Παράλληλα και Διανεμημένα Συστήματα
Εργασία 4-Σεπτέμβριος 2018
Μελεζιάδης Ιωάννης ΑΕΜ 8760

Το παρακάτω link περιέχει τον κώδικα, τα εκτελέσιμα,την αναφορά και τα διαγράμματα :
https://drive.google.com/open?id=15hDRGOKYVlXP9R-wBR0YKCiPS8QPhkqB

Σημείωση : Υπάρχουν σχόλια σε όλη την έκταση του κώδικα οπότε ο αναγνώστης καλείται να τον έχει ανοιχτό συγχρόνως με το παρών έγγραφο για ευκολότερη κατανόηση. Επιπλέον αφέθηκαν σκοπίμως κάποιες εντολές(σε σχόλια) που χρησιμοποιήθηκαν για την αποσφαλμάτωση.

1. Δεδομένα

Για τα δεδομένα της άσκησης χρησιμοποιήθηκε το αρχείο hollins.dat[2] .Περιέχει στην πρώτη σειρά τον αριθμό των κόμβων και των συνδέσεων, στη συνέχεια τα ονόματα όλων των ιστοσελίδων και τέλος τις συνδέσεις.

Στην αρχή αποθηκεύω τον αριθμό των κόμβων και των συνδέσεων και στη συνέχεια δημιουργώ έναν πίνακα από strings με τις ονομασίες των ιστοσελίδων(για εκτύπωση μετά την εκτέλεση του Pagerank). Στη συνέχεια δημιουργώ έναν nxn πίνακα S στον οποίο τοποθετείται άσσος όταν η σελίδα (i) δείχνει έχει outlink στη σελίδα (j). Ο πίνακας αυτός ονομάζεται και adjacency matrix και είναι η βάση του αλγορίθμου Pagerank.

2. Αλγόριθμος Pagerank

Για τον αλγόριθμο Pagerank χρησιμοποιήθηκε κυρίως η πηγή [3] σε συνδυασμό με την [4]. Παρακάτω ορίζονται με τη σειρά τα βήματα που έγιναν.

Dangling pages : Αφού έχει δημιουργηθεί ο S(adjacency) πρέπει να γίνει διαχείριση των dangling nodes, δηλαδή των ιστοσελίδων που δεν έχουν outlink σε καμία ιστοσελίδα. Υποθέτουμε λοιπόν ότι όταν ένας τυχαίος χρήστης φτάσει σε μια τέτοια ιστοσελίδα τότε πηγαίνει ισοπίθανα σε οποιαδήποτε ιστοσελίδα του data set. Για αυτό το λόγο δημιουργείται το διάνυσμα outlinks[n] το οποίο αποθηκεύει πόσα outlinks έχει κάθε ιστοσελίδα(δηλαδή πόσους άσσους έχει κάθε γραμμή του πίνακα S). Σύμφωνα με τον παρακάτω τύπο[3] :

$$S(i,j) = \begin{cases} \frac{1}{l_i} & \text{if there is an outlink from } i \text{ to } j \\ 0 & \text{if there is no outlink from } i \text{ to } j \end{cases}$$

όπου l_i είναι το πλήθος των outlinks της i ιστοσελίδας, κάνω στοχαστικό(κατά γραμμές) τον πίνακα S. Στην περίπτωση που έχω dangling page τότε τοποθετώ 1/n σε όλη τη γραμμή του S. Τέλος κάνω αναστροφή του πίνακα S στον St ώστε να είναι στοχαστικός κατά στήλη.

Teleportation : Για να οριστεί πλήρως το πρόβλημα της βέλτιστης ιστοσελίδας, πρέπει να σκεφτούμε ότι ένας χρήστης μπορεί ανά πάσα στιγμή να πληκτρολογήσει την ίδια τη διεύθυνση της ιστοσελίδας που θέλει να πάει χωρίς να χρησιμοποιήσει κάπιο outlink. Έτσι για να συμπεριλάβουμε τη συγκεκριμένη πιθανότητα πρέπει να ορίσουμε τη μεταβλητή d(που ζητείται από τον χρήστη) σύμφωνα με τον παρακάτω τύπο :

$$G = d \cdot St + (1 - d) \cdot T$$

όπου ο St είναι ο στοχαστικός και ο T είναι ένα διάνυσμα όπου όλες οι τιμές του είναι η ισοπίθανη έκβαση 1/n. Ο πίνακας G όμως δεν δημιουργείται σε αυτή τη φάση καθώς σκοπός μου είναι να ορίσω το πρόβλημα Ax = b όπως στο κεφάλαιο 5.1(πηγή [3]).

Definition of Ax = b : Σύμφωνα με τις πηγές ο ισχύουν A = I – dSt και b = (1-d)(1/n) τα οποία και υλοποιούνται στον κώδικα. Μετά από αυτό το βήμα εκτελείται επαναλληπτικά ο αλγόριθμος Gauss-Seidel πάνω στο σύστημα Ax = b και βρίσκει τις λύσεις του.

3. Αλγόριθμος Gauss-Seidel

Στα πλαίσια της εργασίας, υλοποιήθηκε ο αλγόριθμος Pagerank χρησιμοποιώντας τη μέθοδο επίλυσης Gauss-Seidel. Ο αλγόριθμος που επιλέχθηκε για την υλοποίηση του Gauss-Seidel είναι ο παρακάτω[1] :

$$x_i^{(k+1)} = x_i^{(k)} + \frac{1}{A_{ii}} \left(b_i - \sum_{j=0}^{i-1} A_{ij} x_j^{(k+1)} - \sum_{j=i}^{n} A_{ij} x_j^{(k)} \right)$$

Σε κάθε επανάληψη του αλγορίθμου, σε αντίθεση με τη μέθοδο Jacobi,αξιοποιούνται και όσες $x_i^{(k+1)}$ τιμές έχουν υπολογιστεί πριν. Για παράδειγμα για τον υπολογισμό του $x_n^{(k+1)}$ χρησιμοποιούνται οι $x_{n-1}^{(k+1)}$... $x_0^{(k+1)}$ αντί των $x_{n-1}^{(k)}$ … $x_0^{(k)}$. Για τις υπόλοιπες(μεγαλύτερες του i-1) τιμές που είναι άγνωστες για (k+1) χρησιμοποιούνται τα (k). Ακολουθεί η επεξήγηση του σειριακού Gauss-Seidel.

Για την υλοποίηση του παραπάνω αλγορίθμου δημιουργώ δυο πίνακες μεγέθους n, τον pageRank και τον shift. Ο πίνακας pageRank περιέχει σε κάθε επανάληψη τις παλιές τιμές αλλά και όσες έχουν υπολογιστεί μέχρι αυτήν την επανάληψη(δηλαδή αν βρίσκεται στην i-επανάληψη τότε τα στοιχεία του pageRank μέχρι την i-1 μεταβλητή έχουν τις καινούριες τιμές). Ο πίνακας shift υπολογίζει σε κάθε επανάληψη την μετακίνηση που πρέπει να γίνει, σύμφωνα με τον τύπο, ώστε το pageRank να πλησιάζει στη σωστή λύση.

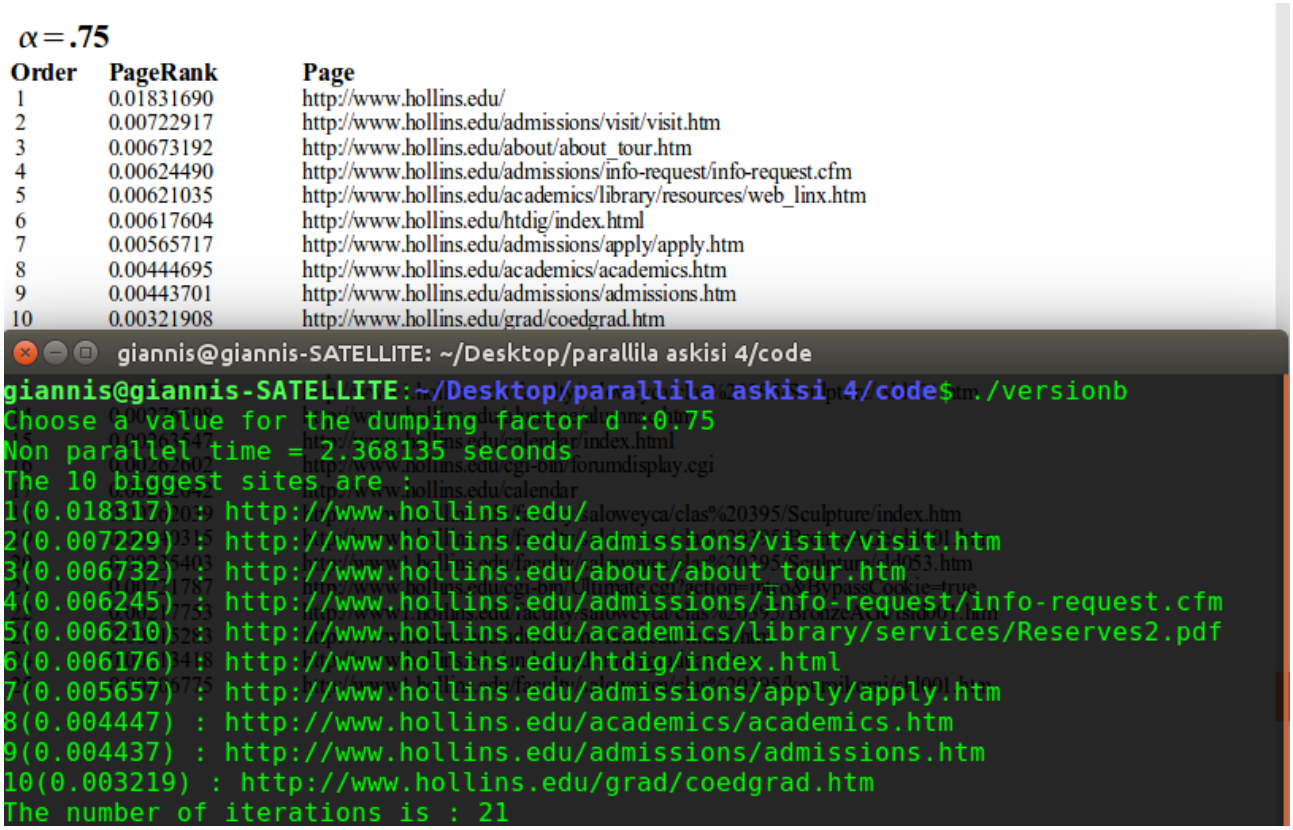
Ακολουθεί ενα while loop όπου μέσα του πραγματοποιείται ο Gauss-Seidel. Σε κάθε επανάληψη του αλγορίθμου υπολογίζεται και το σφάλμα σαν άθροισμα όλων των shifts που έχουν γίνει σε όλα τα σημεία του διανύσματος pageRank. Όταν αυτό το σφάλμα είναι μικρότερο του 0.000001 τότε σημαίνει ότι το διάνυσμα μετακινείται ελάχιστα και άρα η λύση είναι μια βέλτιστη προσέγγιση της πραγματικής και για αυτό το λόγο διακόπτεται ο αλγόριθμος.

Αποχωρόντας από το while loop είναι αποθηκευμένα στον πίνακα pageRank οι τελικές βαθμολογίες για κάθε ιστοσελίδα. Απομένει μόνο η ταξινόμησή τους.

4. Έλεγχος ορθότητας

Μετά την ολοκλήρωση του αλγορίθμου Gauss-Seidel χρησιμοποιώ τον αλγόριθμο Bubblesort για την ταξινόμηση των ιστοσελίδων κατά φθινουσα σειρά. Τέλος τυπώνω τα αποτελέσματα από τον πίνακα websites μαζί με το pageRank της κάθε ιστοσελίδας. Ο έλεγχος των παραγόμενων αποτελεσμάτων έγινε με σύγκριση με το Appendix B της πηγής [4]. Εξακριβώθηκαν και οι τιμές του διανύσματος pageRank αλλά και η θέση των υψηλότερων ιστοσελίδων. Παρατηρήθηκαν

κάποιες διαφορές σε κάποιες ονομασίες των ιστοσελίδων χωρίς όμως να αλλάζει η θέση και η τιμή του pageRank τους. Ακριβώς για αυτόν το λόγο θεωρώ ότι ο αλγόριθμος μου είναι σωστός απλά υπάρχουν μικροδιαφορές στα δυο datasets που χρησιμοποιήθηκαν καθώς προκύπτουν από αληθινά δεδομένα. Ενδεικτικά πάνω είναι η πηγή [4] και κάτω το αποτέλεσμα του κώδικα μου για d = 0.75:



$\alpha = .75$

Order	PageRank	Page
1	0.01831690	http://www.hollins.edu/
2	0.00722917	http://www.hollins.edu/admissions/visit/visit.htm
3	0.00673192	http://www.hollins.edu/about/about_tour.htm
4	0.00624490	http://www.hollins.edu/admissions/info-request/info-request.cfm
5	0.00621035	http://www.hollins.edu/academics/library/resources/web_linx.htm
6	0.00617604	http://www.hollins.edu/htdig/index.html
7	0.00565717	http://www.hollins.edu/admissions/apply/apply.htm
8	0.00444695	http://www.hollins.edu/academics/academics.htm
9	0.00443701	http://www.hollins.edu/admissions/admissions.htm
10	0.00321908	http://www.hollins.edu/grad/coedgrad.htm

```
giannis@giannis-SATELLITE: ~/Desktop/parallila askisi 4/code
giannis@giannis-SATELLITE:~/Desktop/parallila askisi 4/code$ ./versionb
Choose a value for the dumping factor d :0.75
Non parallel time = 2.368135 seconds
The 10 biggest sites are :
1(0.018317) : http://www.hollins.edu/
2(0.007229) : http://www.hollins.edu/admissions/visit/visit.htm
3(0.006732) : http://www.hollins.edu/about/about_tour.htm
4(0.006245) : http://www.hollins.edu/admissions/info-request/info-request.cfm
5(0.006210) : http://www.hollins.edu/academics/library/services/Reserves2.pdf
6(0.006176) : http://www.hollins.edu/htdig/index.html
7(0.005657) : http://www.hollins.edu/admissions/apply/apply.htm
8(0.004447) : http://www.hollins.edu/academics/academics.htm
9(0.004437) : http://www.hollins.edu/admissions/admissions.htm
10(0.003219) : http://www.hollins.edu/grad/coedgrad.htm
The number of iterations is : 21
```

5. Μέθοδος παραλληλισμού

Για την παραλληλοποίηση του κώδικα χρησιμοποιήθηκε το πακέτο openMP. Η λογική είναι ακριβώς η ίδια με το σειριακό κώδικα. Η μόνη διαφορά είναι ότι παραλληλοποιείται το κομμάτι του Gauss-Seidel που όλες οι τιμές είναι γνωστές από πριν, δηλαδή το κομμάτι των δυο αθροισμάτων. Μόλις τελειώσει το παράλληλο κομμάτι πρέπει να ακολουθήσει σειριακό για τον υπολογισμό του pageRank[i] και την πρόσθεση στο σφάλμα και αυτό διότι το pageRank[i] πρέπει να είναι γνωστό πρωτού ξαναγίνουν τα αθροίσματα για το i+1. Για αυτόν το σκοπό χρησιμοποιήθηκαν οι μεταβλητές chunk, start, end καθώς κάθε thread λαμβάνει ένα συγκεκριμένο κομμάτι των υπολογισμών.

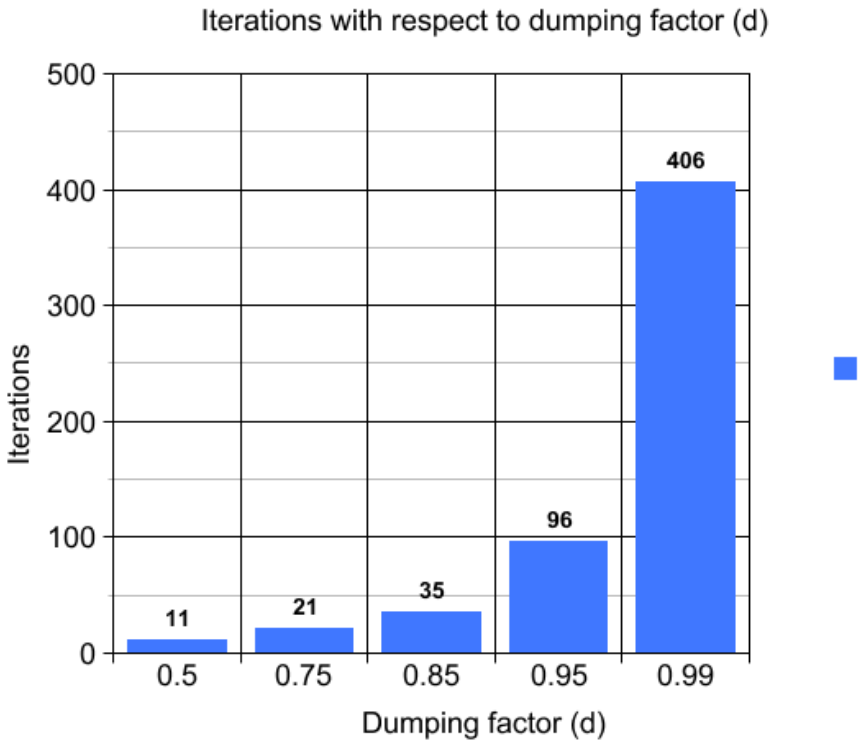
Οι μεταβλητές start και end ορίζουν το κομμάτι που θα παραλάβει κάθε thread. Η μεταβλητή chunk ορίστηκε ως **ceil((float) n /threads)** διότι ο κώδικας πρέπει να λειτουργεί και σε περίπτωση που ο αριθμός των threads που δόθηκε δεν διαιρείται τέλεια με τον αριθμό των συνδέσεων(n).

Όταν δεν υπάρχει τέλεια διαίρεση ο αλγόριθμος πρέπει να κάνει ελέγχους. Συγκεκριμένα, στο παραλληλοποιημένο κομμάτι γίνεται έλεγχος μόνο στο τελευταίο thread για το αν ο αλγόριθμος έχει ξεπεράσει τα όρια του αρχικού πίνακα(δηλαδή αν j >= n). Είναι αναγκαίο να γίνεται μόνο στο τελευταίο thread καθώς διαφορετικά σπαταλάται πολύς χρόνος στους ελέγχους σε threads που δεν υπάρχει περίπτωση να έχουν βγει εκτός ορίων. Η οικονομία αυτή στο χρόνο επιτυγχάνεται με τη χρήση της μεταβλητής flag.

Όταν όμως υπάρχει τέλεια διαίρεση ο κώδικας επιβαρύνεται με παραπάνω ελέγχους(για πρόσβαση εκτός ορίων) που δεν χρειάζονται όπως παρατηρήθηκε κατά τη συγγραφή του κώδικα. Έτσι, για να μην υπάρχει αυτή η επιβάρυνση σε αριθμό threads που διαιρείται τέλεια με το n, δημιουργείται διαφορετικό κομμάτι κώδικα για τέλεια και για μη τέλεια διαίρεση. Το ποιό κομμάτι θα τρέξει ελέγχεται από τη μεταβλητή perfect της οποίας η τιμή είναι :
if(ceil((float)n/threads) == (float)n/threads) perfect = 1; ,όταν υπάρξει τέλεια διαίρεση τοποθετείται 1 στη μεταβλητή perfect αλλιώς παραμένει 0.

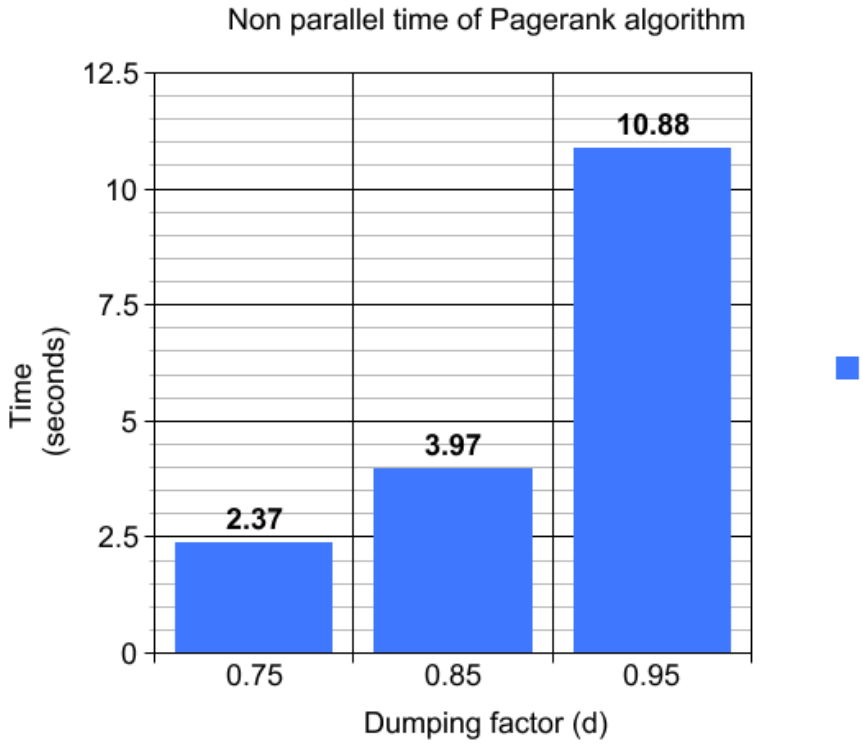
Ιδιαίτερης σημασίας είναι το κομμάτι **#pragma omp critical**. Το critical υποχρεώνει να γίνεται η αφαίρεση από ενα thread κάθε φορά ώστε να μην συμβεί race conditions με αποτέλεσμα να μην έχει το shift[i] τη σωστή τιμή.

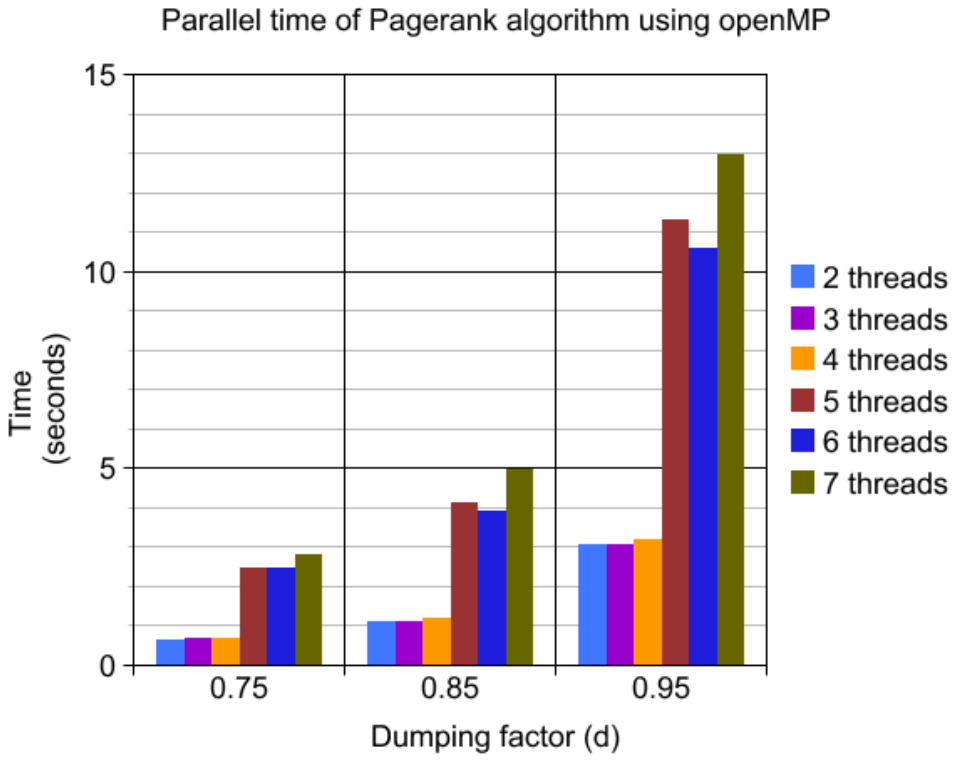
Επιπλέον χρησιμοποιήθηκε **private(threadid,j,start,end,temp3)** καθώς οι μεταβλητές αυτές είναι ξεχωριστές για κάθε νήμα. Δεν ορίστηκαν μέσα στο parallel region καθώς αυτό αυξάνει τον χρόνο εκτέλεσης. Τέλος, αριθμός των βημάτων είναι ίδιος με το σειριακό κώδικα ανεξαρτήτως της παραλληλοποίησης. Όπως φαίνεται και από το διάγραμμα παρακάτω ο αριθμός των βημάτων αυξάνεται πάρα πολύ για d κοντά στη μονάδα ενώ είναι πολύ μικρός για d κοντά στο 0.5 :



6. Σχόλια ταχύτητας σειριακής και παράλληλης υλοποίησης

Τα πειράματα ταχύτητας έγιναν σε υπολογιστή με επεξεργαστή **Intel(R) Core(TM) i5-4200M CPU @ 2.50GHz** με 4 πυρήνες που υποστηρίζει hyperthreading. Παρακάτω ακολουθούν διαγράμματα με τις ταχύτητες των υλοποιήσεων για διαφορετική επιλογή d και threads :





Αναφορικά με τα αποτελέσματα το openMP φαίνεται να επιλέγει αυτόματα το πως θα παραλληλοποιήσει καθώς αρχικά με 2, 3, 4 threads η ταχύτητα ήταν ~0.55 sec. για d=0.75 ενώ μετά από επανεκκίνηση του υπολογιστή οι τιμές είναι αυτές που καταγράφηκαν στα διαγράμματα.

Από τα διαγράμματα φαίνεται να μην υπάρχει διαφορά ανάμεσα σε 2,3 ή 4 threads λόγω της βελτιστοποίησης του openMP. Επιπλέον η διαφορά στην ταχύτητα ανάμεσα στα 5 και 6 threads

οφείλεται στο γεγονός ότι το 5 δεν διαιρεί τέλεια το 6012(αριθμός των συνδέσεων του dataset), έτσι ξοδεύεται χρόνος σε ελέγχους για να μην ξεπεραστούν τα όρια. Τελικά η μέγιστη επιτάχυνση που επιτεύχθηκε είναι 3.64(d = 0.75), 3.6(d = 0.75), 3.58(d = 0.75)

7. Συνθήκες και περιορισμοί

Ένας περιορισμός είναι ότι ο κώδικας είναι προσαρμοσμένος στο συγκεκριμένο σετ δεδομένων και αρα χρειάζεται αλλαγές για να λειτοργήσει σε διαφορετικά. Σαν συνθήκη ορίζεται η τιμή του d καθώς από τη θεωρία πρέπει να ανήκει στο διάστημα (0,1).

Πηγές :

[1] https://www.maa.org/press/periodicals/loci/joma/iterative-methods-for-solving-iaxi-ibi
[2] https://www.limfinity.com/ir/
[3] https://repository.tudelft.nl/islandora/object/uuid:0483fd00-117d-43a3-b61f-6ce8a178e709?collection=education
[4] https://www.limfinity.com/ir/kristen_thesis.pdf
[5] https://www.sciencedirect.com/science/article/pii/S0024379504000023