i = j:** Το κάθε νήμα υπολογίζει τοπικά τα στοιχεία του πίνακα Β, με την λογική του ενιαίου διαμοιρασμού (οδηγία collapse)

d) **Υπολογισμός ελάχιστου στοιχείου του Β:** Το κάθε νήμα υπολογίζει τοπικά το ελάχιστο στοιχείο που του αναλογεί από τους διαμοιρασμούς και συγκεντρώνονται τα τοπικά αποτελέσματα για να υπολογιστεί το ελάχιστο στοιχείο του Β. Η εργασία πραγματοποιείται με 3 μεθόδους, με οδηγία reduction, με μηχανισμό προστασίας κρίσιμης περιοχής και με αλγόριθμο δυαδικού δένδρου

2.3 Περιγραφή των δομών δεδομένων και των αλγορίθμων

2.3.1 Δομές δεδομένων και μεταβλητές

Για την επίλυση του προβλήματος χρησιμοποιήθηκαν οι παρακάτω δομές δεδομένων και μεταβλητές:

Όνομα Μεταβλητής	Περιγραφή Μεταβλητής
int	
**A	Δυναμικός τετραγωνικός 2Δ πίνακας μέγεθους N, όπου αποθηκεύονται ακέραιοι αριθμοί (θετικοί ή αρνητικοί) μέσω ενός προγράμματος-γεννήτρια που δημιουργεί είτε αυστηρά διαγώνιους δεσπόζων πίνακες είτε όχι
**B	Δυναμικός τετραγωνικός 2Δ πίνακας μέγεθους N, όπου αποθηκεύονται οι ακέραιοι αριθμοί που αναφέρονται στο κεφάλαιο 1.2 Συνοπτική περιγραφή του προβλήματος που επιλύεται στην εργασία c)
*M	Δυναμικός 1Δ πίνακας μεγέθους T όσο του αριθμού των νημάτων, όπου χρησιμοποιείται για την υλοποίηση του αλγορίθμου δυαδικού δένδρου για την εύρεση της ελάχιστης τιμής του B
i, j, k	Δείκτες επανάληψης για τους διαμοιρασμούς
rowSum	Άθροισμα των στοιχείων μιας γραμμής του πίνακα A
chunk	Αριθμοί επαναλήψεων ανα thread
flag	Μεταβλητή «δείκτης» για να ελέγξουμε αν ο A είναι αυστηρά διαγώνια δεσπόζων
tid	Αναγνωριστικό thread
loc_sum	Τοπική μεταβλητή για κάθε thread που υπολογίζει το άθροισμα των στοιχείων που του έχουν διαμοιραστεί
loc_flag	Τοπική μεταβλητή «δείκτης» για να ελέγχει το κάθε thread αν ισχύει η ιδιότητα της διαγώνιας κυριαρχίας στα στοιχεία που του έχουν διαμοιραστεί
loc_index	Τοπική μεταβλητή για κάθε thread που αποθηκεύει το στοιχείο που ανήκει στην διαγώνιο του A
loc_min	Τοπική μεταβλητή για κάθε thread που χρησιμοποιείται στον αλγόριθμο δυαδικού δένδρου και αποθηκεύει το μικρότερο στοιχείο από τα M[tid], M[tid+incr]
incr	Δείκτης επανάληψης για τον αλγόριθμο δυαδικού δένδρου
temp0	Το στοιχείο M[tid]
temp1	Το στοιχείο M[tid+incr]

m	Το μέγιστο στοιχείο της διαγωνίου του πίνακα A
min_val	Το ελάχιστο στοιχείο του πίνακα B
double	
loc_time_start	Έναρξη χρόνου μέτρησης της παράλληλης επεξεργασίας για κάθε εργασία ξεχωριστά
loc_time_end	Λήξη χρόνου μέτρησης της παράλληλης επεξεργασίας για κάθε εργασία ξεχωριστά
all_time_start	Έναρξη χρόνου μέτρησης ολόκληρου του παράλληλου προγράμματος
all_time_end	Λήξη χρόνου μέτρησης ολόκληρου του παράλληλου προγράμματος
FILE*	
fpA	Αρχείο εξόδου για την αποθήκευση του πίνακα A
fpB	Αρχείο εξόδου για την αποθήκευση του πίνακα B

2.3.2 Πρόγραμμα γεννήτρια για παραγωγή 2Δ πινάκων

Ο αλγόριθμος αυτός δημιουργεί τον τετραγωνικό πίνακα A διαστάσεων N x N με βάση 2 επιλογών: την επιλογή να είναι αυστηρά διαγώνια δεσπόζων ή να μην είναι. Η επιλογή γίνεται τυχαία με την παρακάτω εντολή, όπου παράγει την τιμή 1 για να είναι αυστηρά διαγώνια δεσπόζων ή την τιμή 0 για να μην είναι:

```
rand() % 2
```

- **Αυστηρά διαγώνια δεσπόζων (τιμή 1):**
  - Κώδικας:

```
for (i = 0; i < N; i++)
{
	rowSum = 0;
	for (j = 0; j < N; j++)
	{
		if (i == j)
		{
			Array[i][i] = rand() % 21 - 10;
			Array[i][i] = Array[i][i] >= 0 ? Array[i][i] + 20 :
Array[i][i] - 20;
		}
		else
		{
			Array[i][j] = rand() % 21 - 10;
			rowSum += abs(Array[i][j]);
		}
	}
```

```
            if (rowSum >= abs(Array[i][i]))
            {
                Array[i][i] = rowSum + rand() % 5 + 1;
                Array[i][i] *= (rand() % 2 == 0) ? 1 : -1;
            }
        }
```

- Λειτουργία:
  - Με 2 βρόχους κάνουμε προσπέλαση τον 2Δ πίνακα
  - Ελέγχουμε αν βρισκόμαστε στην κύρια διαγώνιο
  - Επιλέγουμε τιμές από -10 εώς 10
  - Αλλάζουμε την τιμή κατά 20 ανάλογα το πρόσημο, ώστε να δώσουμε μεγαλύτερες τιμές στην κύρια διαγώνιο για να ισχύει η ιδιότητα της διαγώνιας κυριαρχίας
  - Αν δεν βρισκόμαστε στην κύρια διαγώνιο, τότε επιλέγουμε τιμές πάλι στο διάστημα -10 εώς 10, χωρίς να αλλάξουμε την τιμή, ώστε να μην υπερβούν τα στοιχεία της κύριας διαγωνίου
  - Προσθέτουμε για κάθε γραμμή τα στοιχεία της μη κυρίας διαγωνίου στην μεταβλητή *rowSum*
  - Ελέγχουμε αν ισχύει η ιδιότητα

$$|A_{ii}| > \sum |A_{ij}| \text{ όπου } j = 0 \dots N-1 \text{ για } i \neq j$$

  - Αν δεν ισχύει, τότε προσαρμόζουμε τα στοιχεία της κυρίας διαγωνίου
  - Προσθέτουμε το άθροισμα των υπόλοιπων στοιχείων της γραμμής με τιμές από 1 εώς 5 και αποθηκεύουμε το αποτέλεσμα στο στοιχείο της κύριας διαγωνίου
  - Επιλέγουμε τυχαία πρόσημο

- **Μη αυστηρά διαγώνια δεσπόζων (τιμή 0):**
  - Κώδικας

```
for (i = 0; i < N; i++)
{
      rowSum = 0;
      for (j = 0; j < N; j++)
      {
            if (i == j)
            {
                  Array[i][i] = rand() % 11 - 5;
            }
            else
            {
                  Array[i][j] = rand() % 21 - 10;
                  rowSum += abs(Array[i][j]);
```

```
            }
        }
        if (abs(Array[i][i]) > rowSum)
        {
            Array[i][i] = rowSum - rand() % 5 - 1;
            Array[i][i] *= (rand() % 2 == 0) ? 1 : -1;
        }
    }
```

- Λειτουργία:
  - Με 2 βρόχους κάνουμε προσπέλαση τον 2Δ πίνακα
  - Ελέγχουμε αν βρισκόμαστε στην κύρια διαγώνιο
  - Επιλέγουμε τιμές από -5 εώς 5
  - Αν δεν βρισκόμαστε στην κύρια διαγώνιο, τότε επιλέγουμε τιμές στο διάστημα -10 εώς 10, ώστε να υπερβούν τα στοιχεία της κύριας διαγωνίου
  - Προσθέτουμε για κάθε γραμμή τα στοιχεία της μη κυρίας διαγωνίου στην μεταβλητή *rowSum*
  - Ελέγχουμε αν ισχύει η ιδιότητα

$$|A_{ii}| > \sum |A_{ij}| \text{ όπου } j = 0 \dots N-1 \text{ για } i \neq j$$

  - Αν ισχύει, τότε προσαρμόζουμε τα στοιχεία της κυρίας διαγωνίου
  - Αφαιρούμε το άθροισμα των υπόλοιπων στοιχείων της γραμμής με τιμές από -1 εώς 3 και αποθηκεύουμε το αποτέλεσμα στο στοιχείο της κύριας διαγωνίου
  - Επιλέγουμε τυχαία πρόσημο

2.3.3 Αλγόριθμος δυαδικού δένδρου

Ο αλγόριθμος δυαδικού δένδρου είναι μια γρήγορη τεχνική που αντικαθιστά την οδηγία reduction και συγχρονίζει τα νήματα με φάσεις να κάνουν παράλληλους υπολογισμούς, όπως άθροισμα, μέγιστο στοιχείο κλπ. Η ιδέα του αλγόριθμου είναι να μειώνει διαδοχικά τον αριθμό των νημάτων που ενεργούν, συγκρίνοντας τα δεδομένα τους ανά ζεύγη σε κάθε φάση μέχρι να μείνει το επιθυμητό αποτέλεσμα. Συνήθως, τα δεδομένα αποθηκεύονται σ' έναν 1Δ πίνακα μέγεθους ίσου με τον αριθμό των νημάτων και το αποτέλεσμα αποθηκεύεται στην $1^{η}$ θέση του πίνακα. Η λειτουργία του όπως και η υλοποίηση του σε OpenMP αναφέρεται στο κεφάλαιο 3.2 Επεξήγηση παράλληλων τμημάτων του κώδικα.

3. Υλοποίηση

3.1 Αναφορά στις βασικές λειτουργίες του κώδικα

Οι βασικές λειτουργίες του κώδικα περιλαμβάνουν:

- **Δημιουργία πίνακα (create2DArray):**
  Η συνάρτηση δημιουργεί έναν πίνακα N x N με τυχαίες τιμές. Εξαρτάται από το αποτέλεσμα του προγράμματος-γεννήτρια που αναφέρεται και στο κεφάλαιο 2.3.2 Πρόγραμμα γεννήτρια για παραγωγή 2Δ πινάκων αν ο πίνακας θα είναι:
  - Αυστηρά διαγώνια δεσπόζων: Το απόλυτο κάθε διαγωνίου στοιχείου είναι μεγαλύτερο από το άθροισμα των απόλυτων τιμών των υπόλοιπων στοιχείων της αντίστοιχης γραμμής.
  - Μη αυστηρά διαγώνια δεσπόζων: Η παραπάνω ιδιότητα δεν ισχύει.
- **Εκτύπωση πίνακα (print2DArray):**
  Η συνάρτηση εκτυπώνει έναν πίνακα N x N σ' ένα αρχείο εξόδου. Κάθε γραμμή του πίνακα καταγράφεται σε μια γραμμή του αρχείου με τα στοιχεία να διαχωρίζονται από κενά.
- **Έλεγχος αυστηρά διαγώνιας δεσπόζουσας (Task a):**
  Εκτελείται παράλληλος έλεγχος αν ο πίνακας A είναι αυστηρά διαγώνια δεσπόζων. Για κάθε γραμμή, το άθροισμα των υπόλοιπων στοιχείων συγκρίνεται με το διαγώνιο στοιχείο. Αν οποιαδήποτε γραμμή παραβιάζει την ιδιότητα, το πρόγραμμα τερματίζεται.
- **Υπολογισμός μέγιστης διαγώνιας τιμής (Task b):**
  Βρίσκεται η μέγιστη τιμή $m = \max(|A_{ii}|)$, με παράλληλη επεξεργασία. Ο υπολογισμός γίνεται με τη χρήση της OpenMP και της οδηγίας reduction.
- **Δημιουργία νέου πίνακα B (Task c):**
  Ο πίνακας B υπολογίζεται παράλληλα ως εξής:
  - Αν $i = j$ τότε $B_{ij} = m$
  - Αν $i \neq j$ τότε $B_{ij} = m - |A_{ij}|$
- **Υπολογισμός ελάχιστης τιμής πίνακα B (Task d):**
  Το ελάχιστο στοιχείο στον πίνακα B υπολογίζεται με διάφορες τεχνικές:
  - **(d1)** Με τη χρήση της οδηγίας reduction της OpenMP
  - **(d2.1)** Χωρίς την χρήση της οδηγίας reduction, με μηχανισμό προστασίας κρίσιμης περιοχής
  - **(d2.2)** Χωρίς την χρήση της οδηγίας reduction, με χρήση αλγορίθμου δυαδικού δένδρου, όπου η σύγκριση και ο υπολογισμός ελαχίστων γίνονται σταδιακά σε φάσεις.
- **Μετρήσεις χρόνου:**
  Σε κάθε εργασία καταγράφεται:
  - Ο χρόνος έναρξης και λήξης κάθε εργασίας
  - Ο συνολικός χρόνος εκτέλεσης όλων των παράλληλων υπολογισμών

3.2 Επεξήγηση παράλληλων τμημάτων του κώδικα

Όπως αναφέρθηκε και στο κεφάλαιο 2.1 Περιγραφή της προσέγγισης που ακολουθήθηκε, για κάθε εργασία ορίστηκε ξεχωριστή περιοχή παράλληλης επεξεργασίας, ώστε να διακριθούν οι ακολουθιακές εργασίες που πρέπει να γίνονται πριν και μετά την παράλληλη επεξεργασία.Οι παράλληλες ενότητες εξηγούνται παρακάτω:

- **Έλεγχος αυστηρά διαγώνιας δεσπόζουσας (Task a):**
  - Κώδικας:

```
#pragma omp parallel shared(flag) private(i, j, loc_sum, loc_flag, loc_index)
{
        loc_flag = 1;
        #pragma omp for schedule(static, chunk)
        for (i = 0; i < N; i++)
        {
            loc_sum = 0;

            for (j = 0; j < N; j++)
                if (i != j)
                    loc_sum += abs(A[i][j]);
                else
                    loc_index = abs(A[i][i]);

            if (loc_index <= loc_sum)
                loc_flag = 0;
        }

        #pragma omp atomic
        flag *= loc_flag;
}
```

  - Λειτουργία:
    - Η **#pragma omp parallel** ενεργοποιεί παράλληλα νήματα
    - Η **#pragma omp for** διαμοιράζει τις επαναλήψεις του εξωτερικού βρόχου **for** στα νήματα με βάση την παράμετρο CZ που ορίσαμε
    - Η τοπική μεταβλητή *loc_flag* χρησιμοποιείται για να αποφεύγεται η πρόσβαση στην κοινή μεταβλητή *flag*, όπου παίρνει την τιμή 0 αν δεν ισχύει η ιδιότητα της διαγώνιας κυριαρχίας για τα στοιχεία κάποιου thread. Αυτή στην συνέχεια πολλαπλασιάζεται με την κοινή μεταβλητή flag και προσδιορίζει ότι θα έχει πάντα την τιμή 0 δηλώνοντας ότι ο πίνακας δεν είναι αυστηρά διαγώνια δεσπόζων
    - Η **#pragma omp atomic** αποτελεί μηχανισμός προστασίας κρίσιμης περιοχής και εξασφαλίζει ασφαλή πρόσβαση στην κοινή μεταβλητή *flag*

- **Υπολογισμός μέγιστης διαγώνιας τιμής (Task b):**
  - Κώδικας:

```
#pragma omp parallel default(shared) private(i)
{
        #pragma omp for schedule(static, chunk) reduction(max : m)
        for (i = 0; i < N; i++)
            if (A[i][i] > m)
                m = A[i][i];
}
```

  - Λειτουργία:
    - Η **#pragma omp for** διαμοιράζει CZ επαναλήψεις στα νήματα για τον υπολογισμό της μέγιστης τιμής.
    - Η οδηγία **reduction(max : m)** εξασφαλίζει ότι κάθε νήμα υπολογίζει ένα τοπικό μέγιστο και τα αποτελέσματα συγκεντρώνονται στο τέλος στην μεταβλητή *m*

- **Δημιουργία νέου πίνακα B (Task c):**
  - Κώδικας:

```
#pragma omp parallel default(shared) private(i, j)
{
        #pragma omp for schedule(static, chunk) collapse(2)
        for (i = 0; i < N; i++)
            for (j = 0; j < N; j++)
                if (i == j)
                    B[i][j] = m;
                else
                    B[i][j] = m - A[i][j];
}
```

  - Λειτουργία:
    - Η οδηγία **collapse(2)** συγχωνεύει τους δύο βρόχους **for** σ' έναν, επιτρέποντας την παράλληλη εκτέλεση του. Ισοδύναμα εκτελείται ο βρόχος:

```
for (i = 0; i < N * N; i++) { ... }
```

    - Οι τιμές του πίνακα B υπολογίζονται ανεξάρτητα, επομένως δεν απαιτούνται μηχανισμοί συγχρονισμού

- **Υπολογισμός ελάχιστης τιμής (Task d):**
  Το ελάχιστο στοιχείο του πίνακα B υπολογίζεται με τρεις διαφορετικές μεθόδους.

  - Με οδηγία reduction (Task d1):

- Κώδικας

```
#pragma omp parallel default(shared) private(i, j)
{
        #pragma omp for schedule(static, chunk) reduction(min : min_val)
        for (i = 0; i < N; i++)
            for (j = 0; j < N; j++)
                if (B[i][j] < min_val)
                    min_val = B[i][j];
}
```

- Λειτουργία
  - Η οδηγία **reduction(min : min_val)** εξασφαλίζει ότι κάθε νήμα βρίσκει το τοπικό ελάχιστο και τα αποτελέσματα συγκεντρώνονται στην μεταβλητή *min_val*

- Με προστασία κρίσιμης περιοχής (Task d2.1):
  - Κώδικας

```
#pragma omp parallel shared(min_val) private(i, j)
{

        #pragma omp for schedule(static, chunk)
        for (i = 0; i < N; i++)
            for (j = 0; j < N; j++)
                if (B[i][j] < min_val)
                {
                    #pragma omp critical (inc_min_val)
                    {
                        min_val = B[i][j];
                    }
                }
}
```

- Λειτουργία
  - Η οδηγία **#pragma omp critical** προστατεύει την κοινή μεταβλητή *min_val* από ταυτόχρονη τροποποίηση που θα οδηγούσε σε εσφαλμένα αποτελέσματα
  - Εξασφαλίζεται σωστή λειτουργία αλλά με κόστος απόδοσης λόγω σειριακής πρόσβασης στην κρίσιμη περιοχή

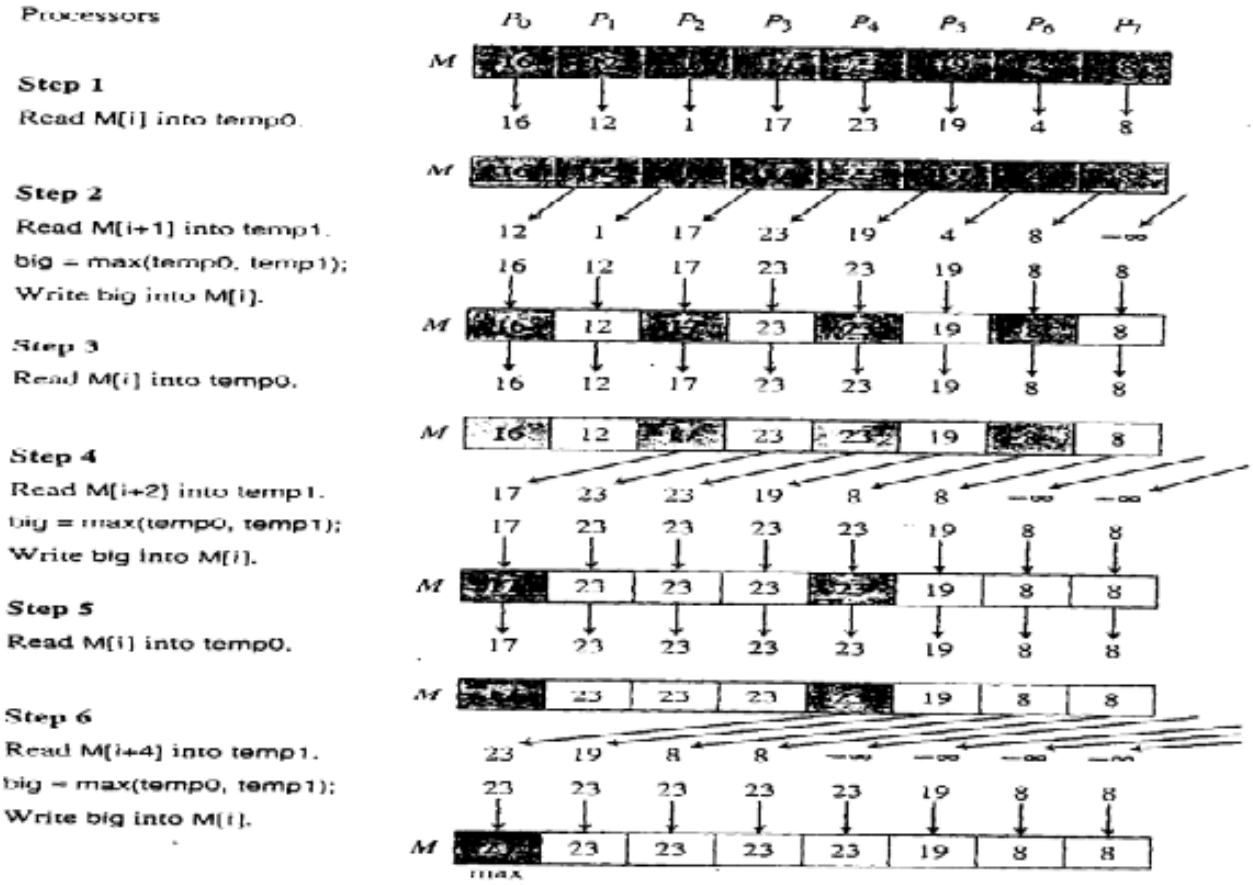
- Με αλγόριθμο δυαδικού δένδρου (Task d2.2):
  - Κώδικας

```
#pragma omp parallel default(shared) private(tid, i, j, incr, temp0, temp1,
loc_min)
{
```

```
        tid = omp_get_thread_num();
        loc_min = 1000000;

        #pragma omp for schedule(static, chunk)
        for (i = 0; i < N; i++)
            for (j = 0; j < N; j++)
                if (B[i][j] < loc_min)
                    loc_min = B[i][j];

        M[tid] = loc_min;

        #pragma omp barrier

        incr = 1;

        while (incr < T)
        {
            if (tid % (2 * incr) == 0 && tid + incr < T)
            {
                temp0 = M[tid];
                temp1 = M[tid + incr];
                loc_min = (temp0 <= temp1) ? temp0 : temp1;
                M[tid] = loc_min;
            }

            #pragma omp barrier

            incr = 2 * incr;
        }
}
```

- Λειτουργία
  - Κάθε νήμα υπολογίζει το τοπικό ελάχιστο στοιχείο του πίνακα B, περιορίζοντας την αναζήτηση του στα στοιχεία που έχουν κατανεμηθεί σ'αυτό χάρις στη χρήση της οδηγίας **#pragma omp for**.
  - Αποθηκεύει το τοπικό ελάχιστο στον πίνακα *M* στη θέση *M[tid]*, όπου *tid* το αναγνωριστικό του thread.
  - Ο αλγόριθμος εκτελείται σε φάσεις, όπου τα νήματα συγκρίνουν ζεύγη στοιχείων του πίνακα *M*, το μικρότερο στοιχείο διατηρείται στη θέση *M[tid]* και τα ενεργά νήματα μειώνονται σε κάθε φάση.
  - Στην $1^η$ φάση (*incr = 1*) τα νήματα με *tid = 0, 2, 4, …* συγκρίνουν τις τιμές *M[tid]* και *M[tid+1]*
  - Στην $2^η$ φάση (*incr = 2*) τα νήμα με *tid = 0, 4, 8, …* συγκρίνουν τις τιμές *M[tid]* και *M[tid+2]*.

- Η διαδικασία συνεχίζεται με τον δείκτη επανάληψης *incr* να διπλασιάζεται σε κάθε φάση.
- Η συνθήκη *tid % (2 * incr) == 0* εξασφαλίζει ότι το σωστό thread είναι υπεύθυνο γι' αυτή τη φάση
- Η συνθήκη *tid + incr < T* εξασφαλίζει την αποφυγή της πρόσβασης εκτός ορίων του πίνακα *M*
- Η οδηγία **#pragma omp barrier** διασφαλίζει ότι όλα τα νήματα ολοκληρώνουν την φάση τους πριν προχωρήσουν στην επόμενη
- Το ελάχιστο στοιχείο του πίνακα *B* βρίσκεται στη θέση *M[0]*.



Εικόνα 1. Υπολογισμός Max στο μοντέλο PRAM – EREW

Πηγή: Μάθημα 4ο – Πρότυπα Παράλληλου Υπολογισμού PRAM – Εισαγωγή στον Παράλληλο Υπολογισμό – σελ. 22-23

3.3 Περιγραφή της επικοινωνίας και του συγχρονισμού μεταξύ νημάτων

Η επικοινωνία και ο συγχρονισμός μεταξύ των νημάτων διασφαλίζεται με διάφορους μηχανισμούς που παρέχει η OpenMP. Παρακάτω περιγράφονται οι μέθοδοι που χρησιμοποιήθηκαν στο πρόγραμμα:

- **Κοινή και Ιδιωτική Μνήμη:**
  - Κοινές Μεταβλητές (Shared):
    Χρησιμοποιούνται απ' όλα τα νήματα ταυτόχρονα, π.χ. *flag, min_val, B* κλπ.
  - Ιδιωτικές Μεταβλητές (Private):
    Κάθε νήμα διαθέτει το δικό του αντίγραφο της μεταβλητής, π.χ. *loc_sum, loc_flag, tid* κλπ.
  - Χρήση στο πρόγραμμα:

```
#pragma omp parallel shared(flag) private(i, j, loc_sum, loc_flag, loc_index)
```

    - Οι κοινές μεταβλητές χρησιμοποιούνται για τη συλλογή αποτελεσμάτων από τα νήματα
    - Οι ιδιωτικές μεταβλητές εξασφαλίζουν ότι κάθε νήμα εκτελεί ανεξάρτητες υπολογιστικές πράξεις

- **Μηχανισμοί Συγχρονισμού**
  Η σωστή λειτουργία του προγράμματος βασίζεται σε συγχρονισμό για την αποφυγή συγκρούσεων:
  - Οδηγία Atomic:
    - Χρησιμοποιείται για αμοιβαίο αποκλεισμό για όταν πρόκειται να διασφαλιστεί η ατομικότητα απλών εντολών ενημέρωσης μιας κοινής μεταβλητής
    - Στο πρόγραμα χρησιμοποιείται για την ενημέρωση της κοινής μεταβλητής *flag* κατά τον έλεγχο αν ο πίνακας *A* είναι αυστηρά διαγώνια δεσπόζων (Task a):

```
#pragma omp atomic
flag *= loc_flag;
```

  - Κρίσιμη Περιοχή (Οδηγία Critical):
    - Χρησιμοποιείται όταν πολλά νήματα πρέπει να έχουν αποκλειστική πρόσβαση σε μια μεταβλητή
    - Στο πρόγραμμα εφαρμόζεται για την ενημέρωση της *min_val* κατά τον υπολογισμό της ελάχιστης τιμής στον πίνακα *B* (Task d2.1):

```
#pragma omp critical (inc_min_val)
{
      min_val = B[i][j];
}
```

  - Μηχανισμός Φράγματος (Barrier):
    - Ορίζει ένα σημείο συγχρονισμού όπου όλα τα νήματα πρέπει να φτάσουν πριν προχωρήσουν
    - Στο πρόγραμμα χρησιμοποιείται στον αλγόριθμο δυαδικού δένδρου για τον υπολογισμό του ελάχιστου στοιχείου του *B* (Task d2.2):

```
#pragma omp barrier
```

- Οδηγία Reduction:
  - Χρησιμοποιείται για τον υπολογισμό ενός συνοπτικού αποτελέσματος (π.χ. άθροισμα, μέγιστο, ελάχιστο κλπ.) συνδυάζοντας τις τιμές που υπολογίζουν τα επιμέρους νήματα
  - Στο πρόγραμμα χρησιμοποιείται για τον υπολογισμό του μέγιστου στοιχείου της διαγωνίου του *A* (Task b):

```
#pragma omp for schedule(static, chunk) reduction(max : m)
```

  - και του ελάχιστου στοιχείου του *B* (Task d1):

```
#pragma omp for schedule(static, chunk) reduction(min : min_val)
```

- **Διαμοιρασμός Εργασιών**
  Η διαμοίραση των επαναλήψεων βρόχου στα νήματα εξασφαλίζεται μέσω της οδηγίας **#pragma omp for**. Στο πρόγραμμα, χρησιμοποιείται η οδηγία **schedule(static, chunk)**, όπου:
  - Static: Οι επαναλήψεις μοιράζονται σταθερά στα νήματα με σειρά προτεραιότητας ως προς το αναγνωριστικό τους
  - Chunk: Ο αριθμός των επαναλήψεων που θα εκτελεί το κάθε νήμα
  - Στο πρόγραμμα χρησιμοποιείται σε κάθε εργασία:

```
#pragma omp for schedule(static, chunk)
```

4. Δοκιμές και Αποτελέσματα

4.1 Αναφορά των συνθηκών εκτέλεσης

Η εκτέλεση γίνεται για διαφορετικά μεγέθη πίνακα N, αριθμούς νημάτων T και διαμοιρασμό CZ επαναλήψεων ανά νήμα.

Η μεταγλώττιση του προγράμματος γίνεται μέσω command line σε περιβάλλον Linux, με τον compiler GNU **gcc** και τον διακόπτη **–fopenmp** για την σύνδεση του με την βιβλιοθήκη **omp.h.**

```
gcc –o omp omp.c -fopenmp
```

Η εκτέλεση του προγράμματος γίνεται μέσω command line σε περιβάλλον Linux και πρέπει ο χρήστης να περάσει παραμετρικά 2 αρχεία txt, ώστε να αποθηκευτούν αντίστοιχα οι πίνακες A και B. Ενδεικτική εντολή εκτέλεσης:

```
./omp A.txt B.txt
```

4.2 Παρουσίαση των αποτελεσμάτων σε μορφή κειμένου

Τα αποτελέσματα είναι αποθηκευμένα στον φάκελο Output και οι πίνακες A και B εκάστως στους αντίστοιχους φακέλους. Για εξοικονόμηση χώρου δεν παρουσιάζονται όλα τα αποτελέσματα σε μορφή κειμένου στην παρούσα τεκμηρίωση. Για να ανακατευθυνθείτε στα αρχεία εξόδου πατάτε στον σύνδεσμο που είναι στις παρακάτω υπο-κεφαλίδες και αντίστοιχα για τους πίνακες που το όνομα τους βρίσκεται τόσο στους αντίστοιχους φακέλους όσο και στα αποτελέσματα σε μορφή κειμένου.

*Λόγω του μεγάλου μεγέθους του **δεν** επιλέχθηκε να συμπεριληφθούν οι πίνακες διαστάσεων **10000 x 10000***

Το πρόγραμμα απαιτεί από τον χρήστη να περάσει παραμετρικά 2 .txt αρχεία εξόδου με όνομα της επιλογής του, στα οποία θα αποθηκευτούν οι πίνακες A και B. Σε περίπτωση που ο χρήστης δεν βάλει τον απαιτούμενο αριθμό παραμέτρων, το πρόγραμμα τερματίζεται και εμφανίζεται χαρακτηριστικό μήνυμα

4.2.1 Output_no_args.txt

```
Usage: ./omp A.txt B.txt
```

Αν ο πίνακας δεν είναι αυστηρά διαγώνια δεσπόζων, τότε το πρόγραμμα τερματίζεται πρόωρα.

4.2.2 Output_no_strict_diagonial.txt

```
Threads     : 1
Matrix size : 10 x 10
```

```
Chunk size  : 2
================================== [Task a.] ================================
Is A strictly diagonal dominant?
NO
The array has been stored in file A/A_no_strict_diagonial.txt

-----------------------------------------
Task a. finished in 0.000005 sec.
-----------------------------------------
================================================================================

-----------------------------------------
Parallel program finished in 0.000005 sec.
-----------------------------------------
```

Αν ο πίνακας είναι αυστηρά διαγώνια δεσπόζων, τότε εκτελούνται οι υπόλοιπες παράλληλες εργασίες.

4.2.3 Output_T1_N10_CZ2.txt

```
Threads     : 1
Matrix size : 10 x 10
Chunk size  : 2
================================== [Task a.] ================================
Is A strictly diagonal dominant?
YES
The array has been stored in file A/A_T1_N10_CZ2.txt
-----------------------------------------
Task a. finished in 0.000007 sec.
-----------------------------------------
================================================================================
================================== [Task b.] ================================
m = max(|Aii|) =>
m = 61
-----------------------------------------
Task b. finished in 0.000001 sec.
-----------------------------------------
================================================================================
================================== [Task c.] ================================
Bij = m - |Aij| for i <> j and Bij = m for i = j
The array has been stored in file B/B_T1_N10_CZ2.txt
-----------------------------------------
Task c. finished in 0.000001 sec.
-----------------------------------------
================================================================================
================================== [Task d1.] ================================
With reduction
m = min(|Bij|) =>
m = 51
```

```
----------------------------------------
Task d1. finished in 0.000003 sec.
----------------------------------------
================================================================================
================================== [Task d2.1] ================================
With critical section
m = min(|Bij|) =>
m = 51
----------------------------------------
Task d2.1 finished in 0.000001 sec.
----------------------------------------
================================================================================
================================== [Task d2.2] ================================
Binary Tree Algorithm
m = min(|Bij|) =>
m = 51
----------------------------------------
Task d2.2 finished in 0.000001 sec.
----------------------------------------
================================================================================


----------------------------------------
Parallel program finished in 0.000015 sec.
----------------------------------------
```

4.2.4 Output_T2_N16_CZ3.txt

```
Threads     : 2
Matrix size : 16 x 16
Chunk size  : 3
================================== [Task a.] ================================
Is A strictly diagonal dominant?
YES
The array has been stored in file A/A_T2_N16_CZ3.txt
----------------------------------------
Task a. finished in 0.000081 sec.
----------------------------------------
================================================================================
================================== [Task b.] ================================
m = max(|Aii|) =>
m = 102
----------------------------------------
Task b. finished in 0.000002 sec.
----------------------------------------
================================================================================
================================== [Task c.] ================================
Bij = m - |Aij| for i <> j and Bij = m for i = j
The array has been stored in file B/B_T2_N16_CZ3.txt
----------------------------------------
```

```
Task c. finished in 0.000002 sec.
----------------------------------------
================================================================================
================================== [Task d1.] ==================================
With reduction
m = min(|Bij|) =>
m = 92
----------------------------------------
Task d1. finished in 0.000002 sec.
----------------------------------------
================================================================================
================================== [Task d2.1] =================================
With critical section
m = min(|Bij|) =>
m = 92
----------------------------------------
Task d2.1 finished in 0.000002 sec.
----------------------------------------
================================================================================
================================== [Task d2.2] =================================
Binary Tree Algorithm
m = min(|Bij|) =>
m = 92
----------------------------------------
Task d2.2 finished in 0.000002 sec.
----------------------------------------
================================================================================


----------------------------------------
Parallel program finished in 0.000091 sec.
----------------------------------------
```

4.2.5 Output_T4_N25_CZ5.txt

```
Threads     : 4
Matrix size : 25 x 25
Chunk size  : 5
================================== [Task a.] ===================================
Is A strictly diagonal dominant?
YES
The array has been stored in file A/A_T4_N25_CZ5.txt
----------------------------------------
Task a. finished in 0.000112 sec.
----------------------------------------
================================================================================
================================== [Task b.] ===================================
m = max(|Aii|) =>
m = 175
----------------------------------------
```

```
Task b. finished in 0.000002 sec.
-----------------------------------------
=================================================================================
================================== [Task c.] ====================================
Bij = m - |Aij| for i <> j and Bij = m for i = j
The array has been stored in file B/B_T4_N25_CZ5.txt
-----------------------------------------
Task c. finished in 0.000003 sec.
-----------------------------------------
=================================================================================
================================== [Task d1.] ===================================
With reduction
m = min(|Bij|) =>
m = 165
-----------------------------------------
Task d1. finished in 0.000004 sec.
-----------------------------------------
=================================================================================
================================== [Task d2.1] ==================================
With critical section
m = min(|Bij|) =>
m = 165
-----------------------------------------
Task d2.1 finished in 0.000004 sec.
-----------------------------------------
=================================================================================
================================== [Task d2.2] ==================================
Binary Tree Algorithm
m = min(|Bij|) =>
m = 165
-----------------------------------------
Task d2.2 finished in 0.000004 sec.
-----------------------------------------
=================================================================================


-----------------------------------------
Parallel program finished in 0.000128 sec.
-----------------------------------------
```

4.2.6 Output_T8_N400_CZ20.txt

```
Threads     : 8
Matrix size : 400 x 400
Chunk size  : 20
================================== [Task a.] ====================================
Is A strictly diagonal dominant?
YES
The array has been stored in file A/A_T8_N400_CZ20.txt
-----------------------------------------
```

```
Task a. finished in 0.001359 sec.
-----------------------------------------
================================================================================
================================== [Task b.] ===================================
m = max(|Aii|) =>
m = 2300
-----------------------------------------
Task b. finished in 0.000556 sec.
-----------------------------------------
================================================================================
================================== [Task c.] ===================================
Bij = m - |Aij| for i <> j and Bij = m for i = j
The array has been stored in file B/B_T8_N400_CZ20.txt
-----------------------------------------
Task c. finished in 0.000718 sec.
-----------------------------------------
================================================================================
================================== [Task d1.] ==================================
With reduction
m = min(|Bij|) =>
m = 2290
-----------------------------------------
Task d1. finished in 0.000331 sec.
-----------------------------------------
================================================================================
================================== [Task d2.1] =================================
With critical section
m = min(|Bij|) =>
m = 2290
-----------------------------------------
Task d2.1 finished in 0.000315 sec.
-----------------------------------------
================================================================================
================================== [Task d2.2] =================================
Binary Tree Algorithm
m = min(|Bij|) =>
m = 2290
-----------------------------------------
Task d2.2 finished in 0.000374 sec.
-----------------------------------------
================================================================================

-----------------------------------------
Parallel program finished in 0.003653 sec.
-----------------------------------------
```

4.2.7 Output_T12_N1000_CZ100.txt

```
Threads     : 12
Matrix size : 1000 x 1000
Chunk size  : 100
================================== [Task a.] ================================
Is A strictly diagonal dominant?
YES
The array has been stored in file A/A_T12_N1000_CZ100.txt
-----------------------------------------
Task a. finished in 0.001395 sec.
-----------------------------------------
================================================================================
================================== [Task b.] ================================
m = max(|Aii|) =>
m = 5513
-----------------------------------------
Task b. finished in 0.000109 sec.
-----------------------------------------
================================================================================
================================== [Task c.] ================================
Bij = m - |Aij| for i <> j and Bij = m for i = j
The array has been stored in file B/B_T12_N1000_CZ100.txt
-----------------------------------------
Task c. finished in 0.001915 sec.
-----------------------------------------
================================================================================
================================== [Task d1.] ===============================
With reduction
m = min(|Bij|) =>
m = 5503
-----------------------------------------
Task d1. finished in 0.001270 sec.
-----------------------------------------
================================================================================
================================== [Task d2.1] ==============================
With critical section
m = min(|Bij|) =>
m = 5503
-----------------------------------------
Task d2.1 finished in 0.001271 sec.
-----------------------------------------
================================================================================
================================== [Task d2.2] ==============================
Binary Tree Algorithm
m = min(|Bij|) =>
m = 5503
-----------------------------------------
Task d2.2 finished in 0.001415 sec.
```

```
----------------------------------------
================================================================================


----------------------------------------
Parallel program finished in 0.007376 sec.
----------------------------------------
```

4.2.8 Output_T8_N10000_CZ1000.txt

```
Threads     : 8
Matrix size : 10000 x 10000
Chunk size  : 1000
================================== [Task a.] ================================
Is A strictly diagonal dominant?
YES
The array has been stored in file A/A_T8_N10000_CZ1000.txt
----------------------------------------
Task a. finished in 0.101806 sec.
----------------------------------------
================================================================================
================================== [Task b.] ================================
m = max(|Aii|) =>
m = 53490
----------------------------------------
Task b. finished in 0.000772 sec.
----------------------------------------
================================================================================
================================== [Task c.] ================================
Bij = m - |Aij| for i <> j and Bij = m for i = j
The array has been stored in file B/B_T8_N10000_CZ1000.txt
----------------------------------------
Task c. finished in 0.162018 sec.
----------------------------------------
================================================================================
================================== [Task d1.] ===============================
With reduction
m = min(|Bij|) =>
m = 53480
----------------------------------------
Task d1. finished in 0.081824 sec.
----------------------------------------
================================================================================
================================== [Task d2.1] ==============================
With critical section
m = min(|Bij|) =>
m = 53480
----------------------------------------
Task d2.1 finished in 0.075407 sec.
----------------------------------------
```

```
================================================================================
================================== [Task d2.2] ================================
Binary Tree Algorithm
m = min(|Bij|) =>
m = 53480
----------------------------------------
Task d2.2 finished in 0.080179 sec.
----------------------------------------
================================================================================


----------------------------------------
Parallel program finished in 0.502006 sec.
----------------------------------------
```

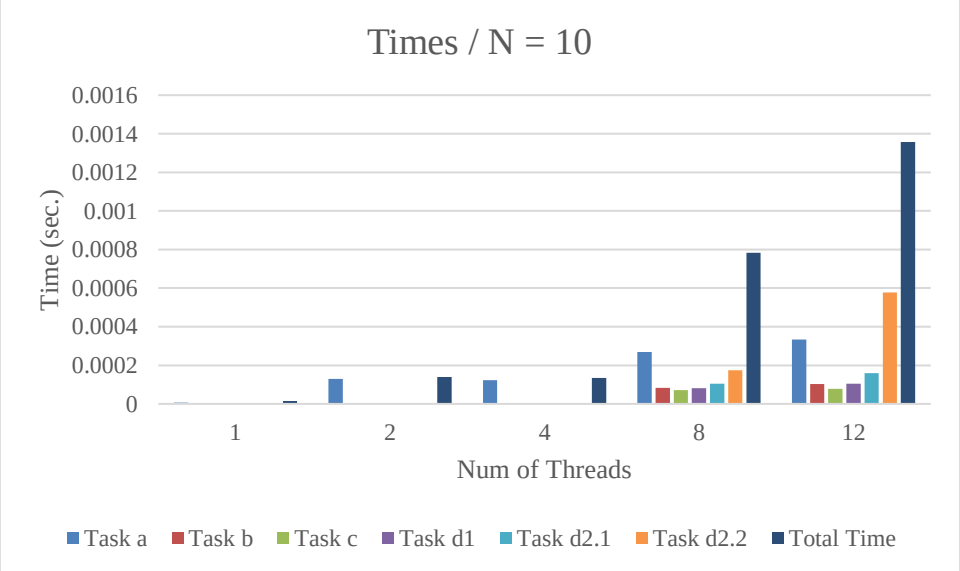
4.3 Ανάλυση της αποδοτικότητας

4.3.1 Χρόνοι εκτέλεσης των παράλληλων εργασιών

Οι χρόνοι που καταγράφηκαν για διαφορετικό αριθμό νημάτων T και μεγέθους πίνακα N παρουσιάζονται στα παρακάτω διαγράμματα. Συγκεκριμένα, τα δεδομένα των αποτελεσμάτων είναι:

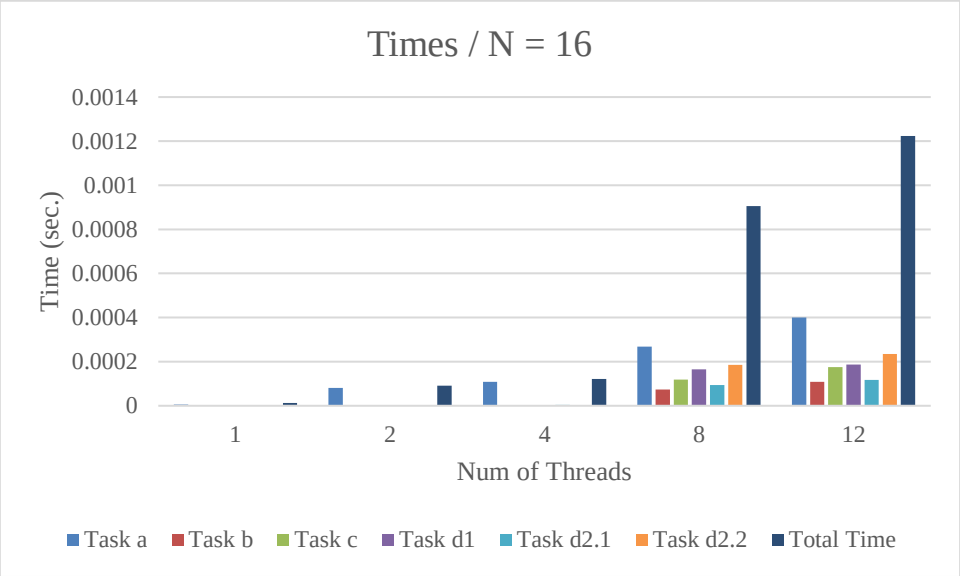
- **Αριθμός νημάτων:** 1, 2, 4, 8 και 12
- **Μέγεθος Πίνακα:** 10 x 10, 16 x 16, 25 x 25, 400 x 400, 1000 x 1000, 10000 x 10000

Threads	Task a	Task b	Task c	Task d1	Task d2.1	Task d2.2	Total Time
1	0,000007	0,000001	0,000001	0,000003	0,000001	0,000001	0,000015
2	0,00013	0,000002	0,000002	0,000002	0,000002	0,000002	0,000139
4	0,000123	0,000003	0,000002	0,000002	0,000003	0,000002	0,000134
8	0,000269	0,000083	0,000071	0,000082	0,000104	0,000175	0,000784
12	0,000333	0,000103	0,000079	0,000105	0,000159	0,000578	0,001358



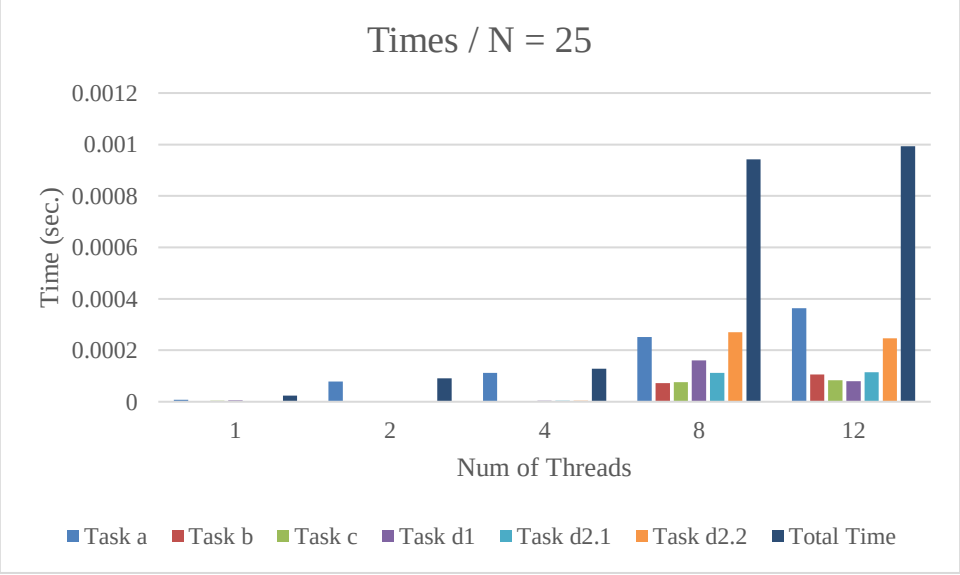
Εικόνα 2. Χρόνοι εκτέλεσης παράλληλων εργασιών για μέγεθος πίνακα 10 x 10

Threads	Task a	Task b	Task c	Task d1	Task d2.1	Task d2.2	Total Time
1	0,000006	0,000001	0,000002	0,000002	0,000001	0,000002	0,000013
2	0,000081	0,000002	0,000002	0,000002	0,000002	0,000002	0,000091
4	0,000109	0,000002	0,000003	0,000002	0,000004	0,000002	0,000122
8	0,000268	0,000073	0,000118	0,000165	0,000094	0,000186	0,000905
12	0,0004	0,000109	0,000176	0,000187	0,000117	0,000235	0,001224



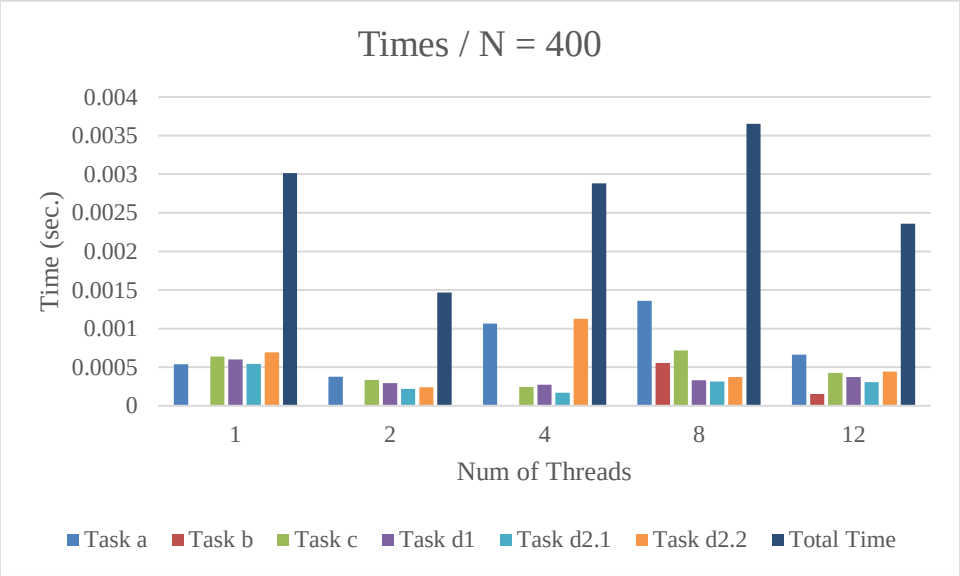
Εικόνα 3. Χρόνοι εκτέλεσης παράλληλων εργασιών για μέγεθος πίνακα 16 x 16

Threads	Task a	Task b	Task c	Task d1	Task d2.1	Task d2.2	Total Time
1	0,000008	0,000001	0,000004	0,000005	0,000003	0,000003	0,000024
2	0,000078	0,000002	0,000003	0,000003	0,000003	0,000003	0,000091
4	0,000112	0,000002	0,000003	0,000004	0,000004	0,000004	0,000128
8	0,000252	0,000072	0,000076	0,000161	0,000112	0,00027	0,000942
12	0,000363	0,000106	0,000083	0,00008	0,000115	0,000246	0,000993



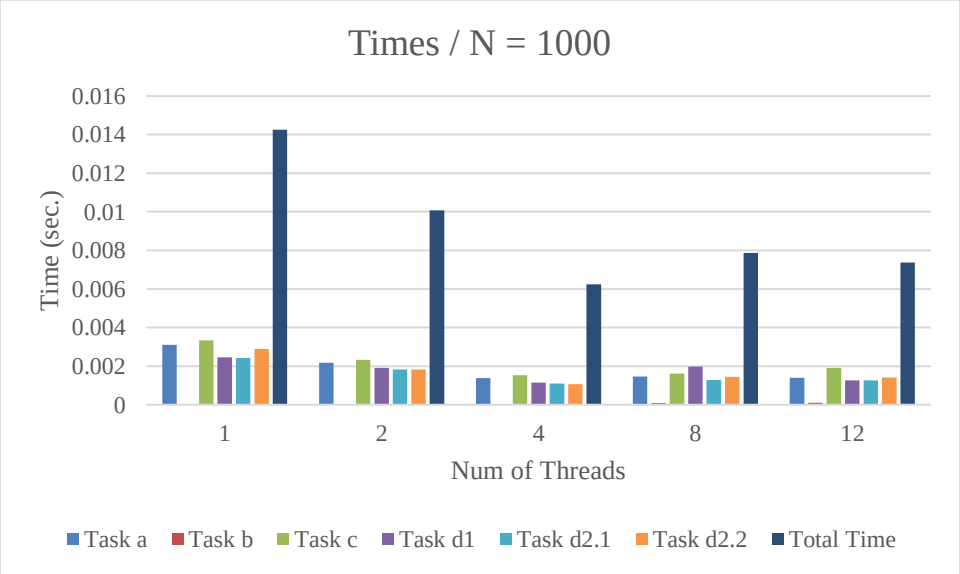
Εικόνα 4. Χρόνοι εκτέλεσης παράλληλων εργασιών για μέγεθος πίνακα 25 x 25

Threads	Task a	Task b	Task c	Task d1	Task d2.1	Task d2.2	Total Time
1	0,00054	0,000003	0,000637	0,000602	0,000542	0,00069	0,003013
2	0,000378	0,000002	0,000335	0,000294	0,000218	0,000239	0,001466
4	0,001065	0,000004	0,000242	0,000274	0,000171	0,001126	0,002882
8	0,001359	0,000556	0,000718	0,000331	0,000315	0,000374	0,003653
12	0,000661	0,000152	0,000425	0,000372	0,000307	0,000442	0,00236



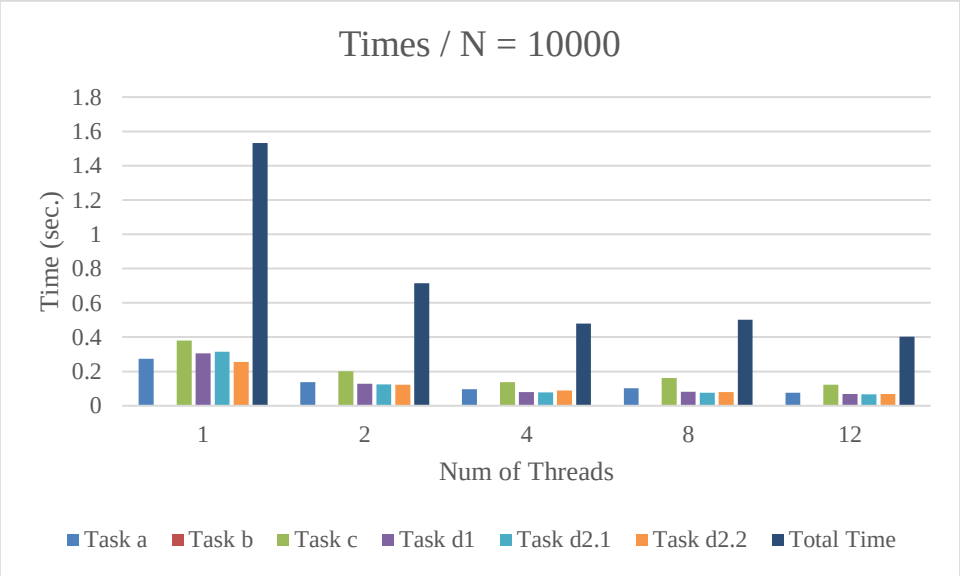
Εικόνα 5. Χρόνοι εκτέλεσης παράλληλων εργασιών για μέγεθος πίνακα 400 x 400

Threads	Task a	Task b	Task c	Task d1	Task d2.1	Task d2.2	Total Time
1	0,00311	0,000028	0,003345	0,002461	0,002426	0,00289	0,01426
2	0,00217	0,000017	0,002321	0,001914	0,001833	0,001825	0,01008
4	0,001383	0,00001	0,001525	0,001148	0,001104	0,001066	0,006236
8	0,001469	0,000089	0,001609	0,001971	0,001283	0,00144	0,007861
12	0,001395	0,000109	0,001915	0,00127	0,001271	0,001415	0,007376



Εικόνα 6. Χρόνοι εκτέλεσης παράλληλων εργασιών για μέγεθος πίνακα 1000 x 1000

Threads	Task a	Task b	Task c	Task d1	Task d2.1	Task d2.2	Total Time
1	0,274573	0,000663	0,379998	0,305948	0,314634	0,255239	1,531054
2	0,13722	0,000315	0,200705	0,127488	0,125518	0,123352	0,714598
4	0,095803	0,000218	0,137405	0,079142	0,078211	0,08829	0,479069
8	0,101806	0,000772	0,162018	0,081824	0,075407	0,080179	0,502006
12	0,0761	0,000303	0,122951	0,068925	0,066186	0,068788	0,403253



Εικόνα 7. Χρόνοι εκτέλεσης παράλληλων εργασιών για μέγεθος πίνακα 10000 x 10000

4.3.2 Επιταχύνσεις

Οι επιταχύνσεις υπολογίζονται από τον παρακάτω μαθηματικό τύπο:

$$\text{Speedup} = {}^{t_s}/_{t_n}$$

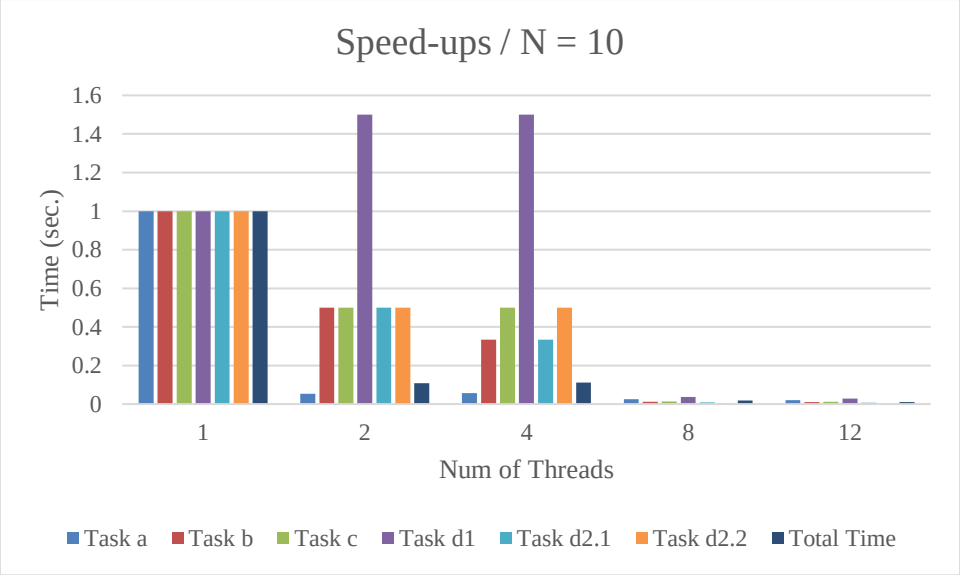
t_s: Ο χρόνος εκτέλεσης του ακολουθιακού προγράμματος, δηλαδή, εκτέλεση μ' ένα 1 νήμα
t_n: Ο χρόνος εκτέλεσης του παράλληλου προγράμματος, δηλαδή, εκτέλεση με n νήματα

Πηγή: Μάθημα 2ο – Παράλληλος Υπολογισμός – Εισαγωγή στον Παράλληλο Υπολογισμό – σελ. 7

Οι επιταχύνσεις παρουσιάζονται στα παρακάτω διαγράμματα. Συγκεκριμένα, τα δεδομένα των αποτελεσμάτων είναι:

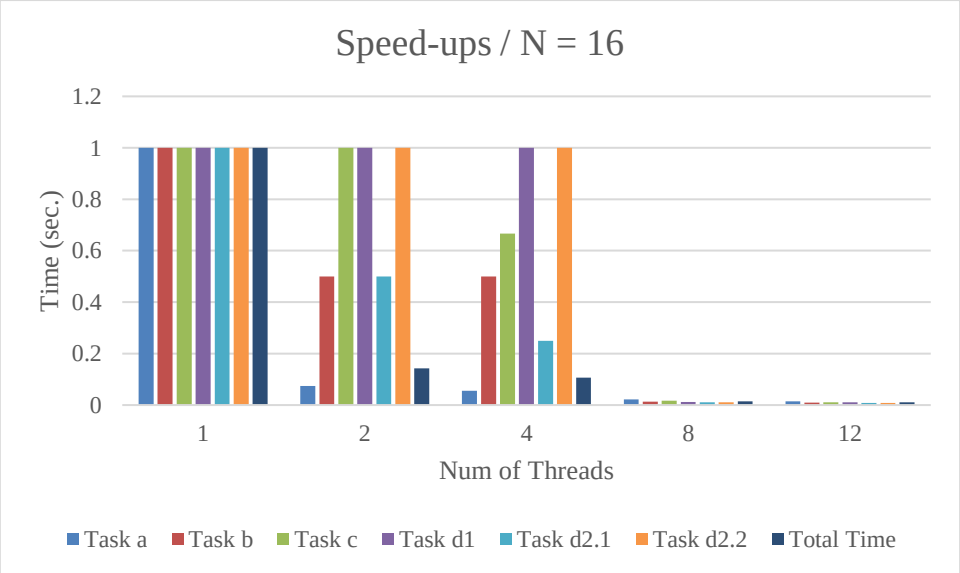
- **Αριθμός νημάτων:** 1, 2, 4, 8 και 12
- **Μέγεθος Πίνακα:** 10 x 10, 16 x 16, 25 x 25, 400 x 400, 1000 x 1000, 10000 x 10000

Threads	Task a	Task b	Task c	Task d1	Task d2.1	Task d2.2	Total Time
1	1	1	1	1	1	1	1
2	0,0538	0,5	0,5	1,5	0,5	0,5	0,1079
4	0,0569	0,3333	0,5	1,5	0,3333	0,5	0,1119
8	0,026	0,012	0,0141	0,0366	0,0096	0,0057	0,0191
12	0,021	0,0097	0,0127	0,0286	0,0063	0,0017	0,011



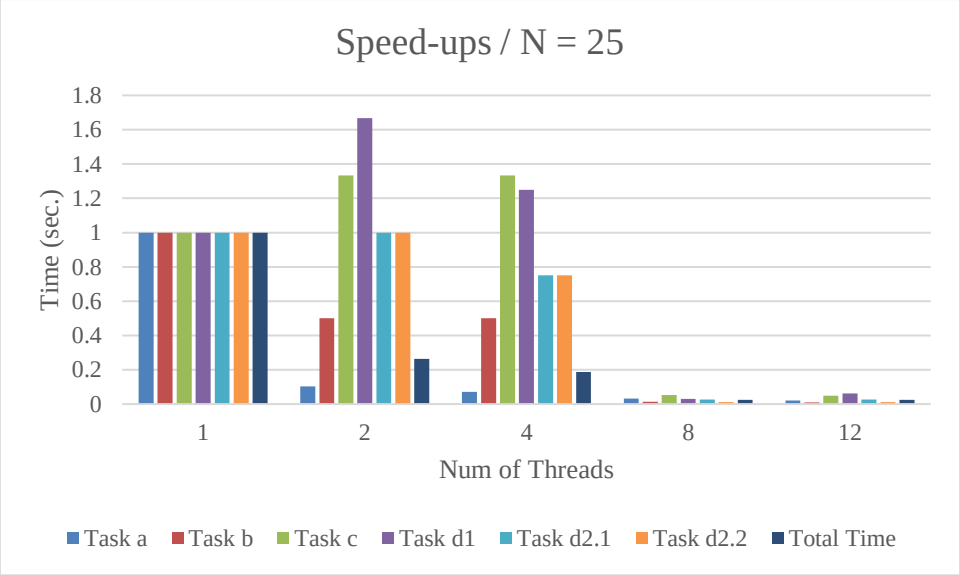
Εικόνα 8. Επιταχύνσεις για μέγεθος πίνακα 10 x 10

Threads	Task a	Task b	Task c	Task d1	Task d2.1	Task d2.2	Total Time
1	1	1	1	1	1	1	1
2	0,0741	0,5	1	1	0,5	1	0,1429
4	0,055	0,5	0,6667	1	0,25	1	0,1066
8	0,0224	0,0137	0,0169	0,0121	0,0106	0,0108	0,0144
12	0,015	0,0092	0,0114	0,0107	0,0085	0,0085	0,0106



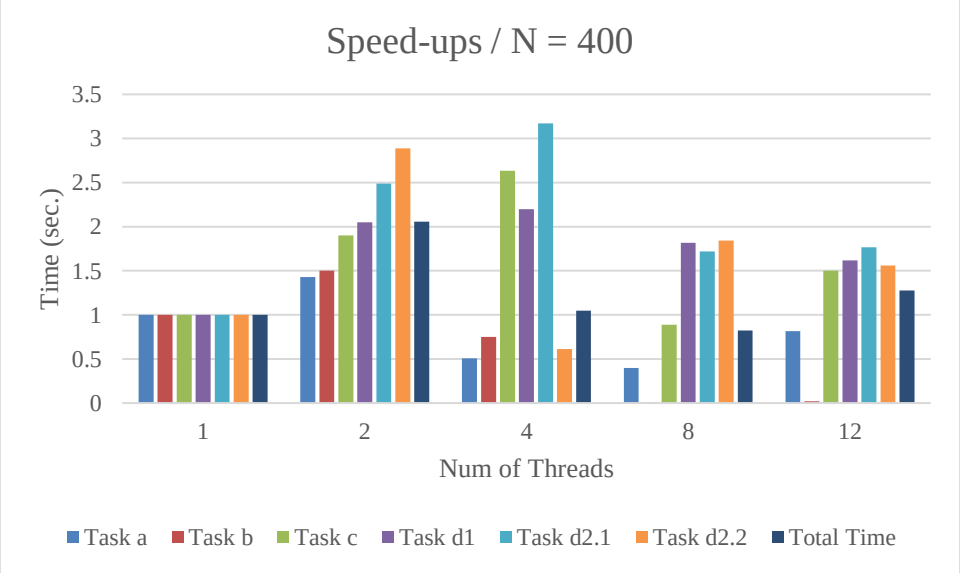
Εικόνα 9. Επιταχύνσεις για μέγεθος πίνακα 16 x 16

Threads	Task a	Task b	Task c	Task d1	Task d2.1	Task d2.2	Total Time
1	1	1	1	1	1	1	1
2	0,1026	0,5	1,3333	1,6667	1	1	0,2637
4	0,0714	0,5	1,3333	1,25	0,75	0,75	0,1875
8	0,0317	0,0139	0,0526	0,0311	0,0268	0,0111	0,0255
12	0,022	0,0094	0,0482	0,0625	0,0261	0,0122	0,0242



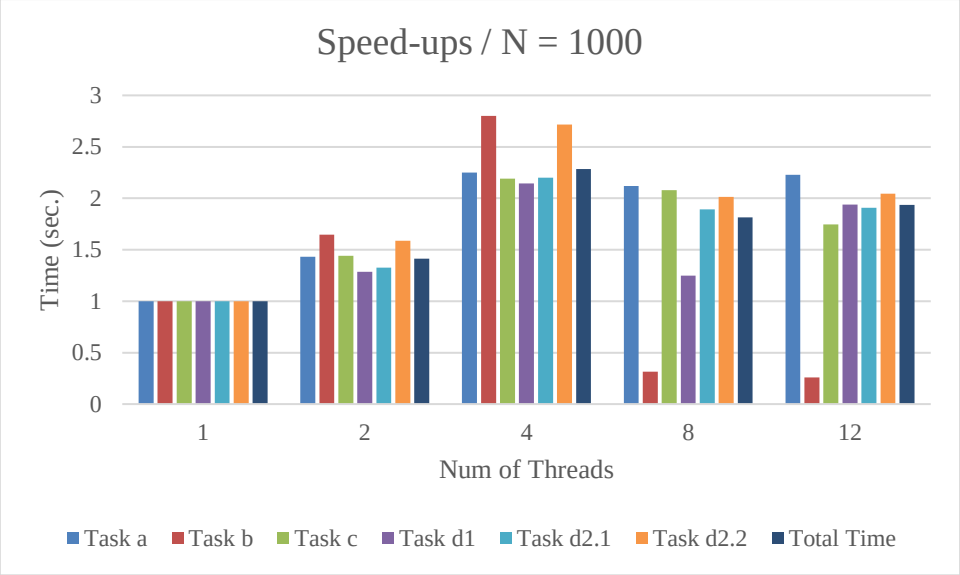
Εικόνα 10. Επιταχύνσεις για μέγεθος πίνακα 25 x 25

Threads	Task a	Task b	Task c	Task d1	Task d2.1	Task d2.2	Total Time
1	1	1	1	1	1	1	1
2	1,42857	1,5	1,9003	2,048	2,4876	2,8895	2,0551
4	0,507	0,75	2,6347	2,1985	3,1696	0,6134	1,046
8	0,3978	0,0054	0,887	1,8184	1,7206	1,843	0,8244
12	0,8168	0,0197	1,5	1,6161	1,7661	1,5581	1,277



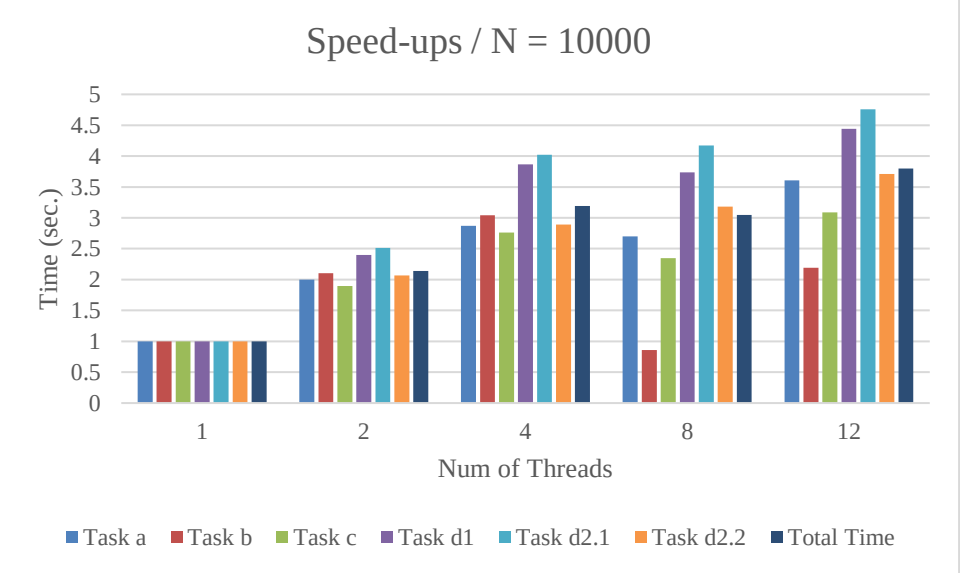
Εικόνα 11. Επιταχύνσεις για μέγεθος πίνακα 400 x 400

Threads	Task a	Task b	Task c	Task d1	Task d2.1	Task d2.2	Total Time
1	1	1	1	1	1	1	1
2	1,4332	1,6471	1,4415	1,2855	1,3252	1,586	1,4143
4	2,2505	2,8	2,1922	2,1455	2,1991	2,7162	2,2855
8	2,1186	0,3146	2,0789	1,2492	1,8927	2,0139	1,8141
12	2,2298	0,2578	1,7466	1,9378	1,9083	2,0432	1,9355



Εικόνα 12. Επιταχύνσεις για μέγεθος πίνακα 1000 x 1000

Threads	Task a	Task b	Task c	Task d1	Task d2.1	Task d2.2	Total Time
1	1	1	1	1	1	1	1
2	2,0003	2,1048	1,8942	2,399	2,5106	2,0681	2,1415
4	2,8689	3,0431	2,7642	3,8672	4,0203	2,8895	3,1943
8	2,6972	0,8581	2,3475	3,7375	4,1731	3,1817	3,0475
12	3,607	2,1898	3,0899	4,4404	4,7562	3,711	3,7967



Εικόνα 13. Επιταχύνσεις για μέγεθος πίνακα 10000 x 10000

4.3.3 Παρατηρήσεις

Παρατηρούμε ότι για μικρά N, η παραλληλία δεν αποδίδει καλύτερους χρόνους από την ακολουθιακή. Αυτό οφείλεται και στο γεγονός ότι η δημιουργία και η διαχείριση περισσότερων νημάτων μπορεί να επιφέρει πρόσθετη επιβάρυνση παρά λιγότερη. Ο χρόνος αυτός αντικρούει τα οφέλη του παράλληλου υπολογισμού, καθώς, τα νήματα ανταγωνίζονται για μνήμη λόγω του μικρού όγκου δεδομένων προκαλώντας καθυστερήσεις. Επίσης, υπάρχουν και καθυστερήσεις στον συγχρονισμό, ειδικά όταν υπάρχει προστασία κρίσιμης περιοχής, καθώς, ο ανταγωνισμός για πρόσβαση σε κάποια κοινή μεταβλητή προκαλεί επίσης σημαντικές καθυστερήσεις.

Για παράδειγμα, στα δεδομένα που συλλέξαμε για **N = 10**, παρατηρούμε ότι οι χρόνοι εκτέλεσης αυξάνονται όσο αυξάνουμε τον αριθμό των ενεργών νημάτων και γι' αυτό δεν πετυχαίνουμε επιταχύνσεις > 1 sec, καθώς, ο χρόνος εκτέλεσης των εργασιών με 1 νήμα είναι μικρότερος από τον αντίστοιχο με 2, 4, 8 ή και 12 νήματα.

Ωστόσο, τα οφέλη του παράλληλου υπολογισμού διακρίνονται για μεγάλα N, όπως παρατηρούμε στα δεδομένα που συλλέξαμε για **N = 10000**, καθώς, εκεί πετυχαίνουμε καλύτερους χρόνους με περισσότερα νήματα και συνεπώς και καλύτερες επιταχύνσεις. Αυτό οφείλεται στην αποδοτική κατανομή εργασιών, καθώς ο φόρτος εργασίας κατανέμεται αποκλειστικά στα νήματα, ελαχιστοποιώντας τον χρόνο αδράνειας και διασφαλίζοντας ότι τα νήματα συμβάλλουν στον υπολογισμό.

Συνεπώς, τα οφέλη του παράλληλου υπολογισμού διακρίνονται πιο αποδοτικά για μεγάλο όγκο δεδομένων, καθώς έτσι δεν έχουμε νήματα που παραμένουν ανενεργά.

5. Προβλήματα και Αντιμετώπιση

5.1 Αναφορά προβλημάτων

Πρόβλημα παρουσιάστηκε κυρίως κατά την ανάπτυξη του αλγορίθμου δυαδικού δένδρου (Task d2.2), καθώς, δεν παρατηρούσαμε τα ίδια αποτελέσματα με τις άλλες εργασίες Task d1 και Task d2.1 για τον υπολογισμό της ελάχιστης τιμής του πίνακα B. Μηνύματα σφάλματος για Segmentation Fault και παράλογων τιμών στον πίνακα M ήτανε από τα συχνότερα αποτελέσματα.

Επίσης, πρόβλημα εντοπίσαμε και στους χρόνους εκτέλεσης των παράλληλων εργασιών, όταν το περιβάλλον εργασίας ήταν το WSL (Windows Subsystem for Linux), καθώς, έπαιρνε παραπάνω χρόνος να ολοκληρωθούν οι εργασίες από το αναμενόμενο.

5.2 Λύσεις που δοκιμάστηκαν και εφαρμόστηκαν

Για τον αλγόριθμο δυαδικού δένδρου εφαρμόστηκε η προσθήκη της συνθήκης :

```
tid % (2 * incr) == 0 && tid + incr < threads
```

όπου εξασφαλίζουμε ότι τα υπεύθυνα thread θα πραγματοποιήσουν την εργασία σ' αυτή την φάση του αλγορίθμου και ότι δεν θα έχουν πρόσβαση εκτός των ορίων του πίνακα M που ήταν και η αφορμή για Segmentation Fault. Έτσι, καταφέραμε να συγχρονίσουμε τα threads και να μην ξεφεύγουν από τα όρια του πίνακα, καθώς, το κύριο πρόβλημα ήτανε ότι δεν συμμετείχαν τα σωστά threads σε κάθε φάση του αλγορίθμου και γι' αυτό βλέπαμε λανθασμένα αποτελέσματα.

Για του χρόνους, επιλέξαμε να αλλάξουμε περιβάλλον και να πάμε σε καθαρή διανομή Linux (Ubuntu) και τα αποτελέσματα ήτανε σαφώς καλύτερα.

WSL

```
Threads     : 1
Matrix size : 10 x 10
Chunk size  : 2
================================== [Task a.] ================================
Is A strictly diagonal dominant?
NO
The array has been stored in file A.txt

----------------------------------------
Task a. finished in 0.000316 sec.
----------------------------------------
================================================================================

----------------------------------------
Parallel program finished in 0.000316 sec.
```

Linux

```
Threads     : 1
Matrix size : 10 x 10
Chunk size  : 2
================================== [Task a.] ================================
Is A strictly diagonal dominant?
NO
The array has been stored in file A.txt

----------------------------------------
Task a. finished in 0.000005 sec.
----------------------------------------
================================================================================

----------------------------------------
Parallel program finished in 0.000005 sec.
```

Έπειτα από σχετική έρευνα στο διαδίκτυο ανακαλύψαμε τον λόγο που στο WSL οι χρόνοι εκτέλεσης παράλληλων εργασιών είναι μεγαλύτεροι:

OMP is incredibly slow in WSL2 due to filesystem boundary

https://github.com/JanDeDobbeleer/oh-my-posh/issues/1268

Το πρόβλημα απόδοσης με το OpenMP στην WSL2 μπορεί συχνά να αποδοθεί στον τρόπο με τον οποίο το WSL2 χειρίζεται το σύστημα αρχείων. Το WSL2 χρησιμοποιεί ένα εικονικοποιημένο περιβάλλον που αλληλεπιδρά με το σύστημα αρχείων των Windows και αυτή η αλληλεπίδραση μπορεί να εισάγει σημαντική καθυστέρηση όταν υπάρχουν συχνές λειτουργίες εισόδου/εξόδου. Αυτό γίνεται ιδιαίτερα αισθητό με εργαλεία όπως το OpenMP, όπου τα παράλληλα νήματα μπορεί να αλληλεπιδρούν σε μεγάλο βαθμό με το σύστημα αρχείων για αποθήκευση δεδομένων ή συγχρονισμό.

6. Συμπεράσματα

6.1 Ανακεφαλαίωση

Ανακεφαλαιώνοντας, τα αποτελέσματα έδειξαν ότι η παράλληλη υλοποίηση επιτυγχάνει σημαντική επιτάχυνση σε σύγκριση με την σειριακή εκτέλεση, ιδιαίτερα για μεγαλύτερες διαστάσεις πινάκων. Ωστόσο, παρατηρήθηκαν περιορισμοί στην κλιμάκωση, ιδιαίτερα σε περιπτώσεις που η εργασία ανά νήμα ήταν μικρή ή όταν ο αριθμός των νημάτων ξεπερνούσε τη διαθέσιμη υπολογιστική ισχύ του συστήματος.

Συνολικά, η εργασία ανέδειξε τη σημασία της σωστής επιλογής παραλληλισμού και τον αντίκτυπο του αριθμού νημάτων, της κατανομής φόρτου και του συγχρονισμού στην αποδοτικότητα του προγράμματος

ΠΑΡΑΛΛΗΛΑ ΣΥΣΤΗΜΑΤΑ

Σας ευχαριστώ για την προσοχή σας.