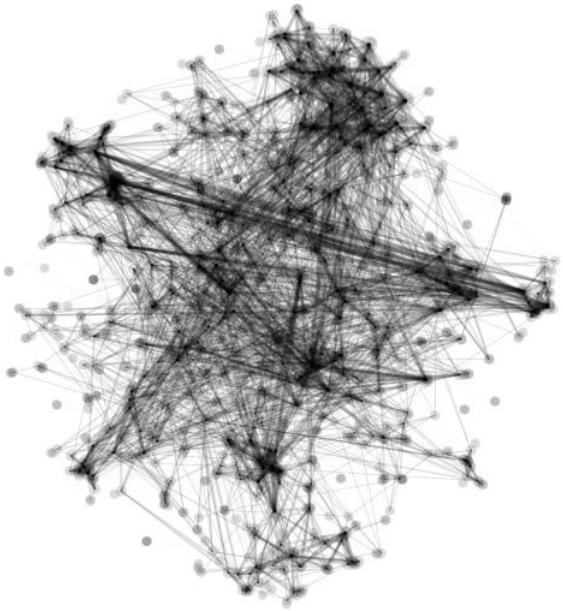


Υπολογισμός τριγώνων σε αραιά γραφήματα

Νίκος Τζάτσης 9701

Παράλληλα και Διανεμημένα Συστήματα

21 / 9 / 21

Λήψη Αραιών Πινάκων Γειτνίασης

Για να πάρουμε τους αραιούς CSC πίνακες γειτνίασης με format Matrix Market τροποποιήσαμε ελαφρώς το example_read.c από το https://math.nist.gov/MatrixMarket/mmio-c.html , ώστε να διαβάζει δυαδικούς πίνακες. Έτσι, πήραμε τον κάτω τριγωνικό πίνακα σε μορφή COO και με βάση αυτόν φτιάξαμε τον πάνω τριγωνικό σε μορφή COO. Μετά μετατρέψαμε τους δύο πίνακες σε μορφή CSC και έπειτα τους ενώσαμε για να πάρουμε ολόκληρο τον πίνακα γειτνίασης σε μορφή CSC. Για την μετατροπή πινάκων από COO σε CSC χρησιμοποιήθηκε η coo2csc( ) συνάρτηση, που βρίσκεται στο codebase του docker του μαθηματος, καθώς και μικρές τροποποιήσεις της.

Σειριακή Υλοποίηση της Συνάρτησης masked_sparse_matrix_matrix_product()

Σύμφωνα με την εκφώνηση της εργασίας χρειάζεται να βρούμε τις τιμές του πίνακα γινομένου C μόνο στις μη μηδενικές θέσεις του πίνακα γειτνίασης A. Επειδή επιπλέον ο πίνακας A είναι συμμετρικός, οπότε και ο πίνακας A^2 θα είναι συμμετρικός, αρκεί να βρούμε τις τιμές του
πίνακα C μόνο στις μη μηδενικές τιμές του κάτω (ή πάνω) τριγωνικού πίνακα γειτνίασης. Έτσι, διατρέχουμε τον κάτω τριγωνικό πίνακα του A και για κάθε στοιχείο (i,j) που παίρνουμε, βρίσκουμε πόσα μη μηδενικά στοιχεία έχουν στα ίδια σημεία οι στήλες i και j. Ο αριθμός αυτών των μη μηδενικών στοιχείων είναι η τιμή του στοιχείου C(i,j). Πλέον έχουμε τον κάτω τριγωνικό πίνακα του C σε μορφή COO και μπορούμε εύκολα να τον μετατρέψουμε σε μορφή CSC όπως περιγράψαμε παραπάνω. Τέλος τυπώνουμε τον αριθμό των τριγώνων (που είναι το άθροισμα των μη μηδενικών στοιχείων του πίνακα C δια του 6) ώστε να ελέγξουμε την ορθότητα των αποτελεσμάτων μας.

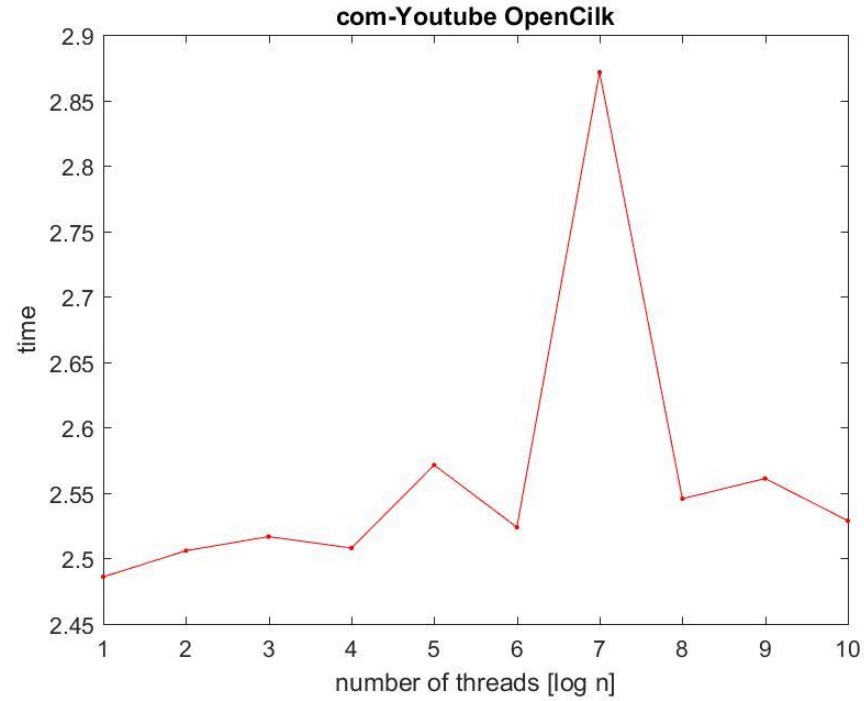
Οι επιδόσεις της σειριακής υλοποίησης ήταν.

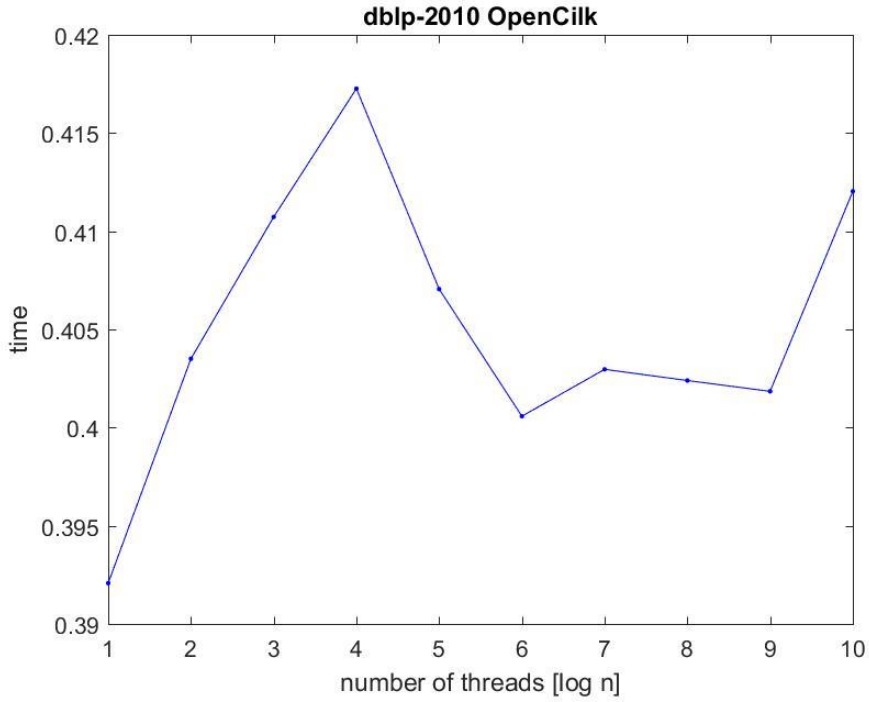
- Com-Youtube: 6.0981s
- Dblp-2010: 0.38416s
- NACA0015: 2.3232s

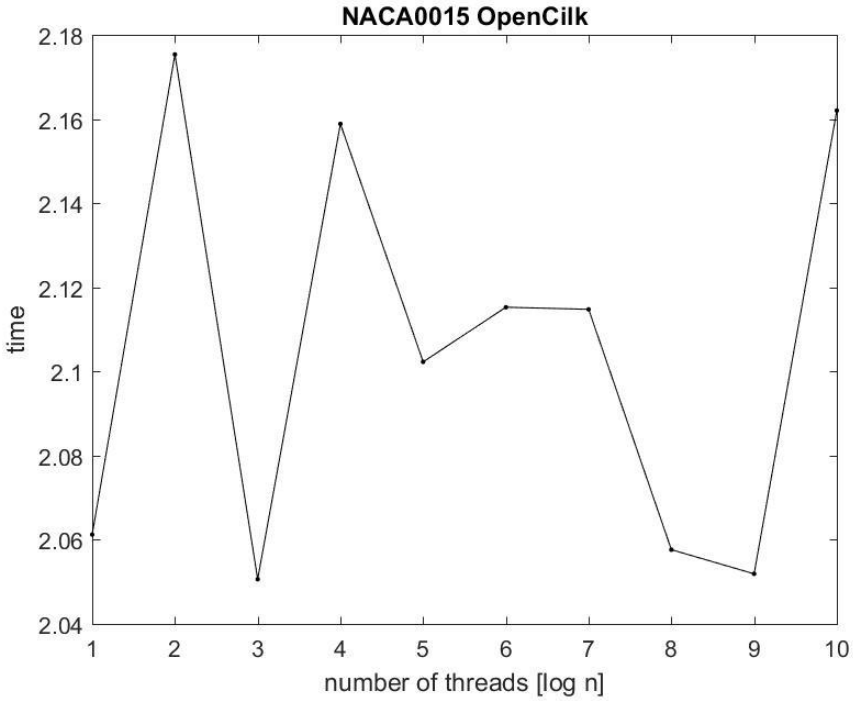
Παράλληλη Υλοποίηση

- ## OpenCilk

Για την παράλληλη υλοποίηση με OpenCilk, απλά αντικαταστήσαμε τις for, που είχαμε για να διατρέχουμε τον κάτω τριγωνικό του πίνακα A, με cilk_for και επιπλέον ορίσαμε μεταβλητό grainsize με βάση τα threads που θα θέλαμε να έχουμε κάθε φορά. Παρακάτω φαίνονται μερικά διαγράμματα με τις επιδώσεις σε χρόνο της υλοποίησής μας συναρτήσει των threads για διάφορα γραφήματα.


com-Youtube OpenCilk
time
number of threads [log n]
2.9
2.85
2.8
2.75
2.7
2.65
2.6
2.55
2.5
2.45
1
2
3
4
5
6
7
8
9
10

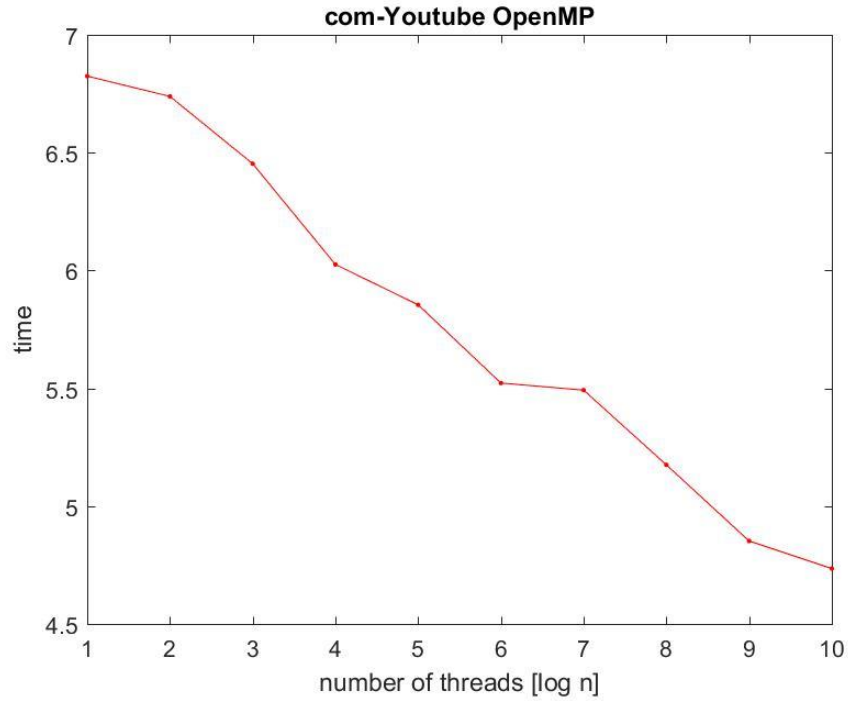

dblp-2010 OpenCilk
time
number of threads [log n]
0.42
0.415
0.41
0.405
0.4
0.395
0.39
1
2
3
4
5
6
7
8
9
10

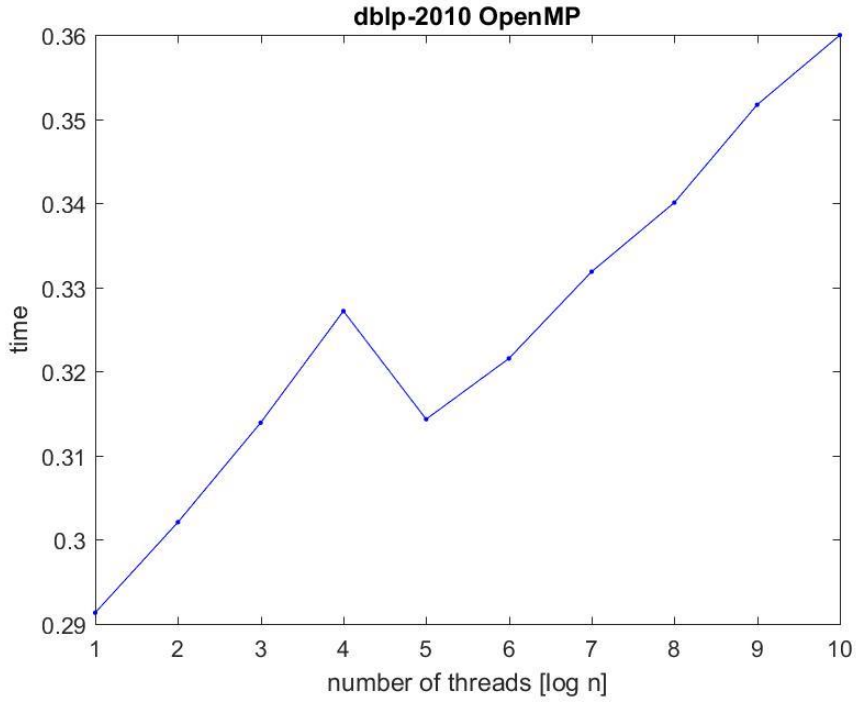


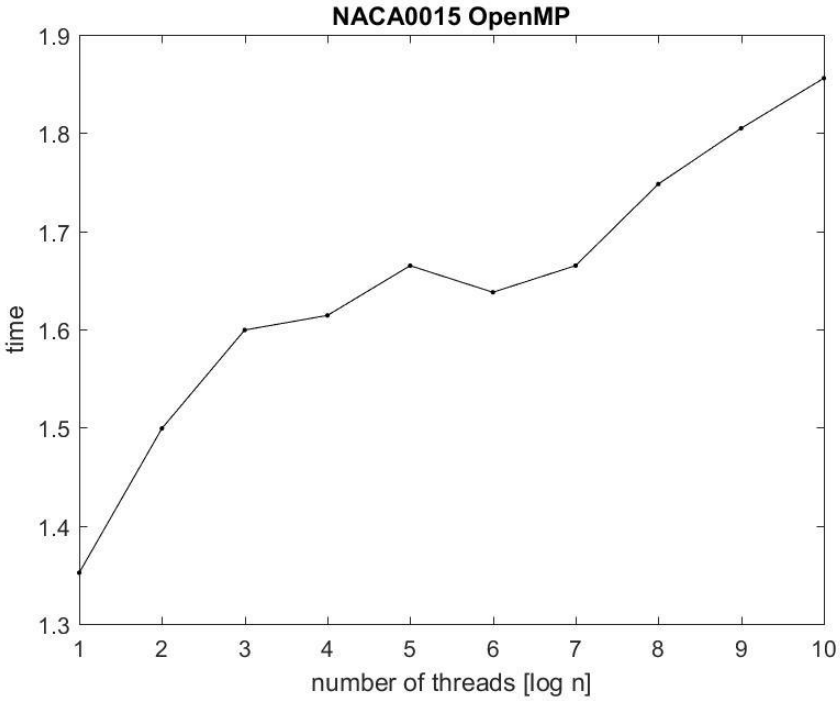
Παρατηρούμε ότι δεν υπάρχει κάποια βελτίωση με βάση τον αριθμό των threads.

- ## OpenMP

Για την OpenMP υλοποίηση αντικαταστήσαμε τις for με #pragma omp parallel for και θέσαμε επίσης omp_set_num_threads(NUM_OF_THREADS) για να ελέγχουμε πόσα threads θα έχουμε κάθε φορά.Παρακάτω έχουμε τα αποτελέσματα.


com-Youtube OpenMP
time
number of threads [log n]
7
6.5
6
5.5
5
4.5
1
2
3
4
5
6
7
8
9
10

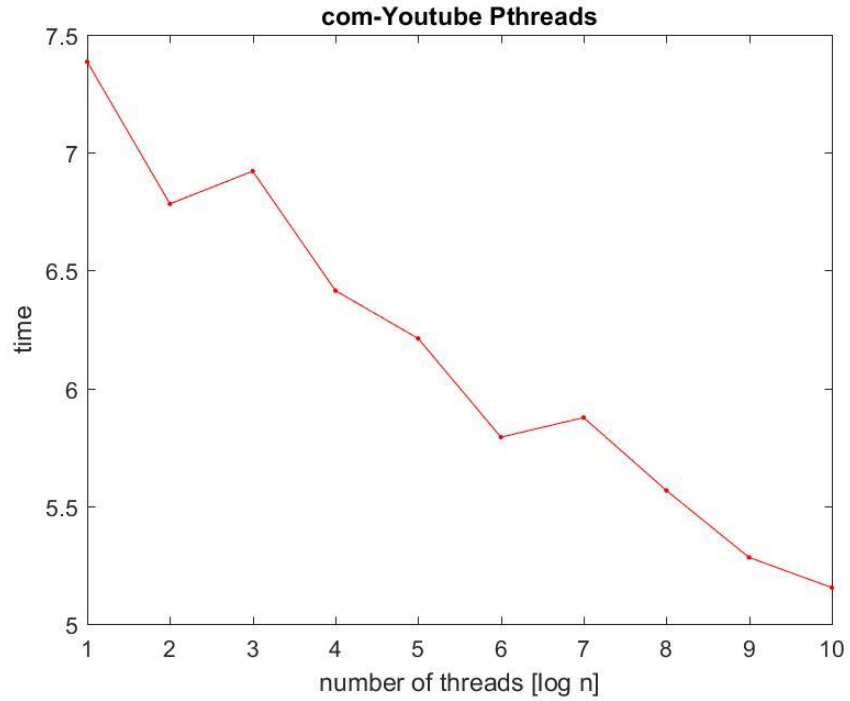

dblp-2010 OpenMP
time
number of threads [log n]
0.36
0.35
0.34
0.33
0.32
0.31
0.3
0.29
1
2
3
4
5
6
7
8
9
10

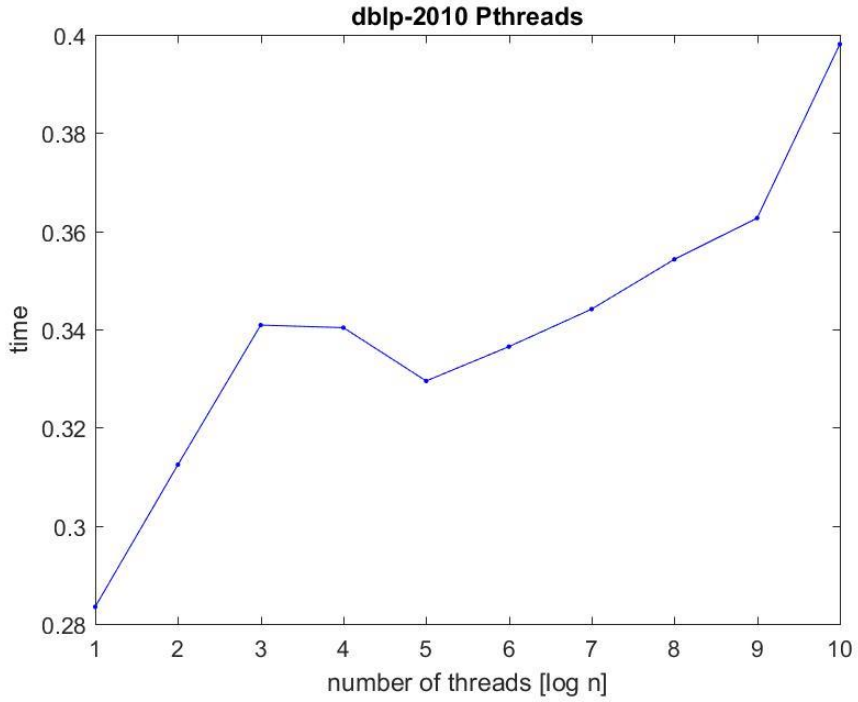


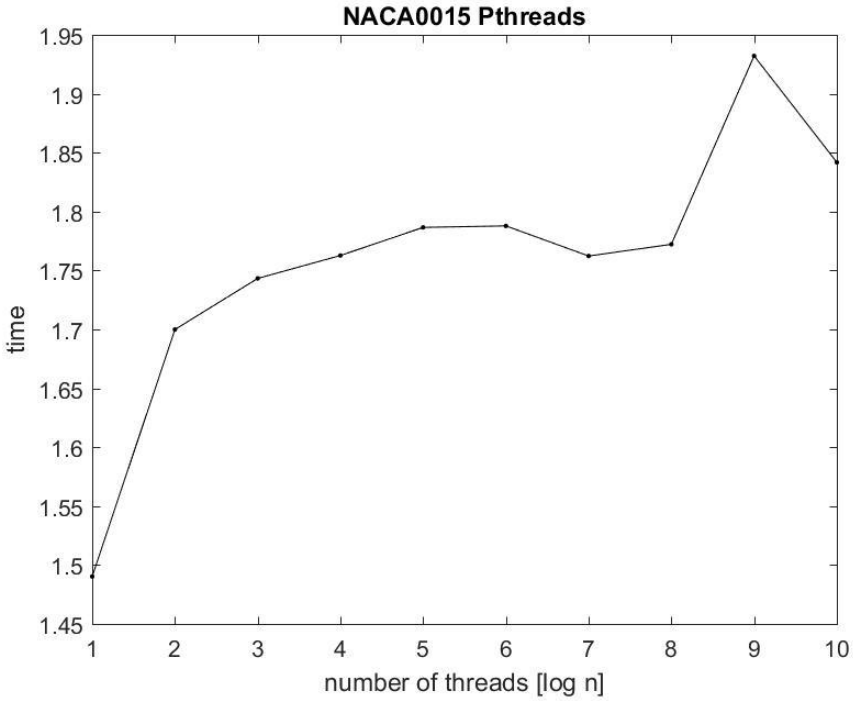
Παρατηρούμε ότι ενώ στο πρώτο διάγραμμα έχουμε απεριόριστη βελτίωση με την αύξηση των threads, στα άλλα δύο εχουμε επιβράδυνση.

- ## Pthreads

Για την υλοποίηση με Pthreads χωρίσαμε τον πίνακα C σε p μέρη και κάθε thread υπολόγιζε τα στοιχεία του C σε N/p στήλες, όπου p ο αριθμός των thread και N ο συνολκός αριθμός στηλών του C. Να σημειωθεί ότι η υλοποίηση στηρίχθηκε πάνω στο παράδειγμα hello.c που βρίσκεται στο docker του μαθήματος. Παρακάτω έχουμε τις επιδόσεις.


com-Youtube Pthreads
time
number of threads [log n]
7.5
7
6.5
6
5.5
5
1
2
3
4
5
6
7
8
9
10


dblp-2010 Pthreads
time
number of threads [log n]
0.4
0.38
0.36
0.34
0.32
0.3
0.28
1
2
3
4
5
6
7
8
9
10



Και εδώ παρατηρούμε την ίδια συμπεριφορά στις επιδόσεις σε σχέση με τον αριθμό των threads όπως και στην υλοποιήση με OpenMP.

Να σημειωθεί πως όλοι οι χρόνοι είναι μετρημένοι σε δευτερόλεπτα, ενώ για την αποφυγή τυχόν data races χρησιοποιήθηκε mutex σε όλες τις παράλληλες υλοποιήσεις. Τέλος, όλη η εργασία έγινε σε διπύρηνο laptop.