1ST ASSIGNMENT

PARALLEL AND DESTRIBUTED SYSTEMS 2021-2022

Βασίλης Κυριαφίνης: 9797 Κόρο Έρικα: 9707

Περιεχόμενα

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.

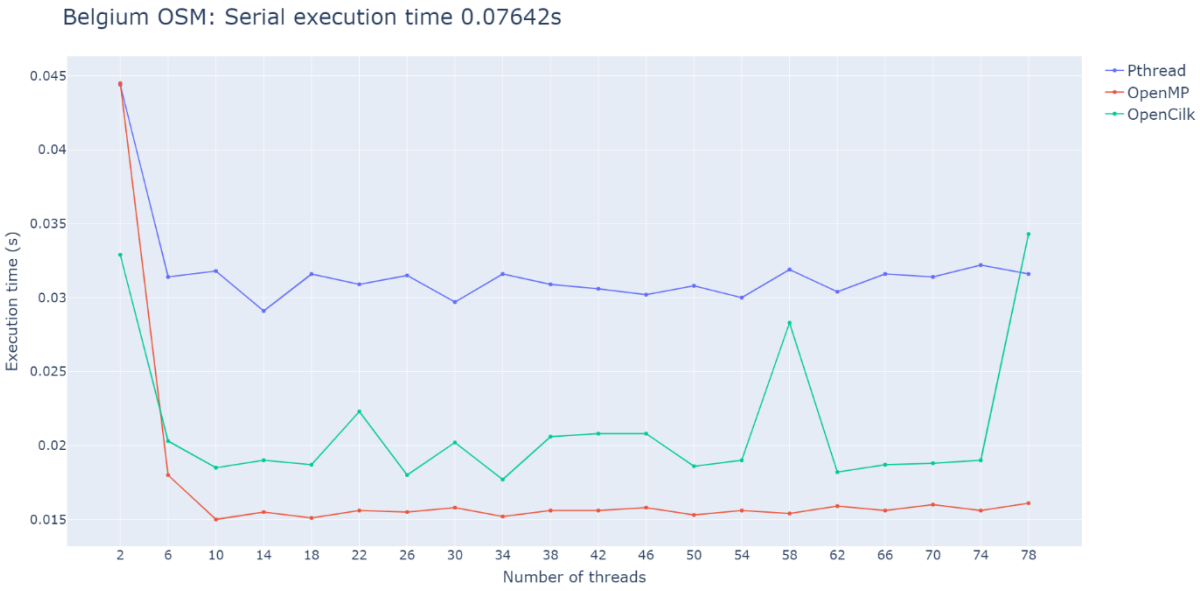
```
// For every row of the matrix...
cilk_for (int row = 0; row < input->size; row++)
    // For every column of the i row...
    for (int col = rowStart; col <= rowEnd; ++col) {
        // Measure the common elements of the row and the column...
    }
}
```

Κώδικας

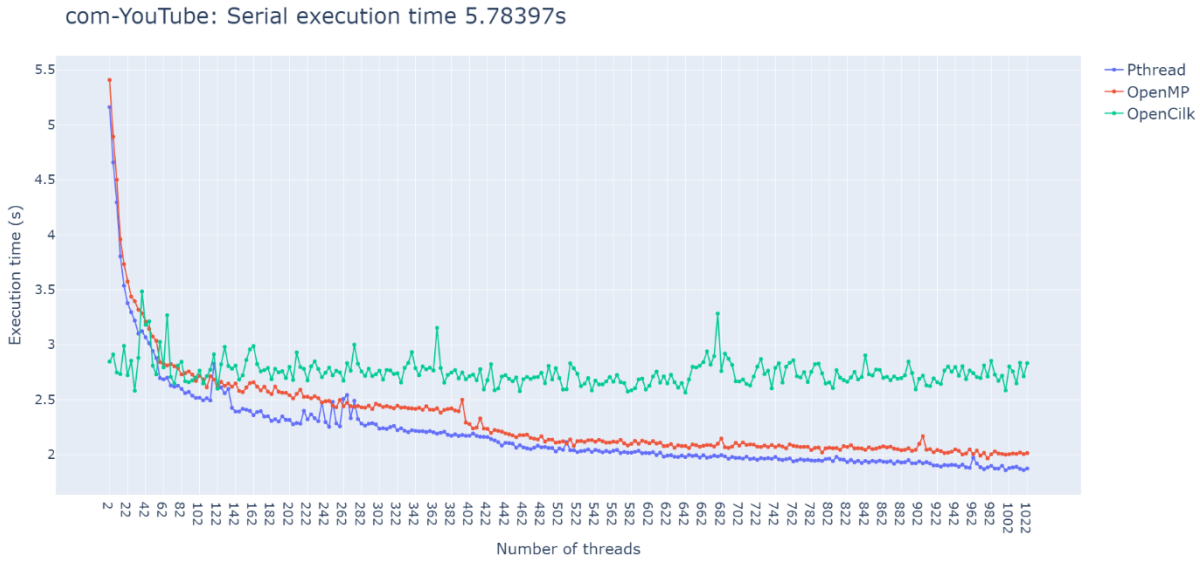
Ο πηγαίος κώδικας της εργασίας βρίσκεται στο GitHub[1] μαζί με οδηγίες για την χρήση του.

[1] https://github.com/Billkyriaf/pds_assignment_1

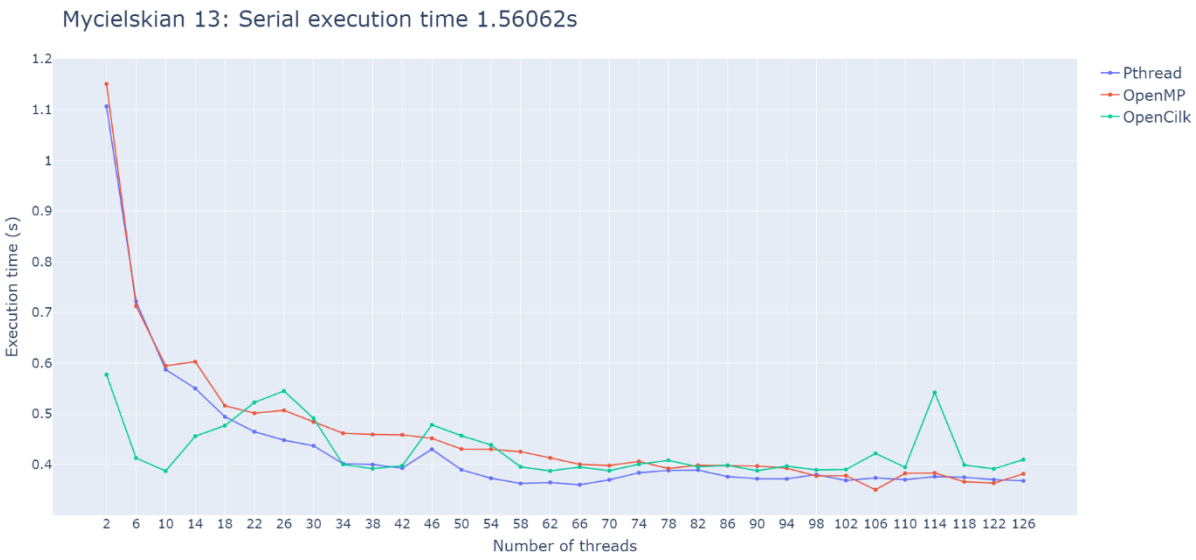
Performance data



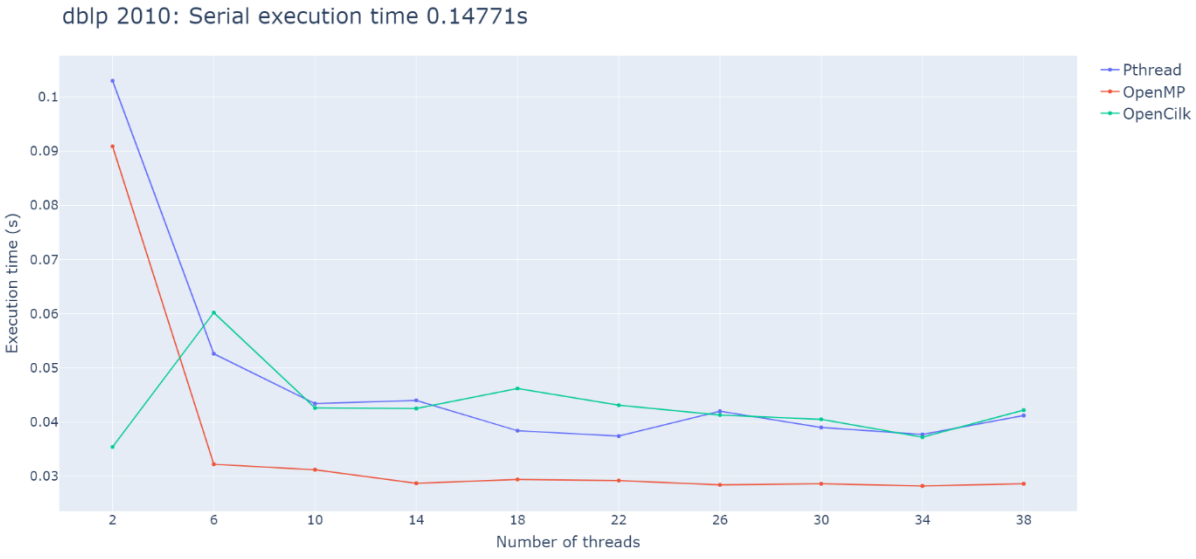
Γράφημα 1 Belgium OSM



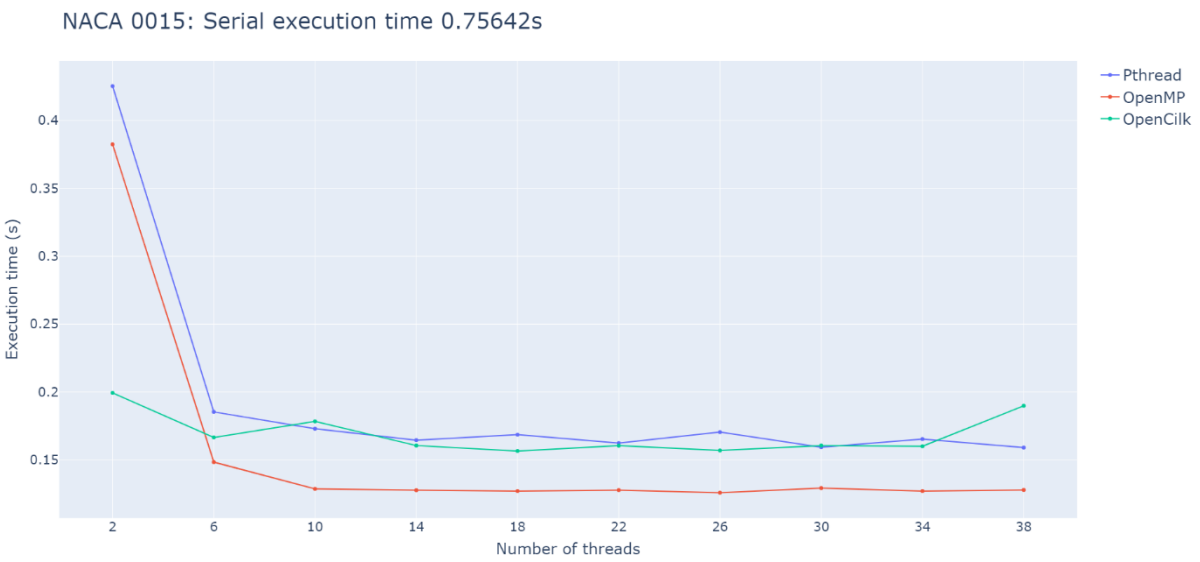
Γράφημα 2 com-YouTube



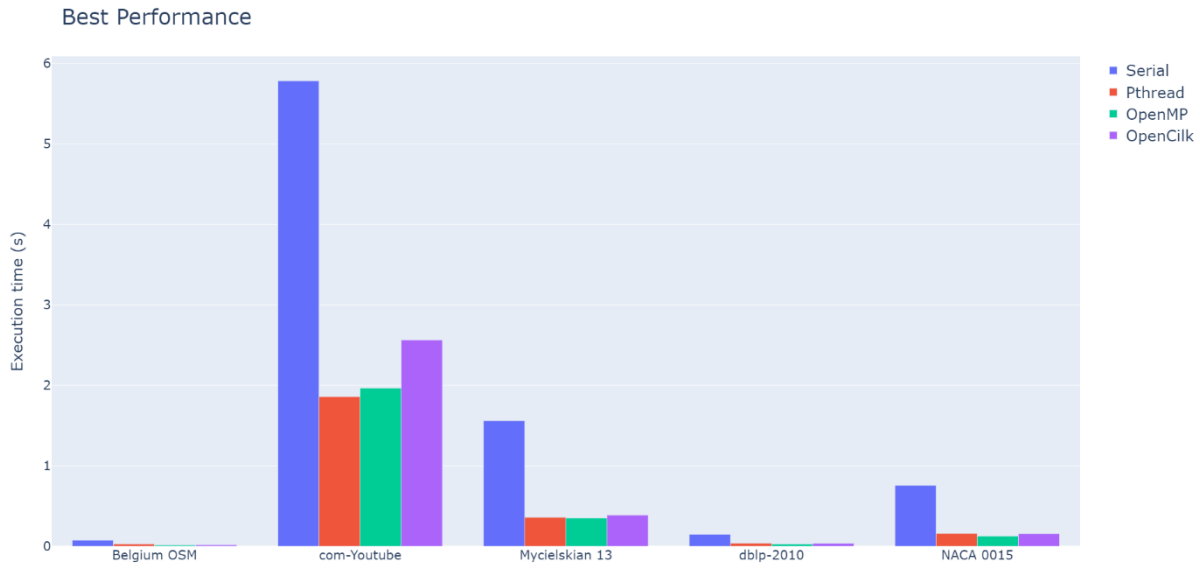
Γράφημα 3 Mycielskian 13



Γράφημα 4 Dblp-2010



Γράφημα 5 NACA 0015



Γράφημα 6 Best performance

Συμπεράσματα

Από τα γραφήματα 1 ως 5 γίνεται αντιληπτό ότι ο κάθε γράφος εμφάνισε την καλύτερη επίδοσή του για διαφορετικό αριθμό threads. Ο μεγάλος αριθμός threads σε κάποιους από τους γράφους μεταξύ άλλων οφείλεται στην αποδοτικότητα της χρήσης της μνήμης cache του αλγορίθμου. Μεγαλύτεροι γράφοι με περισσότερα μη μηδενικά στοιχεία ανά γραμμή και στήλη απαιτούν περισσότερη μνήμη cache για τον υπολογισμό του γινομένου μιας γραμμής με μία στήλη.

Ως αποτέλεσμα αυτού τα threads που υπολογίζουν γραμμές και στήλες με πολλά στοιχεία περνούν περισσότερο χρόνο σε “waiting state” περιμένοντας την μεταφορά των δεδομένων από ψηλότερα επίπεδα cache. Στο χρονικό διάστημα αυτό η εκτέλεση του προγράμματος σταματάει. Αυξάνοντας τον αριθμό των threads όταν κάποιο από τα threads μπαίνει σε “waiting state” αυξάνεται η πιθανότητα το λειτουργικό να ξυπνήσει κάποιο thread του προγράμματος το οποίο περίμενε για δεδομένα και την στιγμή εκείνη βρίσκεται σε “ready state”.

Από το γράφημα 6 μπορεί να γίνει μια σύγκριση μεταξύ των τριών υλοποιήσεων, Pthread, OpenMP και OpenCilk. Η λογική και των τριών υλοποιήσεων είναι η ίδια και σε καμιά από τις τρεις δεν υπάρχουν blocking conditions (mutex locks). Το γεγονός αυτό συνεπάγεται ότι η Pthread υλοποίηση είναι και η πιο γρήγορη για μεγαλύτερους γράφους καθώς διαχειρίζεται τα threads σε χαμηλότερο επίπεδο από τα άλλα δύο APIs και δαπανάται λιγότερος χρόνος εκτέλεσης ο οποίος σχετίζεται με την διαχείριση των threads.