Πανεπιστήμιο Δυτικής Αττικής
Σχολή Μηχανικών
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Πρώτη Εργασία Παράλληλων Συστημάτων

Σωτήριος Αίας Καριώρης
Α.Μ.: 19390079
Πρόγραμμα ΠΑΔΑ
Ημερομηνία Παράδοσης: 05-12-2022

Περιεχόμενα

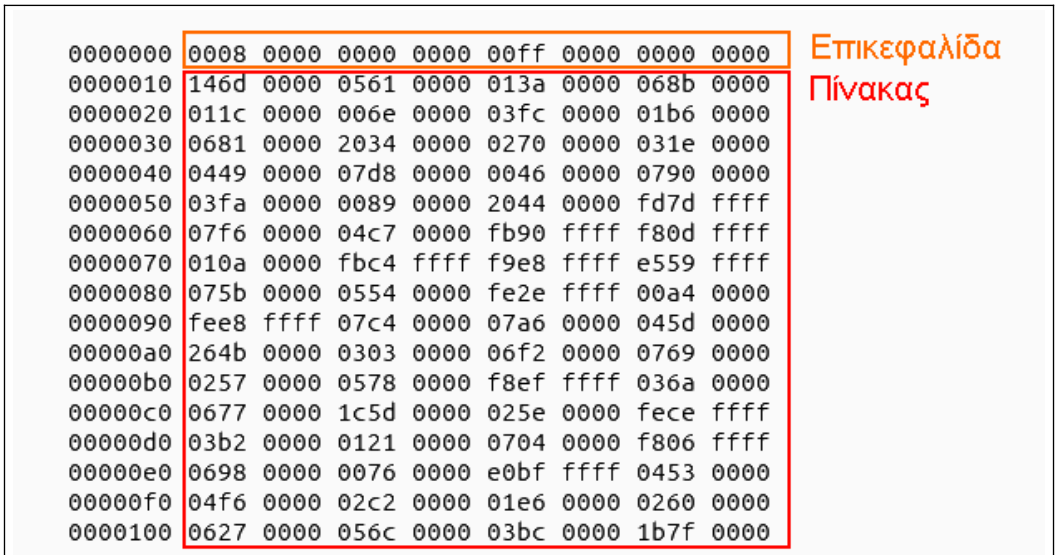
Εισαγωγή

Θέμα αυτής της εργασίας είναι η επεξεργασία πινάκων με χρήση της βιβλιοθήκης *OpenMP*. Όλα τα ερωτήματα έχουν απαντηθεί πλήρως. Στις παρακάτω ενότητες θα αναλυθεί η δομή του προγράμματος, οι λεπτομέρειες του κώδικα και της λειτουργίας του όπως και κάποια στατιστικά για τις επιδόσεις του. Επισυνημμένος με αυτό το έγγραφο είναι ο πηγαίος κώδικας.

Οι δοκιμές έγιναν σε περιβάλλον *Linux*, χρησιμοποιώντας την διανομή *Ubuntu 22.04* σε μηχάνημα 64-Bits με 4 πυρήνες. Σημειώνεται ότι η εικονική μηχανή χρησιμοποίησε και τους 4 πυρήνες όπως και 6GB RAM. Το πρόγραμμα επίσης εξετάστηκε και στην εικονική μηχανή του πανεπιστημίου.

Αρχεία Test

Τα δεδομένα εισόδου του προγράμματος είναι αποθηκευμένα σε ειδικά δυαδικά αρχεία που σχεδιάστηκαν για αυτήν την άσκηση. Η δομή των αρχείων αυτών είναι αρκετά απλή: Η επικεφαλίδα περιλαμβάνει τον αριθμό `N`, ένα `flag` byte που δείχνει αν ο πίνακας που περιέχεται στο αρχείο είναι αυστηρά διαγώνια δεσπόζον (ΑΑΔ) και 7 bytes με αδιάφορες τιμές ώστε η επικεφαλίδα να φτάνει τα 16 Bytes. Έπειτα ακολουθούν N*N ακέραιοι. Σημειώνεται πως το flag byte δεν χρησιμοποιείται από το πρόγραμμα της άσκησης. Τα ονόματα των αρχείων `test` είναι τυχαίοι τριψήφιοι αριθμοί.



Εικόνα 1: Δομή test αρχείου 8x8

Τα αρχεία αυτά δημιουργήθηκαν με το πρόγραμμα `gen` που σχεδιάστηκε αποκλειστικά για αυτόν τον σκοπό. Αν και ο τρόπος λειτουργίας του είναι εκτός του ενδιαφέροντος της άσκησης ο πηγαίος του κώδικας έχει συμπεριληφθεί με την υπόλοιπη εργασία. Τέλος, σημειώνεται πως το πρόγραμμα `gen` όπως και τα προγράμματα της άσκησης αποθηκεύουν και διαβάζουν τα αρχεία `test` από τον κατάλογο απλώς ονομασμένο “`test`”

Δομή Προγράμματος

Το πρόγραμμα της άσκησης είναι χωρισμένο σε έξι μέρη. Στο πρώτο μέρος το πρόγραμμα φορτώνει στην μνήμη των πίνακα Α από το αρχείο που του επισημάνθηκε ως όρισμα. Στα επόμενα τέσσερα μέρη υλοποιούνται τα ζητούμενα της εκφώνησης (ένα ζητούμενο σε κάθε μέρος). Καθώς το μέρος 4 ζητείται να υλοποιηθεί με τρεις διαφορετικούς τρόπους υπάρχουν τρία διαφορετικά προγράμματα με διαφορετικό μέρος D το καθένα. Με εξαίρεση αυτά τα κομμάτια κώδικα, οι τρεις υλοποιήσεις είναι ίδιες. Τέλος, στο έκτο μέρος αποδεσμεύονται απλώς τα δεδομένα που έχουν μείνει.

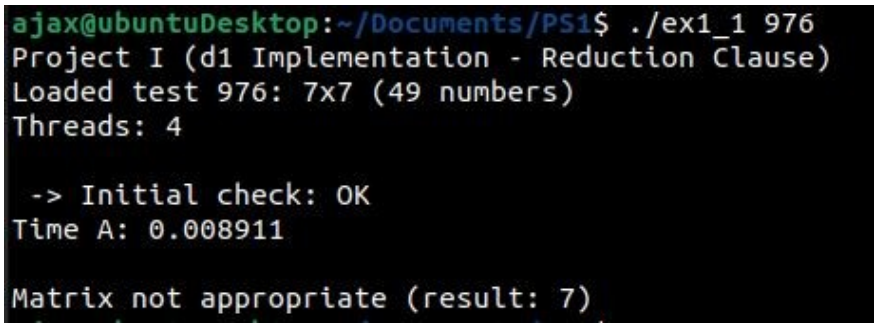
Κάθε μέρος χρονομετρείται και οι αντίστοιχοι χρόνοι εμφανίζονται. Επίσης εμφανίζεται και ο συνολικός χρόνος εκτέλεσης. Καθώς οι δοκιμές που έγιναν είχαν πίνακες με μεγέθη της τάξης του εκατομμυρίου, η αποδέσμευση μνήμης γίνεται πάντοτε το συντομότερο δυνατό ενώ οι επιμέρους χρονομετρήσεις δεν περιλαμβάνουν την διαχείριση μνήμης (`malloc` & `free`). Παρόλα αυτά, αναπόφευκτα ο τελικός χρόνος περιλαμβάνει και αυτές τις σχετικά χρονοβόρες και χρονικά τυχαίες διαδικασίες και για τον λόγο αυτόν τα αθροίσματα των επιμέρους χρόνων είναι μικρότερα από τον τελικό χρόνο. Γενικά, οι χρονομετρήσεις γίνονται για να συγκριθούν τα παράλληλα μέρη μεταξύ τους.

Ενδεικτικά Τρεξίματα (Ι)

Παρακάτω φαίνονται δύο ενδεικτικά τρεξίματα του κώδικα με την πρώτη υλοποίηση. Χρησιμοποιούνται δύο μικροί πίνακες 8x8, ένας εκ των οποίων δεν είναι ΑΔΔ.

```
ajax@ubuntuDesktop:~/Documents/PS1$ ./ex1_1 953
Project I (d1 Implementation - Reduction Clause)
Loaded test 953: 7x7 (49 numbers)
Threads: 4

 -> Initial check: OK
Time A: 0.000239

 -> Max: 4185
Time B: 0.000004

 -> Matrix B: OK (will be displayed/outputed in the end)
Time C: 0.000004

 -> Min in B: 3141
Time D: 0.000003

Total Time: 0.000282 sec

  4185   3161   3145   4169   3157   4169   3145
  4169   4185   4165   3161   4169   4165   4165
  4165   4181   4185   3157   3161   4165   3141
  3161   3161   4181   4185   4169   3157   4185
  3157   4165   3141   4181   4185   4185   3157
  3161   4185   4165   3161   3141   4185   3141
  3161   4185   4181   3141   3157   4165   4185
```

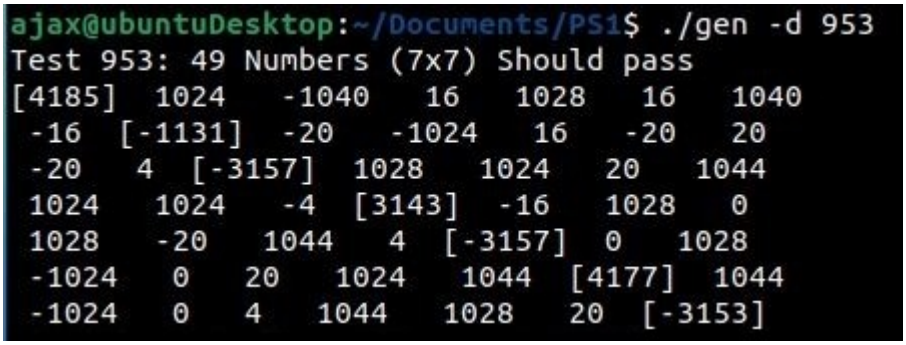
Εικόνα 2: Εκτέλεση με ΑΔΔ πίνακα

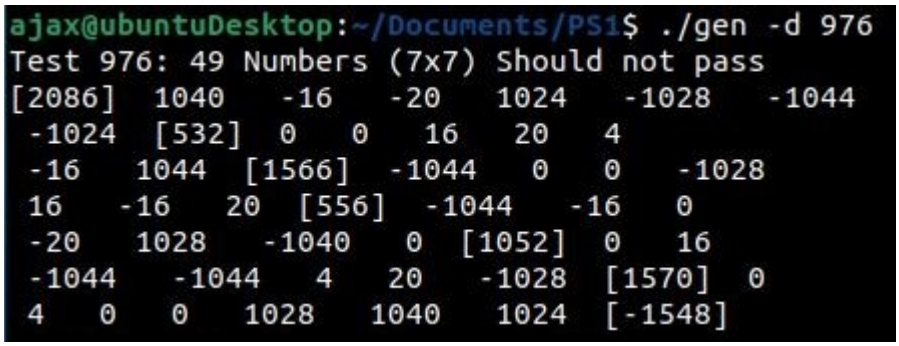


Εικόνα 3: Εκτέλεση με μη-ΑΔΔ

Η μεταβλητή `result` δείχνει για διαγνωστικούς λόγους πόσες γραμμές δεν τηρούν τον ορισμό ενός ΑΔΔ πίνακα. Στον συγκεκριμένο πίνακα καμία γραμμή δεν ήταν κατάλληλη.

Οι πίνακες που χρησιμοποιήθηκαν είναι οι εξής:





Μέρος Α

Στο μέρος Α εξετάζεται αν ο πίνακας είναι ΑΔΔ με παράλληλο τρόπο. Αν διαπιστωθεί πως ο πίνακας δεν είναι ΑΔΔ, το πρόγραμμα τερματίζεται. Η φιλοσοφία του αλγορίθμου είναι η εξής: Κάθε γραμμή αθροίζεται ολόκληρη κατά απόλυτη τιμή. Έπειτα, από το άθροισμα αυτό αφαιρείται το στοιχείο που ανήκει στην διαγώνιο του πίνακα και η διαφορά που μένει συγκρίνεται με το στοιχείο αυτό. Στην περίπτωση που διαπιστωθεί πως το άθροισμα δεν είναι μικρότερο της διαγωνίου αυξάνεται η μεταβλητή `result` που έχει αρχικοποιηθεί με τιμή 0. Στο τέλος του αλγορίθμου εξετάζεται αν η `result` έχει αυξηθεί και μόνο αν είναι ακόμα 0 η εκτέλεση του προγράμματος συνεχίζεται.

Χρησιμοποιώντας το for clause του OMP ο αλγόριθμος παραλληλοποιείται ανά γραμμή του πίνακα. Κάθε νήμα αναλαμβάνει έναν αριθμό γραμμών με βάση την διαταγή `parallel for`.

Αθροίζοντας ολόκληρη την γραμμή του πίνακα έναντι του ελέγχου για το διαγώνιο στοιχείο πριν την πρόσθεση κάθε αριθμού μειώνονται οι εντολές που χρειάζεται να πραγματοποιηθούν ανά στοιχείο του πίνακα. Επίσης, αν και σε έναν μη ΑΔΔ πίνακα ένα ή παραπάνω νήματα μπορεί να αποκλειστούν προσωρινά όσο περιμένουν να αυξήσουν την `result` (που είναι ορισμένη ως ατομική διαδικασία), σε έναν ΑΔΔ κανένα νήμα δεν θα χρειαστεί να περιμένει να εκτελέσει την ατομική διαδικασία καθώς αυτή παραλείπεται.

Μέρος Β

Στο μέρος αυτό υπολογίζεται το μέγιστο στοιχείο της διαγωνίου. Καθώς χρησιμοποιείται το reduction clause όλα τα στοιχεία προς σύγκριση πρέπει να βρίσκονται στον ίδιο πίνακα. Έτσι, τα νήματα παράλληλα γεμίζουν τον πίνακα `D` με τα στοιχεία της διαγωνίου, μοιράζοντας τις γραμμές του πίνακα μεταξύ τους όπως και στο μέρος Α. Έπειτα η διαταγή `for reduction` διαχειρίζεται την πραγματική σύγκριση, επιστρέφοντας το μέγιστο στοιχείο του `D` στην μεταβλητή `m`. Το μέρος αυτό είναι το συντομότερο από άποψη χρόνου, με καμία εκτέλεση να χρειάζεται πάνω από 1 ms για να το ολοκληρώσει.

Μέρος Γ

Στο μέρος Γ δημιουργείται ο πίνακας `B`. Ο πίνακας αυτός περιέχει την διαφορά των στοιχείων του `A` από το `m` του μέρους Β ενώ η διαγώνιος του αποτελείται από το `m`. Ο κώδικας σε αυτό το μέρος είναι αρκετά απλός: Πρώτα υπολογίζεται ο πίνακας `B` μαζί με την διαγώνιο και έπειτα τοποθετούνται τα πραγματικά στοιχεία της διαγωνίου.

Σημειώνεται πως προς αποφυγήν χρήσης συναρτήσεων I/O όσο το πρόγραμμα χρονομετρείται, ο πίνακας εξάγεται στο τέλος του προγράμματος. Αν είναι πολύ μεγάλος το πρόγραμμα θα τον αποθηκεύσει σε ένα αρχείο `txt` ενώ αν τα δεδομένα του ξεπερνούν το 1MB δεν θα τον αποθηκεύσει καθόλου.

Μέρος Δ.1

Η πρώτη υλοποίηση του μέρους Δ χρησιμοποιεί για μια ακόμη φορά το reduction clause του OMP για τον υπολογισμό του μικρότερου στοιχείου του πίνακα `B`. Παρόμοια με το μέρος Α, ο αλγόριθμος ελέγχει όλα τα στοιχεία του πίνακα και ο τρόπος διαμοίρασης των δεδομένων μένει κρυφή στον προγραμματιστή.

Μέρος Δ. 2. 1

Στην δεύτερη υλοποίηση χρησιμοποιούνται κρίσιμες περιοχές. Η κεντρική ιδέα του αλγορίθμου είναι τα στοιχεία του `B` να μοιραστούν στο κάθε νήμα και το κάθε νήμα να υπολογίσει το ελάχιστο στοιχείο στο μερίδιο του. Έπειτα όλα τα νήματα πρέπει να συγκρίνουν τα αποτελέσματά τους μεταξύ τους. Στον αλγόριθμο που υλοποιεί η άσκηση τα νήματα μοιράζονται τα στοιχεία μέσω του for clause για άλλη μια φορά, διαιρώντας τον πίνακα σε γραμμές. Κάθε νήμα χρησιμοποιεί δύο συγκεκριμένες και μοναδικές σε αυτό θέσεις στον πίνακα `S`: Στο `S[tid+P]` αποθηκεύεται προσωρινά το ελάχιστο της εκάστοτε γραμμής και στο `S[tid]` αποθηκεύεται το ολικό ελάχιστο που έχει βρει το νήμα `tid`. Έτσι, μετά το `barrier` όλα τα νήματα θα έχουν στις πρώτες θέσεις του `S` τα ελάχιστα που βρήκαν. Στην συνέχεια μένει το κάθε νήμα να γράψει στην θέση `S[P]` το ελάχιστο του μόνο αν το ελάχιστο αυτό είναι πραγματικά μικρότερο από την τιμή που υπάρχει ήδη στο `S[P]`. Καθώς αυτή η διαδικασία απαιτεί η τιμή `S[P]` να μην αλλάξει από ένα νήμα όσο ένα άλλο νήμα την εξετάζει πρέπει να υπάρξει ένας τρόπος συγχρονισμού και εκεί είναι που χρειάζεται η κρίσιμη περιοχή.

Τεχνικά, οι κρίσιμες περιοχές μπορούν να υλοποιηθούν με τρεις διαφορετικούς τρόπους: atomic operations, locks και critical sections. Στην περίπτωση του αλγορίθμου το `atomic` δεν μπορεί να χρησιμοποιηθεί καθώς ο αλγόριθμος θα πρέπει να εξετάσει την τιμή `S[P]` και έπειτα να γράψει στην θέση αυτή αν η σύγκριση είναι αληθής. Το `atomic` υποστηρίζει μπλοκάρισμα για μόνο μια εντολή. Από την άλλη, αν και τα locks θα λειτουργούσαν στην συγκεκριμένη περίπτωση, οι παραπάνω διαδικασίες που χρειάζονται για την χρήση τους κάνουν την χρήση των critical sections προτιμότερη.

Σημειώνεται πως το πρώτο νήμα που θα μπει στην κρίσιμη περιοχή θα προσπαθήσει να συγκρίνει το ελάχιστο του με οποιαδήποτε τιμή είχε αφήσει στο προηγούμενο βήμα το νήμα 0 στο `S[P]`. Αυτό, ωστόσο, δεν είναι πρόβλημα καθώς το `S[P]` θα είναι το ελάχιστο της τελευταίας γραμμής που εξέτασε το νήμα 0, στοιχείο που σίγουρα είναι μεγαλύτερο ή ίσο του πραγματικού ελαχίστου του πίνακα.

Μέρος Δ. 2. 2

Στην τελευταία έκδοση του προγράμματος χρειάζεται να υλοποιηθεί ένας αλγόριθμος δυαδικού δέντρου, δηλαδή όλοι οι επεξεργαστές να επικοινωνούν με κάποιον άλλον παράλληλα με τους υπόλοιπους, μειώνοντας τον αριθμό των επεξεργαστών που συμμετέχουν στην διαδικασία κατά το ήμισυ σε κάθε βήμα.

Το πρώτο βήμα του αλγορίθμου είναι το κάθε νήμα να υπολογίσει το τοπικό του ελάχιστο, όπως προηγουμένως. Τα ελάχιστα αποθηκεύονται στον πίνακα `T` που έχει μέγεθος ίσο με τον αριθμό των νημάτων.

Πριν ξεκινήσει το μέρος του αλγορίθμου που υλοποιεί τον δυαδικό αλγόριθμο χρειάζεται να εξετασθεί αν ο αριθμός των νημάτων μπορεί να υποστηρίξει έναν τέτοιον αλγόριθμο. Για να γίνει αυτό πρώτα υπολογίζεται η μεγαλύτερη δύναμη του δύο μικρότερη από τον αριθμό των νημάτων στην μεταβλητή `P_bin` με την συνάρτηση `to_binary_tree`. Αφού υπολογισθεί αυτή η τιμή, αν τα νήματα είναι παραπάνω από δύναμη του 2, ο πίνακας T πρέπει να "μειωθεί" σε κατάλληλο μέγεθος ώστε να χρησιμοποιηθεί στο επόμενο βήμα. Φυσικά, το μέγεθος του πίνακα δεν μειώνεται πραγματικά αλλά μειώνονται οι θέσεις του που χρησιμοποιούνται. Συνεπώς, οι τιμές των έστω Π παραπανίσιων θέσεων συγκρίνονται με τις πρώτες Π θέσεις του πίνακα.

Τελικά, ο αλγόριθμος του δυαδικού δέντρου υλοποιείται σε ένα `parallel` block στο οποίο όλα τα νήματα συμμετέχουν. Χρησιμοποιώντας τις μεταβλητές `stop` και `filter` ελέγχεται το ποια

νήματα συμμετέχουν ουσιαστικά στον αλγόριθμο. Η μεταβλητή `stop` του κάθε νήματος είναι στην πραγματικότητα μια `bool` μεταβλητή που γίνεται αληθής αν μία από δύο συνθήκες αληθεύουν: Όταν το `tid` του νήματος δεν είναι μικρότερο του πλήθους νημάτων που πρέπει να συμμετάσχουν ή όταν το `tid` δεν είναι μικρότερο της τιμής του φίλτρου (`filter`). Το φίλτρο είναι μια μεταβλητή η οποία δείχνει πόσο μικρό πρέπει να είναι το `tid` ενός νήματος για να συμμετέχει στον αλγόριθμο. Για παράδειγμα, αν το φίλτρο είναι 2 μόνο τα νήματα 0 και 1 θα συμμετάσχουν. Επίσης χρησιμοποιείται η μεταβλητή `i` για την εύρεση του στοιχείου με το οποίο πρέπει να συγκρίνει το κάθε νήμα το δικό του στοιχείο. Στην πράξη, οι τιμές των `i` και `filter` θα είναι πάντα ίδιες αλλά για την συγγραφή πιο ευανάγνωστου κώδικα ορίζονται ως διαφορετικές μεταβλητές. Τέλος γίνεται χρήση της μεταβλητής `step` που στην αρχή του αλγορίθμου είναι ίση με τον αριθμό βημάτων που θα χρειαστούν για την διεκπεραίωση του αλγορίθμου. Ο λόγος που όλα τα νήματα πρέπει να ξέρουν πόσα βήματα χρειάζεται ο αλγόριθμος είναι επειδή στην πραγματικότητα όλα τα νήματα συμμετέχουν στον αλγόριθμο: απλώς τα νήματα με `stop` αληθές δεν πραγματοποιούν κάποια σύγκριση. Ο λόγος που ο κώδικας είναι σχεδιασμένος έτσι κρύβεται στον τρόπο που λειτουργεί η διαταγή `barrier`. Για την σωστή λειτουργία του αλγορίθμου, κάθε βήμα πρέπει να γίνεται συγχρονισμένα. Αν, για παράδειγμα, το νήμα 0 ήταν στο βήμα 4 ενώ το βήμα 1 ήταν ακόμα στο βήμα 3 ο αλγόριθμος δεν θα λειτουργούσε σωστά. Το πρόβλημα που δημιουργείται εδώ είναι ότι η `barrier` περιμένει όλα τα νήματα που έχουν οριστεί στο πρόγραμμα να φτάσουν στο σημείο συγχρονισμού. Για τον λόγο αυτόν, όλα τα νήματα πρέπει να καλέσουν την `barrier` ενώ ταυτόχρονα χρειάζεται να ξέρουν αν πρέπει να κάνουν κάτι στο κάθε βήμα χωρίς την ανάγκη άλλης επικοινωνίας.

Ενδεικτικά Τρεξίματα (II)

Σε αυτήν την ενότητα παρουσιάζονται εκτελέσεις και των τριών προγραμμάτων για τον ίδιο πίνακα.

```
ajax@ubuntuDesktop:~/Documents/PS1$ ./ex1_1 735
Project I (d1 Implementation - Reduction Clause)
Loaded test 735: 32x32 (1024 numbers)
Threads: 4

 -> Initial check: OK
Time A: 0.000607

 -> Max: 38773
Time B: 0.000006

 -> Matrix B: OK (will be displayed/outputed in the end)
Time C: 0.000007

 -> Min in B: 36735
Time D: 0.000005

Total Time: 0.000672 sec

B Matrix written in B_matrix.txt
```

Εικόνα 4: Έκδοση 1 - Πίνακας 32x32

Η εκτύπωση του πίνακα Β γίνεται σε αρχείο. Στους ακόμα μεγαλύτερους πίνακες δεν θα γίνεται καθόλου.

```
ajax@ubuntuDesktop:~/Documents/PS1$ ./ex1_2 735
Project I (d2.1 Implementation - Using Critical Sections)
Loaded test 735: 32x32 (1024 numbers)
Threads: 3

 -> Initial check: OK
Time A: 0.000315

 -> Max: 38773
Time B: 0.000004

 -> Matrix B: OK (will be displayed/outputed in the end)
Time C: 0.000005

 -> Min in B: 36735
Time D: 0.000007

Total Time: 0.000361 sec

B Matrix written in B_matrix.txt
```

Εικόνα 5: Έκδοση 2 - Πίνακας 32x32

```
ajax@ubuntuDesktop:~/Documents/PS1$ ./ex1_3 735
Project I (d2.2 Implementation - Binary tree)
Loaded test 735: 32x32 (1024 numbers)
Threads: 3

 -> Initial check: OK
Time A: 0.000322

 -> Max: 38773
Time B: 0.000004

 -> Matrix B: OK (will be displayed/outputed in the end)
Time C: 0.000005

 -> Using 2 threads for the binary tree
 -> Min in B: 36735
Time D: 0.000015

Total Time: 0.000382 sec

B Matrix written in B_matrix.txt
```

Εικόνα 6: Έκδοση 3 - Πίνακας 32x32

Προβλήματα κατά την υλοποίηση

Ένα πρόβλημα που το πρόγραμμα εμφανίζει είναι στον έλεγχο του πίνακα στο πρώτο μέρος. Αν και για πίνακες της τάξης των 10 εκατομμυρίων αριθμών το πρόγραμμα συμπεριφέρεται σωστά, καθώς τα μεγέθη των πινάκων ξεπερνούν τα 100 εκατομμύρια θα υπάρξουν φορές που ο αλγόριθμος θα συμπεράνει πως κάποιες γραμμές δεν αντιστοιχούν σε ΑΔΔ πίνακα παρόλο που σε άλλες δοκιμές ο ίδιος πίνακας δεν θα προκαλεί πρόωρο τερματισμό του προγράμματος.

Αν και δεν μπόρεσα να εντοπίσω τι προκαλεί αυτήν την συμπεριφορά φαίνεται πως κάποιες γραμμές μπορεί να προκαλούσαν overflow στην μεταβλητή `sum`. Αυξάνοντας τον τύπο της σε `unsigned long long` ενώ ταυτόχρονα μειώνοντας τις τιμές των αριθμών στις γραμμές βοήθησαν αλλά φαίνεται πως το πρόβλημα δεν είναι μόνο εκεί.

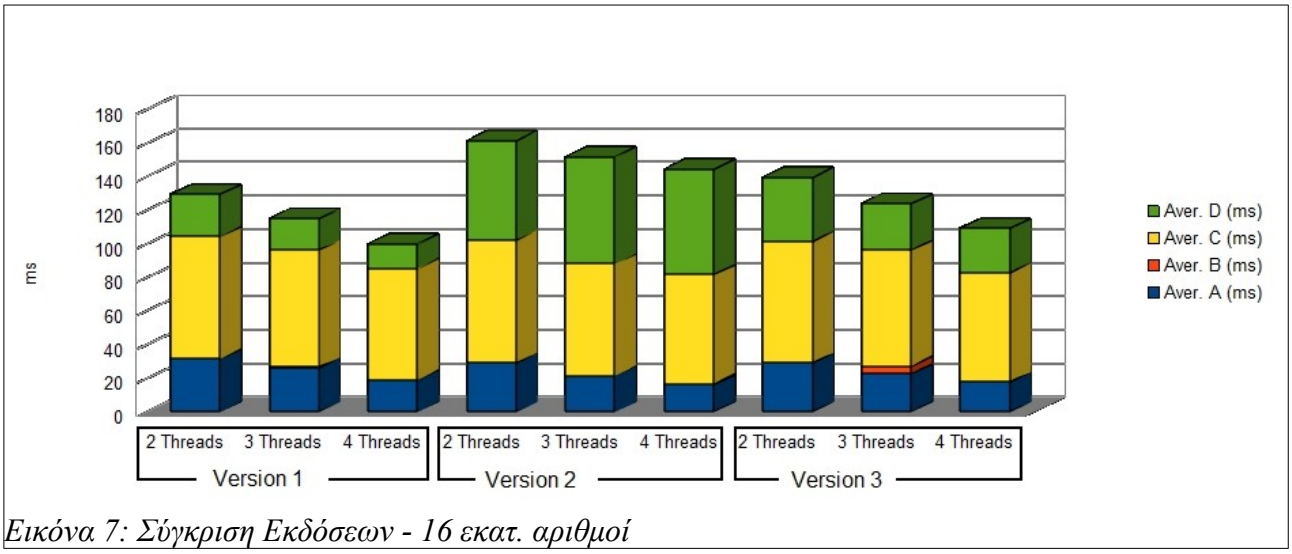
Για την αποσφαλμάτωση του συγκεκριμένου προβλήματος χρησιμοποιήθηκε το εργαλείο `gdb` το όποιο, δυστυχώς, δεν μπόρεσε να εντοπίσει κάτι. Μάλιστα, ακόμα και πριν τις αλλαγές στον τύπο της `sum` όλες οι δοκιμές στο `gdb` εκτελέστηκαν σωστά, σε αντίθεση με πολλές από τις εκτελέσεις για τις χρονομετρήσεις που εμφάνισαν σφάλματα.

Στην τωρινή μορφή του, ο κώδικας δημιουργεί αυτό το πρόβλημα σχετικά σπάνια, με περίπου μια στις δεκαπέντε εκτελέσεις να παράγουν ψευδή αποτελέσματα. Το πρόγραμμα μπορεί να λειτουργήσει με πίνακες έως και περίπου 200000 x 200000 . Πίνακες μεγαλύτερων διαστάσεων δεν εξετάσθηκαν. Στην εικονική μηχανή του πανεπιστημίου το πρόγραμμα φαίνεται να λειτουργεί αλλά εξετάσθηκε με πίνακες μεγέθους έως και περίπου 3 εκατ. αριθμών καθώς λόγω του περιορισμένου χώρου ανά χρήστη οι μεγαλύτεροι πίνακες δεν μπορούσαν να αποθηκευτούν.

Αναλυτικές Μετρήσεις

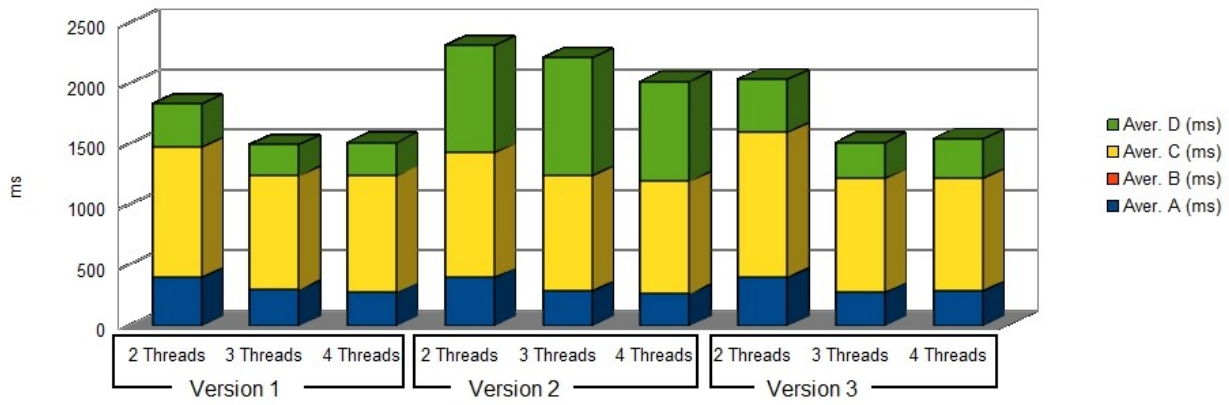
Σε αυτήν την ενότητα θα παρουσιαστούν οι χρονικές επιδόσεις του προγράμματος για διαφορετικών μεγεθών πίνακες όπως και για διαφορετικών αριθμών νημάτων. Καθώς όλες οι δοκιμές που παρουσιάζονται πραγματοποιήθηκαν στο μηχάνημα Ubuntu που αναφέρεται στην εισαγωγή έχουν δοκιμαστεί μόνο τρεξίματα με 2, 3 και 4 νήματα.

Στο παρακάτω γράφημα φαίνονται οι επιδόσεις των διαφορετικών εκδόσεων με διαφορετικούς αριθμούς νημάτων. Όπως είναι αναμενόμενο, καθώς τα νήματα αυξάνονται ο χρόνος εκτέλεσης μειώνεται.



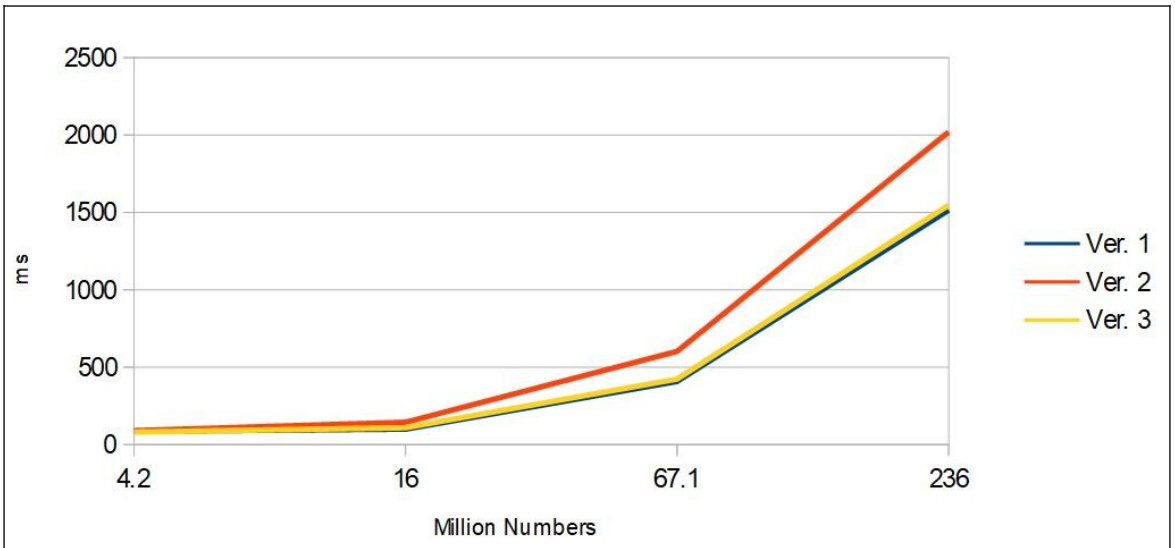
Εικόνα 7: Σύγκριση Εκδόσεων - 16 εκατ. αριθμοί

Το μέρος Β είναι πάντα αμελητέα μικρό ενώ το μέρος Γ είναι πάντα το μεγαλύτερο. Επίσης στα γραφήματα φαίνεται πως η έκδοση με τις κρίσιμες περιοχές είναι πάντα η πιο αργή. Αυτό είναι λογικό, βέβαια, αφού καθώς τα νήματα περιμένουν να μπουν στην κρίσιμη περιοχή δεν αξιοποιούνται, σε αντίθεση με τους άλλους αλγορίθμους που τα νήματα δουλεύουν παράλληλα για τον υπολογισμό του ελαχίστου. Οι εκδόσεις 1 και 3 είναι συγκρίσιμες χρονικά, με την έκδοση 1 να είναι ελαφρώς πιο γρήγορα



Εικόνα 8: Σύγκριση Εκδόσεων - 236 εκατ. αριθμοί

Στην εικόνα 9 φαίνονται οι χρόνοι της κάθε έκδοσης προς όλο και μεγαλύτερα αρχεία πινάκων. Όπως είναι λογικό, η διαφοροποίηση στους χρόνους γίνεται φανερή στα πιο μεγάλα μεγέθη πινάκων. Η έκδοση 2 είναι πράγματι η πιο αργή ενώ οι εκδόσεις 1 και 3 πετυχαίνουν περίπου τους ίδιους χρόνους. Σημειώνεται πως οι χρόνοι που απεικονίζονται είναι τα αθροίσματα των μέσων όρων του κάθε μέρους των αλγορίθμων.



Εικόνα 9: Σύγκριση Εκδόσεων προς Μέγεθος Πίνακα (4 νήματα)

Σχετικά με τις αναλυτικές μετρήσεις

Κάθε συνδυασμός εκτέλεσης (έκδοση προγράμματος, αριθμός νημάτων) πραγματοποιήθηκε αρκετές φορές και τα αποτελέσματα συνδυάστηκαν δημιουργώντας μέσους χρόνους. Αυτοί οι μέσοι όροι χρόνου εκτέλεσης είναι που έχουν παρουσιαστεί.

Η αρχική ιδέα ήταν ο κάθε συνδυασμός να εκτελεστεί πέντε φορές και τα αποτελέσματα αυτά να συνδυαστούν. Ωστόσο, κάποια αποτελέσματα θα τύχαινε να απέχουν θεαματικά από τα άλλα αντίστοιχά τους, όπως για παράδειγμα έτυχε στην "έκδοση 1, 4 νήματα" μια δοκιμή να χρειαστεί 2.7s για το μέρος Γ όταν όλες οι άλλες εκτελέσεις χρειάστηκαν το πολύ 1s. Έτσι, κάποιες δοκιμές επαναλήφθηκαν και τα αποτελέσματα με μεγάλες αποκλίσεις απορρίφθηκαν. Οι επαναληπτικές δοκιμές έγιναν με ένα bash script.

Οι μέσοι χρόνοι που χρησιμοποιήθηκαν φαίνονται παρακάτω.

Version	Threads	Test Used	Aver. A (ms)	Aver. B (ms)	Aver. C (ms)	Aver. D (ms)	Aver. Total (ms)
1	2	520 (16M)	31.56	0.17	72.77	25.01	**138.40**
2	2	520 (16M)	29.59	0.21	72.87	58.87	**170.70**
3	2	520 (16M)	29.15	0.19	72.49	37.88	**148.77**
1	3	520 (16M)	26.63	0.14	69.77	18.65	**124.83**
2	3	520 (16M)	21.64	0.13	66.58	63.61	**161.22**
3	3	520 (16M)	23.12	4.34	68.97	27.81	**130.56**
1	4	520 (16M)	19.13	0.20	66.37	14.48	**109.64**
2	4	520 (16M)	16.65	0.12	65.17	62.61	**154.10**
3	4	520 (16M)	17.79	0.11	65.30	26.07	**118.37**
1	2	529 (236M)	395.32	0.90	1079.89	354.82	**1958.43**
2	2	529 (236M)	394.85	0.92	1033.38	884.29	**2439.13**
3	2	529 (236M)	399.14	0.89	1198.57	439.44	**1970.14**
1	3	529 (236M)	294.19	0.63	944.36	255.03	**1621.11**
2	3	529 (236M)	289.30	0.66	949.86	973.66	**2339.10**
3	3	529 (236M)	277.45	0.64	935.89	300.76	**1640.19**
1	4	529 (236M)	276.27	0.62	961.62	276.37	**1643.48**
2	4	529 (236M)	260.54	0.79	931.41	825.82	**2144.12**
3	4	529 (236M)	284.97	0.52	928.31	332.75	**1672.59**

Οι χρόνοι για τον πίνακα με 67.1 εκατομμύρια προέκυψαν από μέσους όρους δύο δοκιμών για κάθε εκτέλεση.

Version	Threads	Test Used	Aver. A (ms)	Aver. B (ms)	Aver. C (ms)	Aver. D (ms)	Aver. Total (ms)
1	2	178 (67.1M)	123.78	0.39	302.92	100.91	**564.08**
2	2	178 (67.1M)	136.66	0.41	299.04	223.50	**695.22**
3	2	178 (67.1M)	118.96	0.40	306.55	133.43	**595.17**
1	3	178 (67.1M)	76.79	0.32	274.67	80.14	**469.72**
2	3	178 (67.1M)	76.03	0.31	277.94	253.30	**644.54**
3	3	178 (67.1M)	79.15	0.31	273.79	91.75	**481.60**
1	4	178 (67.1M)	84.43	0.25	260.98	62.59	**445.11**
2	4	178 (67.1M)	75.55	0.25	265.57	260.96	**638.60**
3	4	178 (67.1M)	77.79	0.25	262.53	83.54	**461.63**