1ST ASSIGNMENT

PARALLEL AND DESTRIBUTED SYSTEMS 2021-2022

Βασίλης Κυριαφίνης: 9797 Κόρο Έρικα: 9707

Περιεχόμενα

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:

$$c = C \cdot e \cdot ½ \text{ και } C = A \odot (A \cdot A)$$

όπου A 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).

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

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 n γραμμών με όλες τις στήλες του πίνακα όπου,

$$n = \frac{matrix\ size}{number\ of\ threads}$$

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:

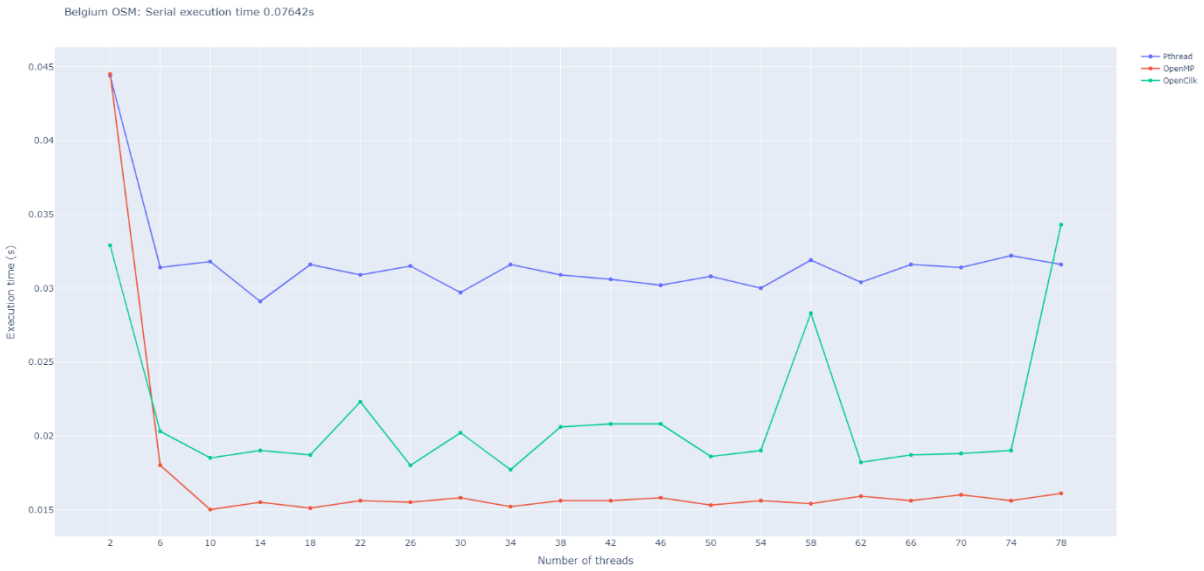
Chunk size	*1*	*5*	*10*	*15*	*25*	*50*	*100*
static	360 ms	350 ms	340 ms	350 ms	370 ms	380 ms	390 ms
Dynamic	320 ms	320 ms	310 ms	320 ms	330 ms	370 ms	420 ms
Guided				540 ms			

OpenCilk

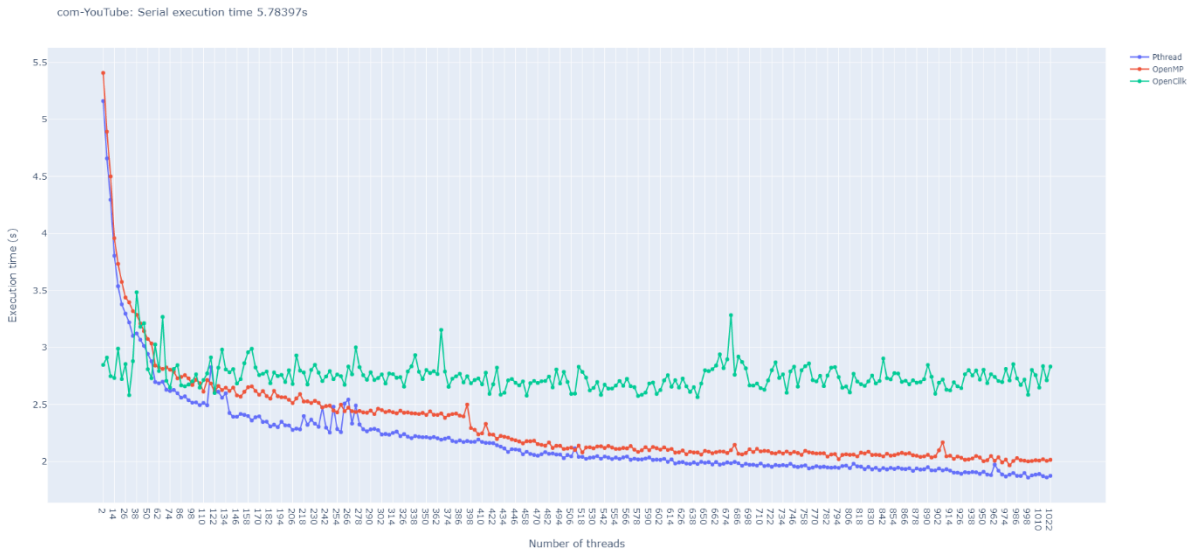
Η υλοποίηση της OpenCilk έγινε και αυτή με βάση τον σειριακό αλγόριθμο, και εφαρμόστηκε για την ταυτόχρονη εκτέλεση του εξωτερικού for loop δηλαδή σε επίπεδο γραμμών.

```
// 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...
    }
}
```

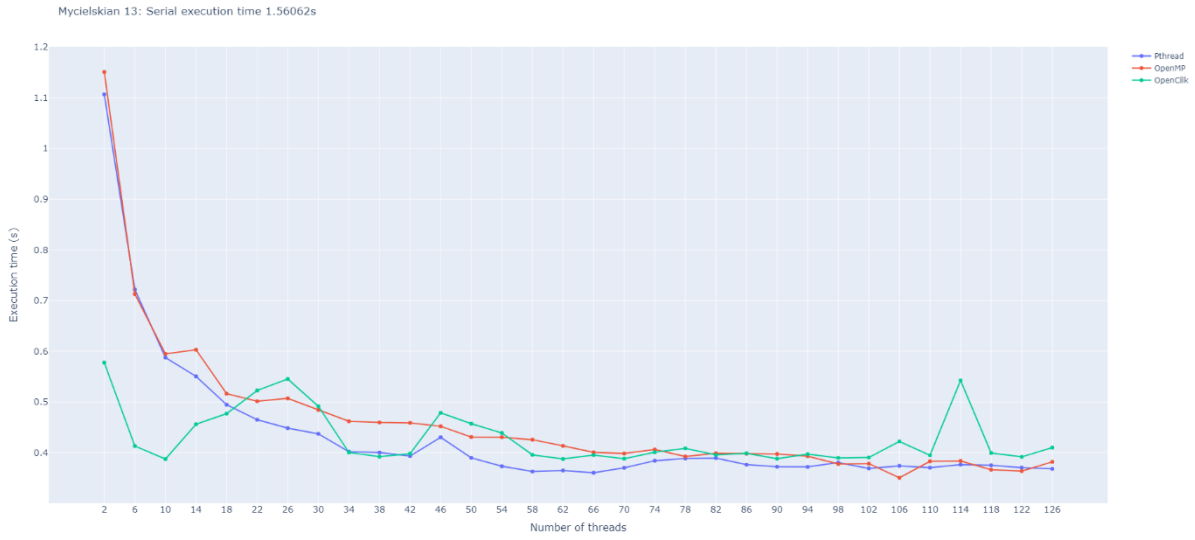
Performance data



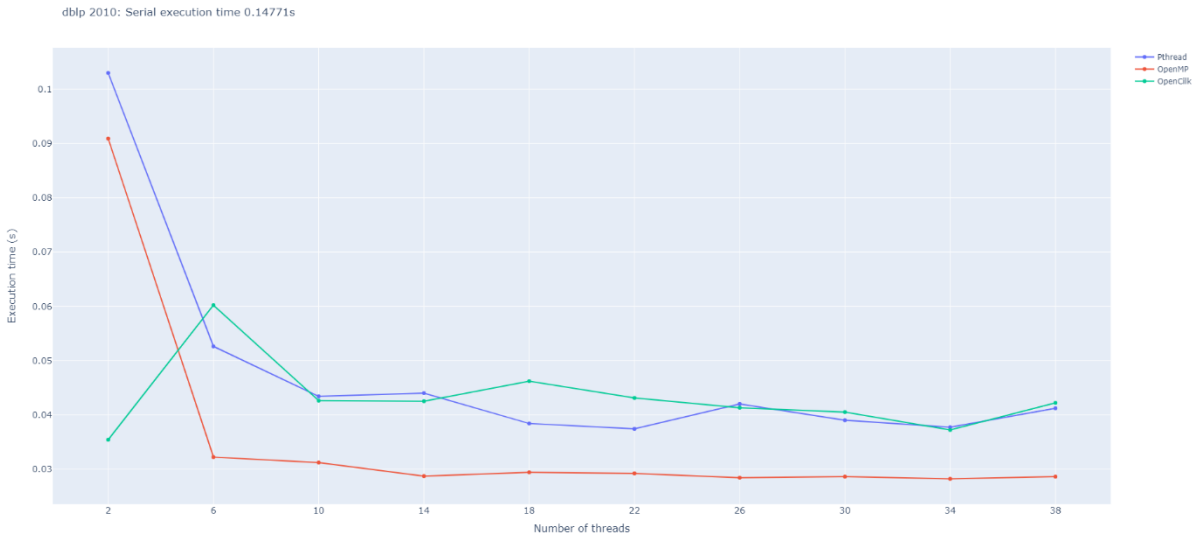
Γράφημα 1 Belgium OSM



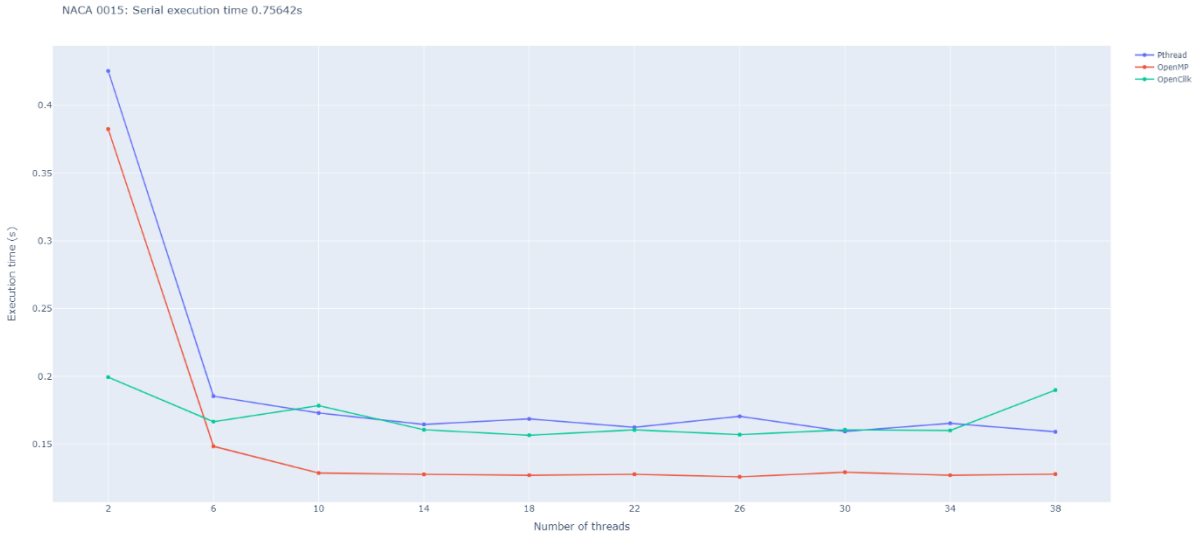
Γράφημα 2 com-YouTube



Γράφημα 3 Mycielskian 13



Γράφημα 4 Dblp-2010



Γράφημα 5 NACA 0015