

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](#_Toc182877388)

[1.1 Σκοπός της άσκησης 3](#_Toc182877389)

[1.2 Συνοπτική περιγραφή του προβλήματος που επιλύεται 3](#_Toc182877390)

[2. Σχεδιασμός 3](#_Toc182877392)

[2.1 Περιγραφή της προσέγγισης που ακολουθήθηκε 3](#_Toc182877393)

[2.2 Ανάλυση της λογικής και των μεθοδολογιών 3](#_Toc182877394)

[2.3 Περιγραφή των δομών δεδομένων και των αλγορίθμων 3](#_Toc182877395)

[3. Υλοποίηση 3](#_Toc182877396)

[3.1 Αναφορά στις βασικές λειτουργίες του κώδικα 3](#_Toc182877397)

[3.2 Επεξήγηση σημαντικών τμημάτων του κώδικα 3](#_Toc182877398)

[3.3 Περιγραφή της επικοινωνίας και του συγχρονισμού μεταξύ νημάτων 3](#_Toc182877399)

[4. Δοκιμές και Αποτελέσματα 3](#_Toc182877400)

[4.1 Αναφορά των συνθηκών εκτέλεσης 3](#_Toc182877401)

[4.2 Παρουσίαση των αποτελεσμάτων σε μορφή κειμένου 3](#_Toc182877402)

[4.3 Σύγκριση ή ανάλυση της αποδοτικότητας 4](#_Toc182877403)

[5. Προβλήματα και Αντιμετώπιση 4](#_Toc182877404)

[5.1 Αναφορά προβλημάτων 4](#_Toc182877405)

[5.2 Λύσεις που δοκιμάστηκαν και εφαρμόστηκαν 4](#_Toc182877406)

[5.3 Αναφορά ζητημάτων που παρέμειναν άλυτα και πιθανοί λόγοι 4](#_Toc182877407)

[6. Συμπεράσματα 4](#_Toc182877408)

[6.1 Ανακεφαλαίωση 4](#_Toc182877409)

[6.2 Επισήμανση των περιορισμών της υλοποίησης 4](#_Toc182877410)

[6.3 Προτάσεις για βελτιώσεις 4](#_Toc182877411)

# 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:

1. την επεξεργασία ενός τετραγωνικού πίνακα Α (Ν 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
2. τον υπολογισμό του μέγιστου κατά απόλυτη τιμή στοιχείου της διαγωνίου του Α
3. τη δημιουργία ενός νέου πίνακα Β επίσης τετραγωνικού (N x N) με στοιχεία
4. τον υπολογισμό όπως του ελάχιστου κατά απόλυτη τιμή στοιχείου του Β με διάφορες μεθόδους όπως είναι ο αλγόριθμος δυαδικού δένδρου
5. 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](#_1.2_Συνοπτική_περιγραφή) 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:

1. **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
2. **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)
3. **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)
4. **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[N][N] | Στατικός τετραγωνικός 2Δ πίνακας μέγεθους Ν, όπου αποθηκεύονται ακέραιοι αριθμοί (θετικοί ή αρνητικοί) μέσω ενός προγράμματος-γεννήτρια που δημιουργεί είτε αυστηρά διαγώνιους δεσπόζων πίνακες είτε όχι |
| B[N][N] | Στατικός τετραγωνικός 2Δ πίνακας μέγεθους Ν, όπου αποθηκεύονται οι ακέραιοι αριθμοί που αναφέρονται στο [κεφάλαιο 1.2 Συνοπτική περιγραφή του προβλήματος που επιλύεται](#_1.2_Συνοπτική_περιγραφή) στην εργασία c) |
| M[T] | Στατικός 1Δ πίνακας μεγέθους Τ όσο του αριθμού των νημάτων, όπου χρησιμοποιείται για την υλοποίηση του αλγορίθμου δυαδικού δένδρου για την εύρεση της ελάχιστης τιμής του Β |
| i, j, k | Δείκτες επανάληψης για τους διαμοιρασμούς |
| rowSum | Άθροισμα των στοιχείων μιας γραμμής του πίνακα Α |
| chunk | Αριθμοί επαναλήψεων ανα thread |
| flag | Μεταβλητή «δείκτης» για να ελέγξουμε αν ο Α 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 Α |
| loc\_min | Τοπική μεταβλητή για κάθε thread που χρησιμοποιείται στον αλγόριθμο δυαδικού δένδρου και αποθηκεύει το μικρότερο στοιχείο από τα Μ[tid], M[tid+incr] |
| incr | Δείκτης επανάληψης για τον αλγόριθμο δυαδικού δένδρου |
| temp0 | Το στοιχείο Μ[tid] |
| temp1 | Το στοιχείο Μ[tid+incr] |
| m | Το μέγιστο στοιχείο της διαγωνίου του πίνακα Α |
| min\_val | Το ελάχιστο στοιχείο του πίνακα Β |
| **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 Α |
| fpB | Αρχείο εξόδου για την αποθήκευση του πίνακα Β |

### 2.3.2 Πρόγραμμα γεννήτρια για παραγωγή 2Δ πινάκων

Ο αλγόριθμος αυτός δημιουργεί τον τετραγωνικό πίνακα Α διαστάσεων Ν 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
    - 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
    - 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 1η θέση του πίνακα. Η λειτουργία του όπως και η υλοποίηση του σε OpenMP αναφέρεται στο [κεφάλαιο 3.2 Επεξήγηση παράλληλων τμημάτων του κώδικα](#_3.2_Επεξήγηση_παράλληλων).

# 3. Υλοποίηση

## 3.1 Αναφορά στις βασικές λειτουργίες του κώδικα

Οι βασικές λειτουργίες του κώδικα περιλαμβάνουν:

* **Δημιουργία πίνακα (create2DArray):**

Η συνάρτηση δημιουργεί έναν πίνακα Ν 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](#_2.3.2_Πρόγραμμα_γεννήτρια) 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 Ν 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 Α 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 , με παράλληλη επεξεργασία. Ο υπολογισμός γίνεται με τη χρήση της OpenMP και της οδηγίας reduction.

* **Δημιουργία νέου πίνακα Β (Task c):**

Ο πίνακας Β υπολογίζεται παράλληλα ως εξής:

* **Υπολογισμός ελάχιστης τιμής πίνακα Β (Task d):**

Το ελάχιστο στοιχείο στον πίνακα Β 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](#_2.1_Περιγραφή_της), 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 Β (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 Β υπολογίζονται ανεξάρτητα, επομένως δεν απαιτούνται μηχανισμοί συγχρονισμού
* **Υπολογισμός ελάχιστης τιμής (Task d):**

Το ελάχιστο στοιχείο του πίνακα Β 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;

            }

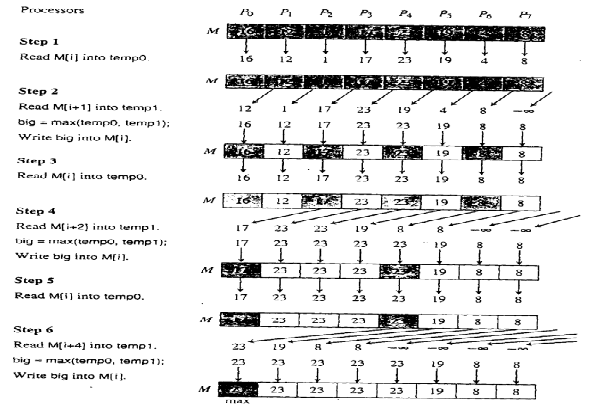
            #pragma omp barrier

          incr = 2 \* incr;

        }

}

* + - Λειτουργία
      * Κάθε νήμα υπολογίζει το τοπικό ελάχιστο στοιχείο του πίνακα Β, 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 *Μ* στη θέση *Μ[tid]*, όπου *tid* το αναγνωριστικό του thread.
      * Ο αλγόριθμος εκτελείται σε φάσεις, όπου τα νήματα συγκρίνουν ζεύγη στοιχείων του πίνακα *Μ,* το μικρότερο στοιχείο διατηρείται στη θέση *M[tid]* και τα ενεργά νήματα μειώνονται σε κάθε φάση.
      * Στην 1η φάση (*incr = 1*) τα νήματα με *tid = 0, 2, 4, …* συγκρίνουν τις τιμές *M[tid]* και *M[tid+1]*
      * Στην 2η 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 *Μ*
      * Η οδηγία **#pragma omp barrier** διασφαλίζει ότι όλα τα νήματα ολοκληρώνουν την φάση τους πριν προχωρήσουν στην επόμενη
      * Το ελάχιστο στοιχείο του πίνακα *Β* βρίσκεται στη θέση *Μ[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 *Α* 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 *Β* (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 *Β* (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 *Α* (Task b):

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

* + - και του ελάχιστου στοιχείου του *Β* (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 Ν, αριθμούς νημάτων T και διαμοιρασμό CZ επαναλήψεων ανά νήμα.

Η μεταγλώττιση του προγράμματος γίνεται μέσω command line σε περιβάλλον Linux ή WSL (Windows Subsystem 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 ή WSL (Windows Subsystem Linux) και πρέπει ο χρήστης να περάσει παραμετρικά 2 αρχεία txt, ώστε να αποθηκευτούν αντίστοιχα οι πίνακες Α και Β. Ενδεικτική εντολή εκτέλεσης:

./omp A.txt B.txt

## 4.2 Παρουσίαση των αποτελεσμάτων σε μορφή κειμένου

*Τα αποτελέσματα είναι αποθηκευμένα στον φάκελο* [*Output*](Output) *και οι πίνακες* [*Α*](A) *και* [*Β*](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*](#_4.2_Παρουσίαση_των) *και αντίστοιχα για τους πίνακες που το όνομα τους βρίσκεται τόσο στους αντίστοιχους φακέλους όσο και στα αποτελέσματα σε μορφή κειμένου.*

Το πρόγραμμα απαιτεί από τον χρήστη να περάσει παραμετρικά 2 .txt αρχεία εξόδου με όνομα της επιλογής του, στα οποία θα αποθηκευτούν οι πίνακες Α και Β. 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](Output/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](Output/Output_no_strict_diagonial.txt)

Threads     : 4

Matrix size : 16 x 16

Chunk size  : 4

=================================== [Task a.] ================================

Is A strictly diagonal dominant?

NO

The array has been stored in file A/[A\_no\_strict\_diagonial.txt](A/A_no_strict_diagonial.txt)

--------------------------------------------

Task a. finished in 0.000335 sec.

--------------------------------------------

==============================================================================

--------------------------------------------

Parallel program finished in 0.000335 sec.

--------------------------------------------

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

### 4.2.3 [Output\_T1\_N10\_CZ2.txt](Output/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](A/A_T1_N10_CZ2.txt)

--------------------------------------------

Task a. finished in 0.000019 sec.

--------------------------------------------

==============================================================================

================================= [Task b.] ==================================

m = max(|Aii|) =>

m = 58

--------------------------------------------

Task b. finished in 0.000068 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](B/B_T1_N10_CZ2.txt)

--------------------------------------------

Task c. finished in 0.000002 sec.

--------------------------------------------

==============================================================================

=================================== [Task d1.] ===============================

With reduction

m = min(|Bij|) =>

m = 48

--------------------------------------------

Task d1. finished in 0.000004 sec.

--------------------------------------------

==============================================================================

=================================== [Task d2.1] ==============================

With critical section

m = min(|Bij|) =>

m = 48

--------------------------------------------

Task d2.1 finished in 0.000012 sec.

--------------------------------------------

==============================================================================

=================================== [Task d2.2] ==============================

Binary Tree Algorithm

m = min(|Bij|) =>

m = 48

--------------------------------------------

Task d2.2 finished in 0.000003 sec.

--------------------------------------------

==============================================================================

--------------------------------------------

Parallel program finished in 0.000109 sec.

--------------------------------------------

### 4.2.4 [Output\_T2\_N16\_CZ3.txt](Output/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](A/A_T2_N16_CZ3.txt)

--------------------------------------------

Task a. finished in 0.000186 sec.

--------------------------------------------

==============================================================================

=================================== [Task b.] ================================

m = max(|Aii|) =>

m = 109

--------------------------------------------

Task b. finished in 0.000003 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](B/B_T2_N16_CZ3.txt)

--------------------------------------------

Task c. finished in 0.000004 sec.

--------------------------------------------

==============================================================================

=================================== [Task d1.] ===============================

With reduction

m = min(|Bij|) =>

m = 99

--------------------------------------------

Task d1. finished in 0.000020 sec.

--------------------------------------------

==============================================================================

=================================== [Task d2.1] ==============================

With critical section

m = min(|Bij|) =>

m = 99

--------------------------------------------

Task d2.1 finished in 0.000015 sec.

--------------------------------------------

==============================================================================

=================================== [Task d2.2] ==============================

Binary Tree Algorithm

m = min(|Bij|) =>

m = 99

--------------------------------------------

Task d2.2 finished in 0.000005 sec.

--------------------------------------------

==============================================================================

--------------------------------------------

Parallel program finished in 0.000234 sec.

--------------------------------------------

### 4.2.5 [Output\_T4\_N25\_CZ5.txt](Output/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](A/A_T2_N25_CZ5.txt)

--------------------------------------------

Task a. finished in 0.000285 sec.

--------------------------------------------

==============================================================================

=================================== [Task b.] ================================

m = max(|Aii|) =>

m = 147

--------------------------------------------

Task b. finished in 0.000051 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](B/B_T2_N25_CZ5.txt)

--------------------------------------------

Task c. finished in 0.000005 sec.

--------------------------------------------

==============================================================================

=================================== [Task d1.] ===============================

With reduction

m = min(|Bij|) =>

m = 137

--------------------------------------------

Task d1. finished in 0.000008 sec.

--------------------------------------------

==============================================================================

=================================== [Task d2.1] ==============================

With critical section

m = min(|Bij|) =>

m = 137

--------------------------------------------

Task d2.1 finished in 0.000011 sec.

--------------------------------------------

==============================================================================

=================================== [Task d2.2] ==============================

Binary Tree Algorithm

m = min(|Bij|) =>

m = 137

--------------------------------------------

Task d2.2 finished in 0.000004 sec.

--------------------------------------------

==============================================================================

--------------------------------------------

Parallel program finished in 0.000364 sec.

--------------------------------------------

### 4.2.6 [Output\_T8\_N400\_CZ20.txt](Output/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](A/A_T8_N400_CZ20.txt)

--------------------------------------------

Task a. finished in 0.000766 sec.

--------------------------------------------

==============================================================================

=================================== [Task b.] ================================

m = max(|Aii|) =>

m = 2303

--------------------------------------------

Task b. finished in 0.000082 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](B/B_T8_N400_CZ20.txt)

--------------------------------------------

Task c. finished in 0.000261 sec.

--------------------------------------------

==============================================================================

=================================== [Task d1.] ===============================

With reduction

m = min(|Bij|) =>

m = 2293

--------------------------------------------

Task d1. finished in 0.000187 sec.

--------------------------------------------

==============================================================================

=================================== [Task d2.1] ==============================

With critical section

m = min(|Bij|) =>

m = 2293

--------------------------------------------

Task d2.1 finished in 0.000185 sec.

--------------------------------------------

==============================================================================

=================================== [Task d2.2] ==============================

Binary Tree Algorithm

m = min(|Bij|) =>

m = 2293

--------------------------------------------

Task d2.2 finished in 0.000193 sec.

--------------------------------------------

==============================================================================

--------------------------------------------

Parallel program finished in 0.001673 sec.

--------------------------------------------

### 4.2.7 [Output\_T12\_N1000\_CZ100.txt](Output/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](A/A_T12_N1000_CZ100.txt)

--------------------------------------------

Task a. finished in 0.001906 sec.

--------------------------------------------

==============================================================================

=================================== [Task b.] ================================

m = max(|Aii|) =>

m = 5481

--------------------------------------------

Task b. finished in 0.000130 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](B/B_T12_N1000_CZ100.txt)

--------------------------------------------

Task c. finished in 0.001282 sec.

--------------------------------------------

==============================================================================

=================================== [Task d1.] ===============================

With reduction

m = min(|Bij|) =>

m = 5471

--------------------------------------------

Task d1. finished in 0.001028 sec.

--------------------------------------------

==============================================================================

=================================== [Task d2.1] ==============================

With critical section

m = min(|Bij|) =>

m = 5471

--------------------------------------------

Task d2.1 finished in 0.000869 sec.

--------------------------------------------

==============================================================================

=================================== [Task d2.2] ==============================

Binary Tree Algorithm

m = min(|Bij|) =>

m = 5471

--------------------------------------------

Task d2.2 finished in 0.000917 sec.

--------------------------------------------

==============================================================================

--------------------------------------------

Parallel program finished in 0.006132 sec.

--------------------------------------------

## 4.3 Ανάλυση της αποδοτικότητας

### 4.3.1 Χρόνοι εκτέλεσης των παράλληλων εργασιών

Οι χρόνοι που καταγράφηκαν για διαφορετικό αριθμό νημάτων Τ και μεγέθους πίνακα Ν 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

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

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

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

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

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

### 4.3.2 Επιταχύνσεις

Οι επιταχύνσεις υπολογίζονται από τον παρακάτω μαθηματικό τύπο:

Ο χρόνος εκτέλεσης του ακολουθιακού προγράμματος, δηλαδή, εκτέλεση μ’ ένα 1 νήμα

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

**Εικόνα 7.** Επιταχύνσεις για μέγεθος πίνακα 10 x 10

**Εικόνα 8.** Επιταχύνσεις για μέγεθος πίνακα 16 x 16

**Εικόνα 9.** Επιταχύνσεις για μέγεθος πίνακα 25 x 25

**Εικόνα 10.** Επιταχύνσεις για μέγεθος πίνακα 400 x 400

**Εικόνα 11.** Επιταχύνσεις για μέγεθος πίνακα 1000 x 1000

### 4.3.3 Παρατηρήσεις

# 5. Προβλήματα και Αντιμετώπιση

## 5.1 Αναφορά προβλημάτων

Αναφέρονται δυσκολίες που προέκυψαν κατά την ανάπτυξη και εκτέλεση του προγράμματος, όπως συγχρονισμός νημάτων ή αποδοτικότητα.

## 5.2 Λύσεις που δοκιμάστηκαν και εφαρμόστηκαν

Παρουσιάζονται οι στρατηγικές που υιοθετήθηκαν για την αντιμετώπιση των προβλημάτων, όπως βελτιστοποίηση κώδικα και χρήση διαφορετικών μεθόδων συγχρονισμού.

## 5.3 Αναφορά ζητημάτων που παρέμειναν άλυτα και πιθανοί λόγοι

Εντοπίζονται περιοχές όπου η βέλτιστη λύση δεν ήταν εφικτή και αναλύονται οι πιθανές αιτίες.

# 6. Συμπεράσματα

## 6.1 Ανακεφαλαίωση

Παρουσιάζεται μια σύντομη περίληψη των βημάτων που ολοκληρώθηκαν και των αποτελεσμάτων που επιτεύχθηκαν.

## 6.2 Επισήμανση των περιορισμών της υλοποίησης

Αναφέρονται τυχόν αδυναμίες ή περιορισμοί της παρούσας προσέγγισης.

## 6.3 Προτάσεις για βελτιώσεις

## 

Παραντίθενται για ιδέες για μελλοντικές βελτιώσεις, όπως χρήση πιο προηγμένων μεθόδων ή βιβλιοθηκών.



Σας ευχαριστώ για την προσοχή σας.

