

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

# ΕΡΓΑΣΙΑ 2Β.1

# CUDA

**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 χρήση της αρχιτεκτονικής CUDA για την επίλυση ενός προβλήματος υπολογισμών σε πίνακες, με αξιοποίηση παράλληλης επεξεργασίας.

## 1.2 Συνοπτική περιγραφή του προβλήματος που επιλύεται

Το πρόγραμμα δημιουργεί έναν τυχαίο 2Δ πίνακα ακέραιων αριθμών και εκτελεί τις παρακάτω πράξεις:

* Υπολογισμό μέσου όρου (calcAvg).
* Εύρεση μέγιστου στοιχείου (findMax).
* Δημιουργία πίνακα B βάσει των στοιχείων του αρχικού πίνακα (createB).
* Δημιουργία πίνακα C βάσει γειτονικών στοιχείων (createC).

# 2. Σχεδιασμός

## 2.1 Περιγραφή της προσέγγισης που ακολουθήθηκε

Η προσέγγιση περιλαμβάνει την παραλληλοποίηση βασικών λειτουργιών σε πίνακες χρησιμοποιώντας CUDA kernels. Οι υπολογισμοί κατανέμονται σε νήματα μέσω πλέγματος (grid) και μπλοκ (blocks).

## 2.2 Ανάλυση της λογικής και των μεθοδολογιών

Η λογική βασίζεται στα εξής:

 Χρήση παράλληλων reduction για υπολογισμό μέσου όρου και εύρεση μέγιστου.

 Χρήση shared memory για βελτίωση απόδοσης.

 Διαίρεση του πίνακα σε τμήματα για την κατανομή της δουλειάς σε threads.

## 2.3 Περιγραφή των δομών δεδομένων και των αλγορίθμων

### 2.3.1 Δομές δεδομένων και μεταβλητές

Οι βασικές δομές δεδομένων περιλαμβάνουν:

* Πίνακες εισόδου (d\_A) και εξόδου (d\_OutArr) στη μνήμη της συσκευής (device).
* Μεταβλητές για αποθήκευση μέσου όρου, μέγιστου και ελάχιστου στοιχείου.
* Δομές για τη δημιουργία CUDA events για μέτρηση χρόνου.

### 2.3.2 Πρόγραμμα γεννήτρια για παραγωγή 2Δ πινάκων

Η συνάρτηση create2DArray δημιουργεί τυχαίο πίνακα μεγέθους NxN, διασφαλίζοντας ότι το μέγιστο στοιχείο του είναι μεγαλύτερο από N⋅mN \cdot mN⋅m, όπου mmm ο μέσος όρος.

### 2.3.3 calcAvg<<<>>>()

Η συνάρτηση **πυρήνα** υπολογίζει τον μέσο όρο όλων των στοιχείων του πίνακα χρησιμοποιώντας παράλληλο reduction και atomic εντολές.

### 2.3.4 findMax<<<>>>()

Η συνάρτηση **πυρήνα** εντοπίζει το μέγιστο στοιχείο του πίνακα με παράλληλο reduction και χρήση atomic εντολών.

### 2.3.5 createB<<<>>>()

Υπολογίζει τον πίνακα Bij=(m−Aij) / amax και βρίσκει το ελάχιστο στοιχείο του πίνακα.

### 2.3.6 createC<<<>>>()

Δημιουργεί τον πίνακα Cij=(Aij+Ai(j+1)+Ai(j−1)) / 3​, λαμβάνοντας υπόψη τα όρια του πίνακα.

### 2.3.7 atomicMin

Η atomicMin είναι μια προσαρμοσμένη συνάρτηση που χρησιμοποιείται για τον υπολογισμό της ελάχιστης τιμής σε αριθμούς κινητής υποδιαστολής στη συσκευή (device). Η χρήση της είναι αναγκαία, καθώς η CUDA δεν παρέχει εγγενή υποστήριξη για atomic εντολές σε αριθμούς κινητής υποδιαστολής (floats).

# 3. Υλοποίηση

## 3.1 Αναφορά στις βασικές λειτουργίες του κώδικα

Ο κώδικας περιλαμβάνει:

* Αρχικοποίηση δεδομένων και παραμέτρων CUDA.
* Δημιουργία και μεταφορά δεδομένων από host σε device.
* Εκτέλεση των παραπάνω CUDA kernels.
* Ανάκτηση και αποθήκευση αποτελεσμάτων.

## 3.2 Επεξήγηση παράλληλων τμημάτων του κώδικα

 Η χρήση shared memory βελτιώνει την ταχύτητα των υπολογισμών, μειώνοντας την προσπέλαση στην παγκόσμια μνήμη.

 Οι παράλληλες λειτουργίες διανέμουν το έργο σε threads και blocks.

## 3.3 Περιγραφή της επικοινωνίας και του συγχρονισμού μεταξύ νημάτων

 Χρήση \_\_syncthreads() για συγχρονισμό threads στο ίδιο μπλοκ.

 Χρήση atomic λειτουργιών για ασφαλή τροποποίηση κοινών δεδομένων.

# 4. Δοκιμές και Αποτελέσματα

## 4.1 Αναφορά των συνθηκών εκτέλεσης

Η εκτέλεση γίνεται για διαφορετικά μεγέθη πίνακα Ν x N, αριθμούς νημάτων T ανά μπλοκ και μπλοκ ανά πλέγμα.

Η μεταγλώττιση του προγράμματος γίνεται μέσω command line σε περιβάλλον Linux, με τον compiler της NVIDIA **nvcc**.

nvcc -o cuda1 cuda1.cu

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 ο πίνακας Α και ο πίνακας Β ή C. Ενδεικτική εντολή εκτέλεσης:

./cuda1 A.txt OutArr.txt

## 4.2 Παρουσίαση των αποτελεσμάτων σε μορφή κειμένου

*Τα αποτελέσματα είναι αποθηκευμένα στον φάκελο <Output> και οι πίνακες [A](A_unsort) και [B ή C](A_sort) 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 ο πίνακας Α και ο B ή ο C. Σε περίπτωση που ο χρήστης δεν βάλει τον απαιτούμενο αριθμό παραμέτρων, το πρόγραμμα τερματίζεται και εμφανίζεται χαρακτηριστικό μήνυμα

### 4.2.1 Output\_no\_args.txt

Usage: ./cuda1 A.txt OutArr.txt

Για να ελέγξουμε την ορθότητα του αλγορίθμου δοκιμάσαμε για αρχή έναν πίνακα μικρού μεγέθους, ώστε να επιβεβαιώσουμε ότι επιτυγχάνεται η ταξινόμηση.

### 4.2.2 [Output.txt](Output/Output.txt)

--------------- Device Properties ---------------

Device name : NVIDIA TITAN RTX

Max threads per block : 1024

Max block dimensions : 1024 x 1024 x 64

Max grid dimensions : 2147483647 x 65535 x 65535

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

--------------- Input Parameters ---------------

Matrix size : 8 x 8

Blocks per Grid : 2

Threads per Block : 4

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

The array A has been stored in file A/A6.txt

Average: 64.41

Time for the kernel calcAvg<<<>>>(): 0.204736 ms

Max: 624

Time for the kernel findMax<<<>>>(): 0.015552 ms

The array B has been stored in file OutArr/OutArr6.txt

Min: -0.0522

Time for the kernel createB<<<>>>(): 0.015040 ms

### 4.2.3 [Output\_T1\_N1000\_L100.txt](Output/Output_T1_N1000_L100.txt)

--------------- Device Properties ---------------

Device name : NVIDIA TITAN RTX

Max threads per block : 1024

Max block dimensions : 1024 x 1024 x 64

Max grid dimensions : 2147483647 x 65535 x 65535

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

--------------- Input Parameters ---------------

Matrix size : 8 x 8

Blocks per Grid : 2

Threads per Block : 4

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

The array A has been stored in file A/A7.txt

Average: 66.98

Time for the kernel calcAvg<<<>>>(): 0.207712 ms

Max: 444

Time for the kernel findMax<<<>>>(): 0.014592 ms

The array C has been stored in file OutArr/OutArr7.txt

Time for the kernel createC<<<>>>(): 0.012992 ms

### 4.2.4 [Output\_T1\_N500000\_L500.txt](Output/Output_T1_N500000_L500.txt)

Threads             : 1

Matrix size         : 500000

Limit for quicksort : 500

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

Before sorting

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

The A has been stored in A\_unsort/A\_unsort\_T1\_N500000\_L500.txt

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

After sorting

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

The A has been stored in A\_sort/A\_sort\_T1\_N500000\_L500.txt

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

Multisort finished in 0.079525 sec.

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

### 4.2.5 [Output\_T1\_N100000000\_L1000.txt](Output/Output_T1_N100000000_L1000.txt)

Threads             : 1

Matrix size         : 100000000

Limit for quicksort : 1000

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

Before sorting

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

The A has been stored in A\_unsort/A\_unsort\_T1\_N100000000\_L1000.txt

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

After sorting

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

The A has been stored in A\_sort/A\_sort\_T1\_N100000000\_L1000.txt

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

Multisort finished in 19.709039 sec.

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

### 4.2.6 [Output\_T2\_N1000\_L100.txt](Output/Output_T2_N1000_L100.txt)

Threads             : 2

Matrix size         : 1000

Limit for quicksort : 100

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

Before sorting

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

The A has been stored in A\_unsort/[A\_unsort\_T2\_N1000\_L100.txt](A_unsort/A_unsort_T2_N1000_L100.txt)

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

After sorting

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

The A has been stored in A\_sort/[A\_sort\_T2\_N1000\_L100.txt](A_sort/A_sort_T2_N1000_L100.txt)

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

Multisort finished in 0.000197 sec.

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

### 4.2.7 [Output\_T3\_N5000\_L100.txt](Output/Output_T3_N5000_L100.txt)

Threads             : 3

Matrix size         : 5000

Limit for quicksort : 100

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

Before sorting

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

The A has been stored in A\_unsort/[A\_unsort\_T3\_N5000\_L100.txt](A_unsort/A_unsort_T3_N5000_L100.txt)

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

After sorting

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

The A has been stored in A\_sort/[A\_sort\_T3\_N5000\_L100.txt](A_sort/A_sort_T2_N5000_L100.txt)

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

Multisort finished in 0.000470 sec.

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

### 4.2.8 [Output\_T4\_N1000000\_L500.txt](Output/Output_T4_N1000000_L500.txt)

Threads             : 4

Matrix size         : 1000000

Limit for quicksort : 500

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

Before sorting

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

The A has been stored in A\_unsort/A\_unsort\_T4\_N1000000\_L500.txt

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

After sorting

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

The A has been stored in A\_sort/A\_sort\_T4\_N1000000\_L500.txt

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

Multisort finished in 0.057440 sec.

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

### 4.2.9 [Output\_T6\_N500000\_L500.txt](Output/Output_T6_N500000_L500.txt)

Threads             : 6

Matrix size         : 500000

Limit for quicksort : 500

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

Before sorting

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

The A has been stored in A\_unsort/A\_unsort\_T6\_N500000\_L500.txt

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

After sorting

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

The A has been stored in A\_sort/A\_sort\_T6\_N500000\_L500.txt

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

Multisort finished in 0.021820 sec.

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

### 4.2.10 [Output\_T8\_N100000000\_L1000.txt](Output/Output_T8_N100000000_L1000.txt)

Threads             : 8

Matrix size         : 100000000

Limit for quicksort : 1000

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

Before sorting

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

The A has been stored in A\_unsort/A\_unsort\_T8\_N100000000\_L1000.txt

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

After sorting

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

The A has been stored in A\_sort/A\_sort\_T8\_N100000000\_L1000.txt

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

Multisort finished in 6.076829 sec.

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

### 4.2.11 [Output\_T12\_N50000000\_L1000.txt](Output/Output_T12_N50000000_L1000.txt)

Threads             : 12

Matrix size         : 50000000

Limit for quicksort : 1000

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

Before sorting

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

The A has been stored in A\_unsort/A\_unsort\_T12\_N50000000\_L1000.txt

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

After sorting

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

The A has been stored in A\_sort/A\_sort\_T12\_N50000000\_L1000.txt

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

Multisort finished in 2.945010 sec.

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

### 4.2.12 [Output\_T16\_N100000000\_L1000.txt](Output/Output_T16_N100000000_L1000.txt)

Threads             : 16

Matrix size         : 100000000

Limit for quicksort : 1000

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

Before sorting

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

The A has been stored in A\_unsort/A\_unsort\_T16\_N100000000\_L1000.txt

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

After sorting

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

The A has been stored in A\_sort/A\_sort\_T16\_N100000000\_L1000.txt

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

Multisort finished in 6.174742 sec.

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

## 4.3 Ανάλυση της αποδοτικότητας

### 4.3.1 Χρόνοι εκτέλεσης του παράλληλου αλγορίθμου

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, 3, 4, 6, 8, 12 και 16
* **Μέγεθος Πίνακα:** 1000, 5000, 10000, 50000, 100000, 500000, 1000000, 5000000, 10000000, 50000000, 100000000

|  |  |
| --- | --- |
| **Threads** | **Total Time** |
| 1 | 0,00012 |
| 2 | 0,000197 |
| 3 | 0,000504 |
| 4 | 0,001087 |

**Εικόνα 1.** Χρόνοι εκτέλεσης του παράλληλου αλγορίθμου για Ν = 1000

|  |  |
| --- | --- |
| **Threads** | **Total Time** |
| 1 | 0,000654 |
| 2 | 0,000459 |
| 3 | 0,00047 |
| 4 | 0,00194 |

**Εικόνα 2.** Χρόνοι εκτέλεσης του παράλληλου αλγορίθμου για Ν = 5000

|  |  |
| --- | --- |
| **Threads** | **Total Time** |
| 1 | 0,001412 |
| 2 | 0,000955 |
| 3 | 0,00138 |
| 4 | 0,003595 |

**Εικόνα 3.** Χρόνοι εκτέλεσης του παράλληλου αλγορίθμου για Ν = 10000

|  |  |
| --- | --- |
| **Threads** | **Total Time** |
| 1 | 0,015074 |
| 4 | 0,005436 |
| 6 | 0,005173 |
| 8 | 0,005402 |

**Εικόνα 4.** Χρόνοι εκτέλεσης του παράλληλου αλγορίθμου για Ν = 100000

|  |  |
| --- | --- |
| **Threads** | **Total Time** |
| 1 | 0,079525 |
| 4 | 0,030407 |
| 6 | 0,02182 |
| 8 | 0,025397 |

**Εικόνα 5.** Χρόνοι εκτέλεσης του παράλληλου αλγορίθμου για Ν = 500000

|  |  |
| --- | --- |
| **Threads** | **Total Time** |
| 1 | 0,151469 |
| 4 | 0,05744 |
| 6 | 0,057255 |
| 8 | 0,060294 |

**Εικόνα 6.** Χρόνοι εκτέλεσης του παράλληλου αλγορίθμου για Ν = 1000000

|  |  |
| --- | --- |
| **Threads** | **Total Time** |
| 1 | 1,795719 |
| 8 | 0,515042 |
| 12 | 0,525958 |
| 16 | 0,607034 |

**Εικόνα 7.** Χρόνοι εκτέλεσης του παράλληλου αλγορίθμου για Ν = 10000000

|  |  |
| --- | --- |
| **Threads** | **Total Time** |
| 1 | 9,168969 |
| 8 | 3,122941 |
| 12 | 2,94501 |
| 16 | 3,050295 |

**Εικόνα 8.** Χρόνοι εκτέλεσης του παράλληλου αλγορίθμου για Ν = 50000000

|  |  |
| --- | --- |
| **Threads** | **Total Time** |
| 1 | 19,70904 |
| 8 | 6,076829 |
| 12 | 6,154808 |
| 16 | 6,174742 |

**Εικόνα 9.** Χρόνοι εκτέλεσης του παράλληλου αλγορίθμου για Ν = 100000000

### 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, 3, 4, 6, 8, 12 και 16
* **Μέγεθος Πίνακα:** 1000, 5000, 10000, 50000, 100000, 500000, 1000000, 5000000, 10000000, 50000000, 100000000

|  |  |
| --- | --- |
| **Threads** | **Speed-up** |
| 1 | 1 |
| 2 | 0,609 |
| 3 | 0,238 |
| 4 | 0,11 |

**Εικόνα 10.** Επιταχύνσεις του παράλληλου αλγορίθμου για Ν = 1000

|  |  |
| --- | --- |
| **Threads** | **Speed-up** |
| 1 | 1 |
| 2 | 1,426 |
| 3 | 1,392 |
| 4 | 0,338 |

**Εικόνα 11.** Επιταχύνσεις του παράλληλου αλγορίθμου για Ν = 5000

|  |  |
| --- | --- |
| **Threads** | **Speed-up** |
| 1 | 1 |
| 2 | 1,479 |
| 3 | 1,023 |
| 4 | 0,393 |

**Εικόνα 12.** Επιταχύνσεις του παράλληλου αλγορίθμου για Ν = 10000

|  |  |
| --- | --- |
| **Threads** | **Speed-up** |
| 1 | 1 |
| 4 | 2,77 |
| 6 | 2,91 |
| 8 | 2,79 |

**Εικόνα 13.** Επιταχύνσεις του παράλληλου αλγορίθμου για Ν = 100000

|  |  |
| --- | --- |
| **Threads** | **Speed-up** |
| 1 | 1 |
| 4 | 2,61 |
| 6 | 3,65 |
| 8 | 3,14 |

**Εικόνα 14.** Επιταχύνσεις του παράλληλου αλγορίθμου για Ν = 500000

|  |  |
| --- | --- |
| **Threads** | **Speed-up** |
| 1 | 1 |
| 4 | 2,63 |
| 6 | 2,64 |
| 8 | 2,51 |

**Εικόνα 15.** Επιταχύνσεις του παράλληλου αλγορίθμου για Ν = 1000000

|  |  |
| --- | --- |
| **Threads** | **Speed-up** |
| 1 | 1 |
| 8 | 3,48 |
| 12 | 3,41 |
| 16 | 2,96 |

**Εικόνα 16.** Επιταχύνσεις του παράλληλου αλγορίθμου για Ν = 10000000

|  |  |
| --- | --- |
| **Threads** | **Speed-up** |
| 1 | 1 |
| 8 | 2,93 |
| 12 | 3,11 |
| 16 | 3,01 |

**Εικόνα 17.** Επιταχύνσεις του παράλληλου αλγορίθμου για Ν = 50000000

|  |  |
| --- | --- |
| **Threads** | **Speed-up** |
| 1 | 1 |
| 8 | 3,25 |
| 12 | 3,21 |
| 16 | 3,19 |

**Εικόνα 18.** Επιταχύνσεις του παράλληλου αλγορίθμου για Ν = 100000000

### 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 έχουμε να κάνουμε με διακριτές εργασίες που αναλαμβάνει 1 νήμα και μηχανισμούς συγχρονισμού με οδηγίες taskwait.

Για παράδειγμα, στα δεδομένα που συλλέξαμε για **Ν = 1000**, 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, 3 ή και 4 νήματα.

Ωστόσο, τα οφέλη του παράλληλου υπολογισμού διακρίνονται για μεγάλα Ν, όπως παρατηρούμε στα δεδομένα που συλλέξαμε για **Ν = 100000000**, καθώς, εκεί πετυχαίνουμε καλύτερους χρόνους με περισσότερα νήματα και συνεπώς και καλύτερες επιταχύνσεις. Αυτό οφείλεται και στην ανάθεση των εργασιών στα διαθέσιμα νήματα, όπου ελαχιστοποιείται ο χρόνος αδράνειας λόγω του μεγάλου όγκου δεδομένων.

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 εντοπίστηκε στην διαχείριση των δεικτών στην ρουτίνα multisort και merge. Συγκεκριμένα, δεν κατανοήθηκε πλήρως ο ρόλος του δείκτη space και πώς θα αποθηκεύαμε τον τελικό ταξινομημένο πίνακα. Πρόβλημα εντοπίσαμε και στο 4ο τμήμα του πίνακα, όπου αν δεν διαιρούτανε ο πίνακας διά 4 ακριβώς, τότε περίσσευαν στοιχεία του πίνακα που δεν ταξινομήθηκαν.

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 ολοκληρωθεί ο παράλληλος αλγόριθμος από το αναμενόμενο.

## 5.2 Λύσεις που δοκιμάστηκαν και εφαρμόστηκαν

Εν τέλη, ο δείκτης space έδειχνε σ’ έναν προσωρινό πίνακα αποθήκευσης όπου θα τον χρησιμοποιούσαμε για αντιγράψουμε τον ταξινομημένο τελικό πίνακα στον δείκτη που έδειχνε στον αρχικό. Όσο για το πρόβλημα του διαμοιρασμού στο 4ο τμήμα του πίνακα αλλάξαμε το μέγεθος που θα εξασφαλίσει ότι το 4ο τμήμα θα έχει και τα περίσσεια στοιχεία σε περίπτωση που το μέγεθος δεν διαιρείται με το 4:

multisort(startD, spaceD, size - 3 \* quarter);

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

**WSL**

Threads             : 6

Matrix size         : 10000000

Limit for quicksort : 100

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

Before sorting

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

The A has been stored in A\_unsort.txt

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

After sorting

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

The A has been stored in A\_sort.txt

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

Multisort finished in 0.670413 sec.

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

**Linux**

Threads             : 6

Matrix size         : 10000000

Limit for quicksort : 100

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

Before sorting

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

The A has been stored in A\_unsort.txt

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

After sorting

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

The A has been stored in A\_sort.txt

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

Multisort finished in 0.562977 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, ο παράλληλος αλγόριθμος multisort με OpenMP tasks είναι μια αποδοτική προσέγγιση για την επίτευξη γρήγορης ταξινόμησης μεγάλων συνόλων δεδομένων σε παράλληλο περιβάλλον, εκμεταλλευόμενος τις δυνατότητες των σύγχρονων επεξεργαστών.

Η τεχνική του διαίρει και βασίλευε σε συνδυασμό με τον παράλληλο υπολογισμό επιτυγχάνουν εκπληκτικές επιδόσεις για μεγάλο όγκο δεδομένων και αυτό φαίνεται και στις επιταχύνσεις που καταγράψαμε για μεγάλα μεγέθη σε σύγκριση με μικρά.



Σας ευχαριστώ για την προσοχή σας.

