Βασίλης Κυριαφίνης: 9797 Κόρο Έρικα: 9707

1ST ASSINGNMENT

parallel and destributed systems 2021-2022

Περιεχόμενα

[Εισαγωγή 1](#_Toc88000150)

[Σειριακός Αλγόριθμος 2](#_Toc88000151)

[Pthread library 3](#_Toc88000152)

[Belgium OSM 3](#_Toc88000153)

[com-YouTube 4](#_Toc88000154)

[Mycielskian 13 4](#_Toc88000155)

[OpenMP API 5](#_Toc88000156)

# Εισαγωγή

Στόχος της παρούσης εργασίας ήταν η σύγκριση τεσσάρων μεθόδων εκτέλεσης ενός αλγορίθμου. Αρχικά, υλοποιήθηκε ο αλγόριθμος C σε σειριακή εκτέλεση για το ζητούμενο πρόβλημά μας και αξιοποιώντας αυτόν, με σκοπό την βελτίωση του χρόνου εκτέλεσής του, πραγματοποιήθηκαν τρεις ακόμη εκδοχές του με την χρήση των βιβλιοθηκών Pthread, OpenMP και OpenCilk.

Για την υλοποίηση των παραπάνω και για την αξιολόγησή τους εισήχθησαν πέντε γράφοι. Για τον κάθε γράφο υπολογίστηκε ο αριθμός των τριγώνων με την μέθοδο:

και

όπου Α ο αραιός πίνακας του κάθε γράφου, e ένας πίνακας στήλη με όλα τα στοιχεία του ίσα με τη μονάδα και c ο πίνακας του οποίου το κάθε στοιχείο υποδηλώνει σε πόσα τρίγωνα συμμετέχει ο αντίστοιχος κόμβος.

Να σημειωθεί ότι για την αποθήκευση των αραιών πινάκων στην μνήμη χρησιμοποιήθηκε η αναπαράσταση Compressed Sparse Row (CSR) η οποία όμως λόγω της συμμετρίας των πινάκων είναι ίδια με την Compressed Sparse Column (CSC).

Ο επεξεργαστής που χρησιμοποιήθηκε για τις δοκιμές είναι ο intel core i7-7700HQ [[1]](#footnote-1) ο οποίος διαθέτει 4 φυσικούς και 8 λογικούς πυρήνες. Το ρολόι είναι χρονισμένο στα 2.8 GHz και το Max Turbo Frequency είναι 3.8 GHz. Επίσης, διαθέτει 6 MB L3 cache, 4x256 Kbytes L2 cache, 4x32 Kbytes L1 instruction cache και 4x32 Kbytes L1 data cache.

Για το Compilation όλων των αρχείων πηγαίου κώδικα χρησιμοποιήθηκε η σημαία -O3 στον gcc με την οποία ο compiler παράγει εκτελέσιμα με βασικό γνώμονα τις επιδόσεις και όχι το τελικό μέγεθος του αρχείου.

# Σειριακός Αλγόριθμος

Ο σειριακός αλγόριθμος δέχεται ως είσοδο έναν πίνακα CSR, υπολογίζει τον πολλαπλασιασμό με τον εαυτό του εξαιρώντας τον υπολογισμό στις μηδενικές θέσεις του αραιού πίνακα. Με τον τρόπο αυτό υπολογίζεται αυτόματα και το “προϊόν” Hadamard. Στη συνέχεια αθροίζονται οι τιμές όλων των στοιχείων του παραγόμενου CSR πίνακα και διαιρώντας με το έξι προκύπτει ο αριθμός των τριγώνων του αρχικού γράφου.

Ο πολλαπλασιασμός του CSR πίνακα με τον εαυτό του γίνεται με ένα διπλό for loop στο οποίο μία μία οι γραμμές του πίνακα πολλαπλασιάζονται με τις στήλες του. Ο πολλαπλασιασμός της κάθε γραμμής με την κάθε στήλη υπολογίζεται μετρώντας τα κοινά στοιχεία τους. Με αυτόν τον τρόπο το αποτέλεσμα αποθηκεύεται κατευθείαν σε μορφή CSR.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Belgium\_osm | com\_YouTube | Mycielskian13 | LAW | NACA0015 |
| Time | 79.267 ms | 4.887 s | 1.236 s | 153.826 ms | 699.452 ms |

Τα παραπάνω αποτελέσματα όπως και τα αποτελέσματα που θα παρουσιαστούν παρακάτω αποτελούν τον μέσο όρο 18 εκτελέσεων από τις 20 συνολικές, καθώς οι δύο πρώτες δεν υπολογίστηκαν στον μέσο όρο, του αλγορίθμου για κάθε έναν από τους γράφους. Επιλέχθηκε αυτός ο τρόπος παρουσίασης των τελικών αποτελεσμάτων γιατί οι χρόνοι διαφέρουν από εκτέλεση σε εκτέλεση, λόγω εξωτερικών του προγράμματος παραγόντων όπως άλλα προγράμματα, προγραμματισμένες διεργασίες του λειτουργικού συστήματος κ.ά.

# Pthread library

Για την πρώτη παράλληλη υλοποίηση χρησιμοποιήθηκε η βιβλιοθήκη Pthread. Στην πορεία, δοκιμάστηκαν διάφορες τεχνικές παραλληλοποίησης μεταξύ των οποίων και τα thread pools, τα οποία προσφέρουν καλύτερη κατανομή του συνολικού φόρτου εργασίας ανάμεσα στα threads αλλά λόγω της έντονης ανάγκης για συγχρονισμό τους, η απόδοση δεν ήταν η αναμενόμενη. Τελικά, ως καλύτερη λύση βρέθηκε η πλήρης ανεξαρτητοποίηση των threads μεταξύ τους ώστε να μην υπάρχει ανάγκη συγχρονισμού όσο αυτά εκτελούνται.

Ο αλγόριθμος που χρησιμοποιήθηκε είναι κατά βάση ο ίδιος με την σειριακή υλοποίηση. Η κατανομή της εργασίας στα thread έγινε σε επίπεδο γραμμών του πίνακα (εξωτερική for), δηλαδή το κάθε thread ήταν υπεύθυνο να υπολογίσει το γινόμενο γραμμών με όλες τις στήλες του πίνακα όπου,

Για την ανίχνευση της βέλτιστης απόδοσης της επίλυσης του ζητούμενου (υπολογισμός τριγώνων) για τους πέντε γράφους δοκιμάστηκε διαφορετικός αριθμός threads και παρατηρήθηκε ότι ο κάθε γράφος ευνοείται διαφορετικά με τη μεταβολή του πλήθους τους. Ακολουθεί γραφική αναπαράσταση της συμπεριφοράς τριών παραδειγματικών γράφων σε σχέση με τα threads.

## com-YouTube

Τα τρίγωνα του YouTube γράφου είναι 3056386. Από το διάγραμμα φαίνεται ότι ο γράφος εκτελείται καλύτερα για 986 threads με χρόνο εκτέλεσης 1,817 s δηλαδή περίπου 2,7 φορές πιο γρήγορα από την σειριακή εκτέλεση. Από τα 986 threads και πάνω φαίνεται ότι δεν υπάρχει κάποια βελτίωση καθώς ο χρόνος παραμένει σχετικά σταθερός.

## Mycielskian 13

Τα τρίγωνα του Mycielskian είναι 0. Από το διάγραμμα φαίνεται ότι ο γράφος εκτελείται καλύτερα για 66 threads με χρόνο εκτέλεσης 349.78 ms δηλαδή περίπου 3.5 φορές πιο γρήγορα από την σειριακή εκτέλεση. Από τα 66 threads και πάνω φαίνεται ότι δεν υπάρχει κάποια βελτίωση καθώς ο χρόνος παραμένει σχετικά σταθερός.

## Naca 0015

Τα τρίγωνα του συγκεκριμένου γράφου είναι 2075635. Από το διάγραμμα φαίνεται ότι ο naca-0015 εκτελείται πιο γρήγορα για 8 threads με χρόνο εκτέλεσης 153.19 ms δηλαδή περίπου 4.6 φορές πιο γρήγορα από την σειριακή εκτέλεση. Όμοια με τον dblp-2010 φαίνεται ότι ο γράφος αυτός δεν ευνοείται από μεγαλύτερο αριθμό threads.

# OpenMP API

Η δεύτερη τεχνική παραλληλοποίησης που χρησιμοποιήθηκε είναι η OpenMP, η οποία εφαρμόστηκε για την ταυτόχρονη εκτέλεση του εξωτερικού for loop δηλαδή σε επίπεδο γραμμών, όπως και στην pthreads. Παρατίθεται το προσχέδιο του κώδικα:

omp\_set\_num\_threads(numberOfThreads);

#pragma omp parallel for default(none) schedule(type, chunk\_size)\ shared(sharedVars) private(privateVars) reduction(+: triangleCounter)  
 *// For every row of the matrix...* for (int row = 0; row < input->size; row++) *// For every column of the i row...*

for (int col = rowStart; col <= rowEnd; ++col) {

*// Measure the common elements of the row and the column...*  
 triangleCounter += measureCommonElements()  
 }  
 }

Για την εύρεση της βέλτιστης απόδοσης του προγράμματος ο κώδικας παραμετροποιήθηκε μεταβάλλοντας τα ορίσματα του schedule. Πιο συγκεκριμένα, δόθηκαν στην παράμετρο type οι τιμές static, dynamic και guided, εφαρμόζοντας για την κάθε μία και διαφορετικό chunk size. Στον πίνακα που ακολουθεί παρατίθενται τα αποτελέσματα:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Chunk size | 1 | 5 | 10 | 15 | 25 | 50 | 100 |
| static | 360 ms | 350 ms | 340 ms | 350 ms | 370 ms | 380 ms | 390 ms |
| Dynamic | 320 ms | 320 ms | 310 ms | 320 ms | 330 ms | 370 ms | 420 ms |
| Guided | 540 ms | | | | | | |

# OpenCilk

Η υλοποίηση της OpenCilk έγινε και αυτή με βάση τον σειριακό αλγόριθμο, και εφαρμόστηκε για την ταυτόχρονη εκτέλεση του εξωτερικού for loop δηλαδή σε επίπεδο γραμμών.

*// For every row of the matrix...*cilk\_for (int row = 0; row < input->size; row++) *// For every column of the i row...*

for (int col = rowStart; col <= rowEnd; ++col) {

*// Measure the common elements of the row and the column...*  
 }  
}

1. <https://ark.intel.com/content/www/us/en/ark/products/97185/intel-core-i77700hq-processor-6m-cache-up-to-3-80-ghz.html> [↑](#footnote-ref-1)