Παράλληλα και διανεμημένα συστήματα / Άσκηση 1

Κωτούλας Εμμανουήλ 9697

emmakoto@ece.auth.gr

2021-2022

Άσκηση 1η

Το θέμα της εργασίας είναι η συγγραφή κώδικα για τον υπολογισμό των τριγώνων σε έναν γράφο, δεδομένου ενός αρχείου matrix market που τον περιέχει σε μορφή COO, και έπειτα να τον παραλληλοποιήσουμε με στόχο την βελτίωση της απόδοσης του.

Ο κώδικας μου σε γλώσσα C και τα αρχεία CSV που παρήγαγα και χρησιμοποίησα για να φτάσω στα αποτελέσματα που θα παρουσιάσω παρακάτω, είναι διαθέσιμος εδώ : Git Repository

Να σημειωθεί πως για τα παρακατώ τεστ χρησιμοποιήθηκαν διαφορετικοί κώδικες που τρέχουν τα τεστ μαζικά. OpenMP, Pthread και Σειριακό έγιναν compile με gcc, ενώ OpenCilk με clang.

Περιγραφή Λύσης

- Εισαγωγή αρχείων στον κώδικα :

Για την ανάγνωση των πινάκων χρησιμοποιήθηκε μια τροποποιημένη έκδοση της υλοποίησης που υπάρχει στο read.c example του matrix market και μέθοδοι που υπάρχουν στο mmio.c. Έτσι στην mtxToCoo που υπάρχει σε όλες τις υλοποιήσεις (σειριακή και 3 παράλληλες) διαβάζει τον πίνακα που παρέχεται σαν όρισμα και τον αποθηκεύει σε ενα Struct Coo το οποίο και επιστρέφεται.

- Μετατροπή COO σε CSC :

Για την μετατροπή του πίνακα από COO σε CSC χρησιμοποιήθηκε έτοιμος κώδικας του Δημήτρη Φλώρου παιρνόντας τα αντίστοιχα ορίσματα, έπειτα ο πίνακας αποθηκεύεται σε ένα Struct για ευκολία χρήσης ιδιαιτέρα στην υλοποιήση της Pthreads όπου περνάμε το Struct σαν ορισμα στην pcreate/Triangle counting.

- Υπολογισμός τριγώνων (Triangle counting) :

Για τον υπολογισμό των τριγώνων πρέπει να γίνει το γίνει το γινόμενο A.*(A*A) όπου το υλοποιούμε διατρέχοντας τα μη μηδενικα στοιχεία με 2 for loops, έπειτα με 2 ακόμα for διατρέχουμε τα στοιχεία που βρίσκονται στο ίδιο στο column όπως το current στοιχείο σε ποιντερ g και index j, αυτό μπορούμε να το κάνουμε επειδή ο πίνακας μας είναι συμμετρικός και όλα τα στοιχεία του είναι μονάδες. Έτσι ο πολλαπλασιασμός γίνεται με το να βρεθούνε 2 μη μηδενικά στοιχεία στις κατάλληλες θέσεις, εκεί θα προστεθεί 1 (productValues[j]++). Ο πίνακας C σύμφωνα με την εκφώνηση θα είναι ο πίνακας με column pointers και row indices τα ιδια με τον A και πίνακα τιμών τον productValues[]. Ο τελικός αριθμός των τριγώνων θα βρεθεί αμα προσθέσουμε όλες τις τιμές του πίνακα productValues και διαιρώντας το άθροισμα δια 6.

Υλοποιήσεις

- Σειριακή:

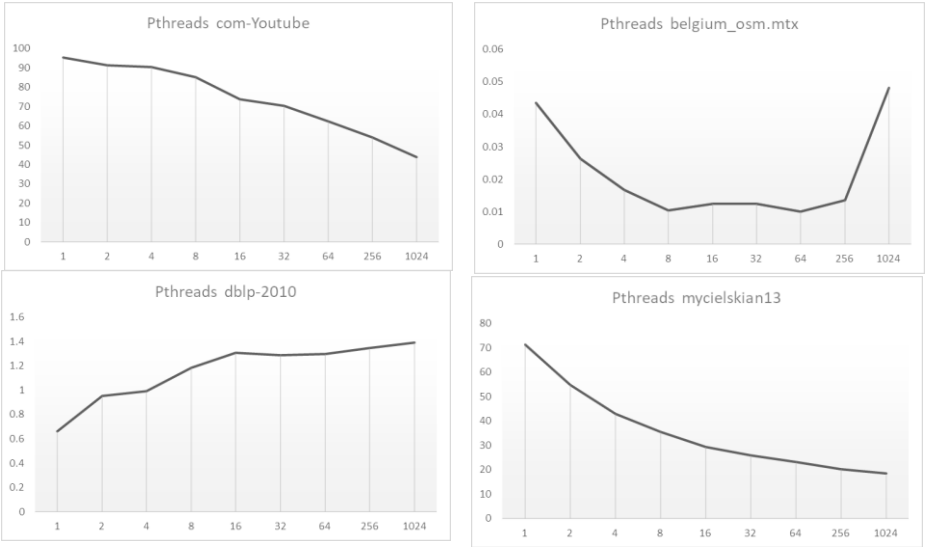
Χρησιμοποιεί τις μεθόδους/συναρτήσεις που αναφέρθηκαν παραπάνω. Δεν υπάρχει κάτι παραπάνω σε αυτήν την υλοποιήση. Παρακάτω παρατείθονται οι χρόνοι για τους 5 πίνακες

που μας δώθηκαν, γίνεται εύκολα φανερό ότι η σειριακή δεν είναι και η πιο γρήγορη υλοποίηση.

Matrix	
belgium_osm.mtx	0.042051
com-Youtube.mtx	127.847
dblp-2010.mtx	0.584671
mycielskian13.mtx	94.84794
NACA0015.mtx	0.649525

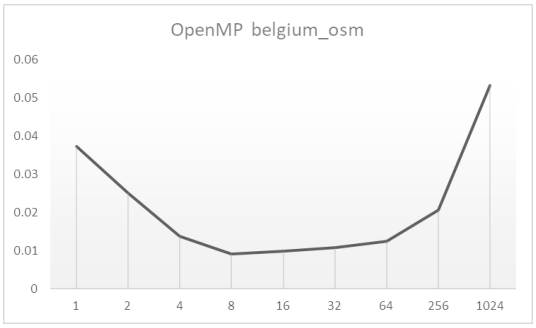
- Pthread :

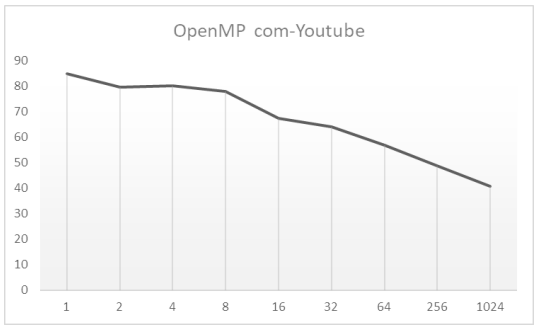
Για την χρήση των μεθόδων της pthread ορίζουμε την triangle counting σαν συνάρτηση που καλείται από την pthread_create όσες φορές όσα threads θέλουμε να χρησιμοποιήσουμε. Έτσι ο πίνακας των pointers χωρίζεται σε όσα ίσα τμήματα όσα και τα thread. Το αποτέλεσμα κάθε κλήσης της συνάρτησης αποθηκεύεται στον ίδιο πίνακα productValues. Παρακάτω παραθέτω 4 από τα 5 benchmark καθώς πιάνουν πολυ χώρο. Σε δύο από τα παραδείγματα φαίνεται οτι συνεχώς πέφτει ο χρόνος καθως αυξάνονται τα threads. Στο Belgium φαίνεται ένα ελάχιστο στα 8 threads. Στο dblp-2010 καθώς και στο NACA0015 ο χρόνος αυξάνεται όσα παραπάνω threads χρησιμοποιήσουμε.



- OpenMP:

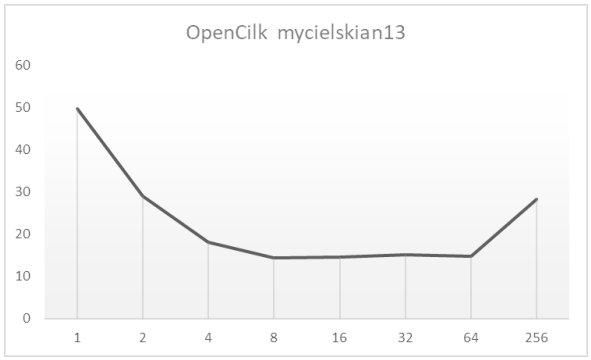
Για την υλοποίηση της χρησιμοποιήθηκε ο κώδικας της σειριακής με την πρόσθεση του της γραμμης 191,194 και 209 όπου ορίζουμε το πλήθος των threads που θελουμε να τρέξουμε και έπειται ορίζουμε την παραλληλοποίηση της for θεωρώντας της μεταβλητές που την διατρέχουν παράλληλες για κάθε thread. Παράλληλη for καλείται και στην 209 για τον υπολογισμό του αθροίσματος. Παρακάτω δίνονται πίνακες για 2 αντιπροσωπευτικά παραδείγματα της συμπεριφοράς του OpenMP. Το Belgium_osm, το dblp-2010 και το NACA0015 παρουσιάζουν σημαντικη βελτίωση απόδοσης μέχρι τα 8 threads και έπειτα μένει στάσιμη η απόδοση ή και πέφτει. Αντίθετα στους πίνακες mycielskian και com-Youtube η απόδοση συνεχώς αυξάνεται με την αύξηση των threads.

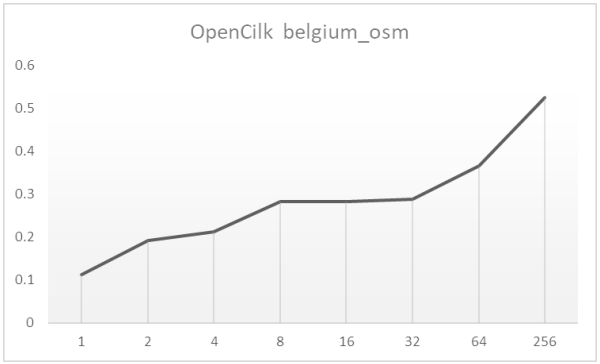




- OpenCilk:

Για την υλοποίηση της OpenCilk χρησιμοποιήθηκε ο κώδικας της serial implementation με την διαφορά στον βρόχο στην γραμμή 193 και 206 όπου χρησιμοποιούμαι την cilk_for για την παραλληλοποιήση του κώδικα μας. Χρησιμοποιήθηκε mutex για το άθροισμα της 208 για την αποφυγή σφάλματος. Σε όλα τα παραδείγματα της OpenCilk εκτός απο το Belgium_osm φαίνεται να υπάρχει ενα ελάχιστο στα 8 ή 4 threads. Δεν γνωρίζω σε τι οφείλεται η επίδοση της Belgium_osm.





Τελική Σύγκριση

Στα περισσότερα τεστ φαίνεται να υπερέχει η OpenCilk και να ειναι πολυ κοντά η OpenMp. Εξαίρεση αποτελεί ο Belgium_osm στον οποίο έχει πολύ χειρότερη απόδοση η OpenCilk. Η σειριακή έχει την χειρότερη απόδοση γενικότερα με εξαίρεση τα δύο τεστ που η Pthread έχει χειρότερη απόδοση. Αυτά τα τεστ έγιναν με 8 Threads καθώς εκεί ειχαν overall την καλύτερη επίδοση.