

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

# ΕΡΓΑΣΙΑ 2Α

# Multisort

**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

## 1.1 Σκοπός της άσκησης

Ο σκοπός της άσκησης είναι η υλοποίηση και η αξιολόγηση του παράλληλου αλγορίθμου ταξινόμησης 1Δ πίνακα **multisort**, όπου αποτελεί μια εναλλακτική επιλογή του αλγορίθμου **mergesort**. Ο παράλληλος υπολογισμός γίνεται 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 την ανάθεση εργασιών (tasks) στα νήματα με την ιδιότητα της αναδρομής.

Τέλος, η άσκηση αποσκοπεί στο να αναδείξει και τους χρόνους εκτέλεσης του παράλληλου αλγορίθμου, 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

Ο αλγόριθμος επιλύει το πρόβλημα της ταξινόμησης ενός 1Δ πίνακα Ν ακεραίων Α[1...Ν-1], χρησιμοποιώντας το OpenMP για την παράλληλη εκτέλεση του. Η μέθοδος αυτή στηρίζεται στην τεχνική **διαίρει και βασίλευε**, όπου το πρόβλημα διασπάται σε υποπροβλήματα και υπολογίζονται τοπικά οι λύσεις των υποπροβλημάτων, ώστε να συγκεντρωθούν στο τέλος για την βέλτιστη δυνατή. Άλλωστε, γι’ αυτό αποτελεί εναλλακτική επιλογή του **mergesort**, όπου ο πίνακας σπάει σε υπο-πίνακες, ταξινομούνται τοπικά και στο τέλος συγχωνεύονται σ’ έναν ενιαίο ο οποίος είναι και ο τελικός ταξινομημένος.

O αλγόριθμος **multisort** διαχωρίζει αρχικά την προς ταξινόμηση ακολουθία σε τέσσερα ισομεγέθη τμήματα, και συνεχίζει εφαρμόζοντας αναδρομικά την παραπάνω διαδικασία διαχωρισμού σε κάθε τμήμα. Με την ολοκλήρωση κάθε αναδρομικής κλήσης, τα τέσσερα επιμέρους τμήματα τα οποία επιστρέφονται στο κυρίως σώμα ταξινομημένα, συγχωνεύονται σε μία ενιαία ταξινομημένη ακολουθία σε δύο βήματα (πρώτα συγχωνεύονται – παράλληλα – ανά δύο σε δύο τμήματα διπλάσιου μεγέθους, και στη συνέχεια τα δύο εναπομείναντα τμήματα συγχωνεύονται μεταξύ τους)

Όσον αφορά την απόδοση, λαμβάνονται υπόψιν οι εξής παράμετροι:

* Μέγεθος πίνακα N
* Αριθμός νημάτων T
* Όριο πίνακα LIMIT όπου σημάνει και την λήξη του παράλληλου υπολογισμού, όπου εκτελείται ο γνωστός σειριακός αλγόριθμος ταξινόμησης **quicksort**

# 2. Σχεδιασμός

## 2.1 Περιγραφή της προσέγγισης που ακολουθήθηκε

Η υλοποίηση βασίζεται στην κατανομή των υπολογισμών σε tasks χρησιμοποιώντας το OpenMP και αναδρομή για την επίτευξη παράλληλης επεξεργασίας, καθώς, 1 task αναλαμβάνεται από μόνο 1 thread. Η προσέγγιση διαχωρίζεται σε διακριτές φάσεις:

* **Ανάθεση εργασιών (tasks) στα νήματα:** Με χρήση OpenMP και αναδρομή το κάθε task αναλαμβάνεται από κάποιο διαθέσιμο ενεργό thread.
* **Αξιοποίηση της δυνατότητας του OpenMP:** Στην κατανομή εργασιών και στον συγχρονισμό με οδηγίες wait.
* **Ανάλυση της δομής του προβλήματος:** Τα δεδομένα οργανώνονται ώστε να διευκολύνουν την παράλληλη επεξεργασία, ελαχιστοποιώντας την επικοινωνία μεταξύ νημάτων.

## 2.2 Ανάλυση της λογικής και των μεθοδολογιών

Η λογική βασίζεται στα εξής:

1. **Διαχωρισμός του πίνακα Α σε 4 ισομεγέθη τμήματα:** Η 1η φάση του αλγορίθμου είναι ο πίνακας να διαχωριστεί σε 4 ισομεγέθη τμήματα. Αυτό επιτυγχάνεται με τον υπολογισμό των κατάλληλων δεικτών, ώστε κάθε τμήμα να έχει το ίδιο περίπου μέγεθος Ν/4
2. **Αναδρομική κλήση της multisort για τα 4 τμήματα με χρήση OpenMP tasks και παράλληλη εκτέλεση από 4 διαθέσιμα νήματα:** Η 2η φάση του αλγορίθμου είναι η αναδρομική κλήση για καθένα από τα 4 τμήματα του πίνακα με χρήση OpenMP tasks που αναθέτουν κάθε αναδρομική κλήση σε ξεχωριστό νήμα. Έτσι, η ταξινόμηση των 4 τμημάτων πραγματοποιείται ταυτόχρονα
3. **Παράλληλη συγχώνευση ανά δύο σε δύο τμήματα διπλάσιου μεγέθους:** Η 3η φάση είναι η παράλληλη συγχώνευση ανά δύο σε δύο τμήματα με τη βοήθεια της μεθόδου **merge**.
4. **Συγχώνευση των δύο εναπομείναντων τμημάτων μεταξύ τους:** Η 4η φάση είναι η συγχώνευση των δύο εναπομείναντων τμημάτων σ’ ένα ενιαίο ταξινομημένο πίνακα με τη βοήθεια και πάλι της μεθόδου **merge**.
5. **Έλεγχος κριτηρίου τερματισμού:** Εφόσον, από την αναδρομική κλήση το μέγεθος του πίνακα δεν υπερβεί το όριο που θέσαμε στην παράμετρο LIMIT, τότε η αναδρομή τερματίζει και εκτελείται ο σειριακός αλγόριθμος ταξινόμησης με την μέθοδο **quicksort**.

## 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 main(int argc, char \*argv[]) | |
| **int** | |
| \*A | Δυναμικός 1Δ πίνακας που θα ταξινομηθεί |
| \*Space | Δυναμικός 1Δ που θα χρησιμοποιηθεί ως προσωρινός πίνακας αποθήκευσης για την ταξινόμηση του Α |
| threads | Το πλήθος των νημάτων που θα δημιουργηθούν |
| size | Το μέγεθος του πίνακα |
| i | Δείκτης επανάληψης |
| **double** | |
| start\_time | Έναρξη χρόνου μέτρησης της παράλληλης επεξεργασίας του αλγορίθμου |
| end\_time | Λήξη χρόνου μέτρησης της παράλληλης επεξεργασίας του αλγορίθμου |
| **FILE** | |
| \*fpA\_unsort | Αρχείο εξόδου για την αποθήκευση του πίνακα Α προτού ταξινομηθεί |
| \*fpA\_sort | Αρχείο εξόδου για την αποθήκευση του πίνακα Α αφού ταξινομηθεί |
| void multisort(int \*start, int \*space, int size) | |
| **int** | |
| \*start | Δείκτης στην αρχή του πίνακα που πρέπει να ταξινομηθεί |
| \*space | Δείκτης σ’ έναν βοηθητικό πίνακα ίδιου μεγέθους με τον αρχικό, που χρησιμοποιείται για την συγχώνευση των τμημάτων |
| size | Μέγεθος του πίνακα που θα ταξινομηθεί |
| quarter | Το μέγεθος του τμήματος που θα διαχωριστεί |
| \*startA | Δείκτης στην αρχή του 1ου υπο-τμήματος |
| \*startB | Δείκτης στην αρχή του 2ου υπο-τμήματος |
| \*startC | Δείκτης στην αρχή του 3ου υπο-τμήματος |
| \*startD | Δείκτης στην αρχή του 4ου υπο-τμήματος |
| \*spaceA | Δείκτης στην αρχή του βοηθητικού πίνακα του 1ου υπο-τμήματος |
| \*spaceB | Δείκτης στην αρχή του βοηθητικού πίνακα του 2ου υπο-τμήματος |
| \*spaceC | Δείκτης στην αρχή του βοηθητικού πίνακα του 3ου υπο-τμήματος |
| \*spaceD | Δείκτης στην αρχή του βοηθητικού πίνακα του 4ου υπο-τμήματος |
| void quicksort(int \*start, int \*end) | |
| **int** | |
| \*start | Δείκτης στην αρχή του τμήματος του πίνακα που πρέπει να ταξινομηθεί |
| \*end | Δείκτης στο τέλος του τμήματος του πίνακα που πρέπει να ταξινομηθεί |
| \*pvt | Το στοιχείο οδηγός που θα τοποθετηθεί στην σωστή θέση ταξινόμησης στον πίνακα |
| int\* pivotPartition(int \*start, int \*end) | |
| **int** | |
| \*start | Δείκτης στην αρχή του τμήματος του πίνακα που πρέπει να διαχωριστεί |
| \*end | Δείκτης στο τέλος του τμήματος του πίνακα που πρέπει να διαχωριστεί |
| \*pvt | Το στοιχείο οδηγός που θα τοποθετηθεί στην σωστή θέση ταξινόμησης στον πίνακα |
| \*i | Δείκτης που δείχνει σε θέση πριν από αυτήν που δείχνει ο δείκτης j |
| \*j | Δείκτης που δείχνει σε θέση μετά από αυτήν που δείχνει ο δείκτης i |
| void swap(int \*a, int \*b) | |
| **int** | |
| \*a | Δείκτης στο πρώτο στοιχείο που θα ανταλλαγεί |
| \*b | Δείκτης στο δεύτερο στοιχείο που θα ανταλλαγεί |
| temp | Προσωρινός χώρος αποθήκευσης |
| void merge(int \*startA, int \*endA, int \*startB, int \*endB, int \*space) | |
| **int** | |
| \*startA | Δείκτης στο πρώτο στοιχείο του πρώτου υποπίνακα (A) |
| \*endA | Δείκτης στο τελευταίο στοιχείο του πρώτου υποπίνακα (A) |
| \*startB | Δείκτης στο πρώτο στοιχείο του δεύτερου υποπίνακα (B) |
| \*endB | Δείκτης στο τελευταίο στοιχείο του δεύτερου υποπίνακα (B) |
| \*space | Δείκτης σε προσωρινό πίνακα αποθήκευσης |
| \*i | Δείκτης αρχικοποίησης για τον υπο-πίνακα Α |
| \*j | Δείκτης αρχικοποίησης για τον υπο-πίνακα Β |
| \*k | Δείκτης αρχικοποίησης για τον προσωρινό πίνακα αποθήκευσης |

### 2.3.2 Πρόγραμμα γεννήτρια για παραγωγή 1Δ πινάκων

Ο αλγόριθμος αυτός δημιουργεί τον τετραγωνικό πίνακα Α διαστάσεων N με ψευδοτυχαίους ακέραιους αριθμούς με βάση ένα διάστημα τιμών.

* **Κώδικας:**

for (i = 0; i < size; i++)

{

        A[i] = rand() % 199 - 99;

        A[i] = A[i] >= 0 ? A[i] + 10 : A[i] - 10;

}

* **Λειτουργία:**
  + Με βρόχο κάνουμε προσπέλαση τον 1Δ πίνακα
  + Επιλέγουμε τιμές από -99 εώς 99
  + Αλλάζουμε την τιμή κατά 10 ανάλογα το πρόσημο

### 2.3.3 Multisort

Η συνάρτηση **multisort** υλοποιεί τον εναλλακτικό παράλληλο αλγόριθμο ταξινόμησης σε σχέση με τον **mergesort**. Ο αλγόριθμος χωρίζει τον πίνακα σε τέσσερα ισομεγέθη τμήματα (quarters), ταξινομεί κάθε τμήμα αναδρομικά και στη συνέχεια συγχωνεύει τα ταξινομημένα τμήματα. Όταν το μέγεθος του τμήματος είναι μικρότερο από το όριο LIMIT, χρησιμοποιείται ο ακολουθιακός αλγόριθμος **quicksort** για την τοπική ταξινόμηση.

### 2.3.4 Quicksort

Η συνάρτηση **quicksort** υλοποιεί τον αναδρομικό αλγόριθμο γρήγορης ταξινόμησης

1. Επιλέγει ένα στοιχείο οδηγός (pivot) από το τμήμα του πίνακα που ταξινομείται, δηλαδή, ένα στοιχείο που βρίσκεται στην σωστή θέση ταξινόμησης.
2. Τοποθετεί όλα τα στοιχεία μικρότερα ή ίσα με τον οδηγό στα αριστερά του και όλα τα μεγαλύτερα στα δεξιά του, διαχωρίζοντας τον πίνακα σε δύο υποπίνακες.
3. Επαναλαμβάνει αναδρομικά τη διαδικασία για τον αριστερό και δεξιό υποπίνακα μέχρι να επιτευχθεί η πλήρης ταξινόμηση.

### 2.3.5 Pivot Partition

Η συνάρτηση **pivotPartition** με την τεχνική δύο δεικτών εκτελεί τον διαχωρισμό του πίνακα γύρω από ένα στοιχείο οδηγός (pivot).

1. Το pivot ορίζεται αρχικά ως το τελευταίο στοιχείο του τμήματος του πίνακα.
2. Μετακινεί όλα τα στοιχεία μικρότερα ή ίσα με το pivot στην αριστερή πλευρά του πίνακα, ενώ τα μεγαλύτερα παραμένουν στη δεξιά πλευρά.
3. Τέλος, το pivot τοποθετείται στη σωστή του θέση, δηλαδή μεταξύ των μικρότερων και των μεγαλύτερων στοιχείων.

### 2.3.6 Swap

Η συνάρτηση **swap** πραγματοποιεί την ανταλλαγή (swap) των τιμών δύο μεταβλητών

που δείχνονται από τους δείκτες a και b.

### 2.3.7 Merge

Η συνάρτηση **merge** συγχωνεύει δύο ταξινομημένα υποτμήματα (A και B) σε έναν ενιαίο ταξινομημένο πίνακα. Ο συγχωνευμένος πίνακας αποθηκεύεται αρχικά στον προσωρινό πίνακα space και στη συνέχεια επιστρέφεται στον αρχικό πίνακα.

# 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;

            }

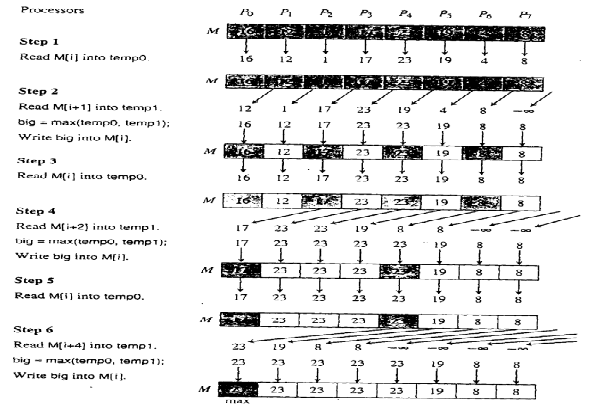
            #pragma omp barrier

          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, 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 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.*

*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 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     : 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](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](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 0.000007 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](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](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 0.000081 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](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](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_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](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](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_T4_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](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](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 0.001395 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](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](Output/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 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, 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:

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, 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 Ν, 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 **Ν = 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 Ν, όπως παρατηρούμε στα δεδομένα που συλλέξαμε για **Ν = 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 Β. Μηνύματα σφάλματος για Segmentation Fault και παράλογων τιμών στον πίνακα Μ 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 Μ 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



Σας ευχαριστώ για την προσοχή σας.

