ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Απεικόνιση γράφων στην κάρτα γραφικών Υλοποιήση των αλγορίθμων Boruvka και Dijkstra

Δημήτριος Γ. Αναστασόπουλος
Αρσάκ Α. Μεγκραμπιάν

Επιβλέπων: Ιωάννης Κοτρώνης, Αναπληρωτής Καθηγητής

ΑΘΗΝΑ
ΣΕΠΤΕΜΒΡΙΟΣ 2016

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Απεικόνιση γράφων στην κάρτα γραφικών
Υλοποιήση των αλγορίθμων Boruvka και Dijkstra

Δημήτριος Γ. Αναστασόπουλος
Α.Μ.:1115201100030

Αρσάκ Α. Μεγκραμπιάν
Α.Μ.:1115201100049

Επιβλέπων: Ιωάννης Κοτρώνης, Αναπληρωτής Καθηγητής

ΠΕΡΙΛΗΨΗ

Στην παρούσα πτυχιακή εργασία παρουσιάζεται μια ενδιαφέρουσα δομή απεικόνισης γράφων στην κάρτα γραφικών και υλοποιούνται οι αλγόριθμοι του Boruvka και Dijkstra σε υψηλό επίπεδο παραλληλοποίησης. Οι υλοποιήσεις αυτές πραγματοποιήθηκαν σε προγραμματιστικό περιβάλλον Unix, με την βοήθεια των CUDA και OpenMP APIs και εκτελέστηκαν για τα οδικά δίκτυα πόλεων και πολιτειών των ΗΠΑ.

Σκοπός της εργασίας αυτής είναι να δούμε αν τελικά η απόδοση που παίρνουμε αξιοποιώντας την κάρτα γραφικών είναι τέτοια ώστε να μπορεί να ανταγωνιστεί την απόδοση του επεξεργαστή και να την ξεπεράσει, παίρνοντας την θέση του ως καλύτερη λύση για την επίλυση προβλημάτων που χρησιμοποιούν τους προαναφερθέντες αλγόριθμους.

Αρχικά, αναπτύσσονται οι βασικές έννοιες του δέντρου επικάλυψης ελαχίστου κόστους και της δομής συμπιεσμένων αραιών γραμμών (CSR). Επίσης, περιγράφεται ο αλγόριθμος του Boruvka, οπου διατυπώνεται ο αλγόριθμος και εξηγούνται όλα τα βήματα του σε φυσική γλώσσα. Στην συνέχεια, περιγράφεται αναλυτικά η υλοποίηση του με CUDA και OpenMP και με ποιο τρόπο χρησιμοποιείται η δομή CSR. Τέλος, παρουσιάζονται οι χρόνοι εκτέλεσης του αλγορίθμου και γίνεται σχολιασμός των αποτελεσμάτων.

Στην συνέχεια, αναλύεται η έννοια του ελάχιστου μονοπατιού, ακολουθούμενη από την περιγραφή του αλγορίθμου του Dijkstra. Όπως και στον αλγόριθμο του Boruvka, έτσι και εδώ αναλύονται λεπτομερώς όλα τα βήματα του αλγορίθμου και περιγράφονται οι υλοποιήσεις του με Cuda και openMP. Ακολουθούν και εδώ οι μετρήσεις των χρόνων εκτέλεσης του αλγορίθμου και ο σχολιασμός των αποτελεσμάτων.

Τέλος, παρουσιάζονται τα συμπεράσματα που προέκυψαν από την παραλληλοποίηση των δύο αλγορίθμων, όπου διαπιστώνεται ότι, στην συντριπτική πλειοψηφία των εκτελέσεων τους, οι επιταχύνσεις που επιτυγχάνονται με την κάρτα γραφικών ξεπερνάνε άλλοτε σημαντικά (αλγόριθμος Dijkstra) και άλλοτε κατά πολύ (αλγόριθμος Boruvka) αυτές του επεξεργαστή.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Θεωρία γράφων
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Compressed Sparse Row (CSR), boruvka, dijkstra, δέντρο επικάλυψης, ελάχιστα μονοπάτια

ABSTRACT

In this thesis, we present parallel implementations of Boruvka and Dijkstra algorithms along with the use of the Compressed Sparse Row (CSR) format to display our input graphs in the main or graphics memory. Our implementations are based on Unix operating systems and are using the CUDA library of NVIDIA as well as the OpenMP library. We test our applications on graphs that represent road maps of cities and states of the USA.

The purpose of this thesis is to determine whether the speed ups we gain in execution time using our GPU are enough to surpass our CPU execution times for both the algorithms used.

We start by explaining the concept of minimum spanning trees and the CSR format. After that, we describe in depth both the Boruvka algorithm and our implementations in CUDA and OpenMP, emphasizing the necessity of the CSR format. We then show our executions times using figures followed by our comments on them.

Afterwards we analyze the concept of shortest path. Similar to Boruvka algorithm, we describe step by step the Dijkstra algorithm and we present our implementations in CUDA and OpenMP. Measurements and execution times are also being displayed using figures followed by our comments on them.

Finally, we summarize the results we gathered from parallelizing the two algorithms in order to write our conclusions. In the majority of our tests we observed significant speed ups of Boruvka algorithm running on CUDA against multithreaded executions on CPU. However, the speed ups we gained in Dijkstra algorithm were not that impressive but were promising enough for further investigation.

SUBJECT AREA: Graph theory
KEYWORDS: Compressed Sparse Row (CSR), boruvka, dijkstra, spanning tree, shortest paths

Η παρούσα εργασία είναι αφιερωμένη στις οικογένειες μας.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

1 Εισαγωγή

Οι μονάδες επεξεργασίας γραφικών (GPU) ή απλούστερα, όπως είναι γνωστές, κάρτες γραφικών, έχουν στις μέρες μας πολλές εφαρμογές διαφορετικού τύπου, πέρα από την βασική τους που είναι η απεικόνιση δισδιάστατων (2D) ή τρισδιάστατων (3D) γραφικών. Οι μοντέρνες κάρτες γραφικών προσφέρουν υψηλή υπολογιστική ισχύς σε χαμηλό κόστος έναντι των κεντρικών μονάδων επεξεργασίας (CPU) ή αλλιώς τους επεξεργαστές των ηλεκτρονικών υπολογιστών, κυρίως εξαιτίας της αρχιτεκτονικής τους που περιλαμβάνει ένα κατά πολύ μεγαλύτερο πλήθος πυρήνων. Τα προηγούμενα, σε συνδυασμό με την ανάπτυξη πολλών διεπαφών προγραμματισμού εφαρμογών (API), όπως είναι το CUDA[1] που έχει αναπτυχθεί από την γνωστή εταιρία παραγωγής καρτών γραφικών Nvidia και το OpenCL[2] καθιστούν τις κάρτες γραφικών μια προσιτή λύση για υπολογιστικά ακριβές διαδικασίες.

Οι εφαρμογές που ευνοούνται περισσότερο από την αρχιτεκτονική των καρτών γραφικών είναι αυτές οι οποίες αποτελούνται από δεδομένα που είναι ομοιόμορφα κατανεμημένα στην μνήμη (regular data structures). Ένα παράδειγμα προβλήματος με τέτοιου είδους δεδομένα αποτελεί το άθροισμα των στοιχείων ενός πίνακα. Τα δεδομένα αυτά βρίσκονται σε συνεχόμενες θέσεις στην μνήμη και μπορούν να προσπελαστούν άμεσα και ανεξάρτητα το ένα από το άλλο ώστε να γίνουν οι επιθυμητοί υπολογισμοί.

Τι συμβαίνει όμως με εφαρμογές οι οποίες εκτελούν αλγορίθμους που βασίζονται σε δεδομένα τα οποία δεν είναι κατανεμημένα στην μνήμη με ομοιόμορφο τρόπο (irregular data algorithms[3]); Αν και οι κάρτες γραφικών δεν έχουν σχεδιαστεί για να αντιμετωπίζουν τέτοιου είδους εφαρμογές, η απάντηση στο ερώτημα αυτό αποτελεί μια σπουδαία πρόκληση για τους προγραμματιστές παράλληλων αλγορίθμων. Συγκεκριμένα, η πρόκληση έγκειται κυρίως στους παρακάτω λόγους.

- Οι υπολογισμοί σε δεδομένα τέτοιου είδους είναι δύσκολο να γίνουν παράλληλα εξαιτίας της ακανόνιστης κατανομής τους στην μνήμη.
- Η απόδοση των υπολογισμών αυτών βασίζεται σημαντικά στον τρόπο απεικόνισης των δεδομένων στην μνήμη.
- Η επιλογή της κατάλληλης απεικόνισης συνδέεται με βαθιά γνώση του προβλήματος η οποία μπορεί να είναι διαθέσιμη μόνο κατά τον χρόνο εκτέλεσης του αλγορίθμου.

Στην παρούσα εργασία θα ασχοληθούμε με την απεικόνιση δομών γράφων στην μνήμη της κάρτας γραφικών καθώς και θα δούμε πως μπορούμε να πετύχουμε σημαντική επιτάχυνση στην απόδοση διάφορων αλγορίθμων όταν αυτοί εκτελούνται στην κάρτα γραφικών, σε σύγκριση με εκτελέσεις που πραγματοποιούνται σε επεξεργαστές με τα δεδομένα να αποθηκεύονται στην κύρια μνήμη (RAM). Το πρόβλημα που εξετάζουμε ανήκει φυσικά στην κατηγορία εφαρμογών με δεδομένα κατανεμημένα με τρόπο ανομοιόμορφο στην μνήμη, αφού η διάταξη των δεδομένων σε ένα γράφο είναι από την φύση της ακανόνιστη και απαιτεί κατάλληλα διαμορφωμένες δομές για την αποδοτική απεικόνιση της στην μνήμη.

Οι υλοποιήσεις που θα παρουσιαστούν στην παρούσα εργασία έχουν πραγματοποιηθεί στο CUDA API αλλά και με την βιβλιοθήκη openMP[4] για την καταμέτρηση των χρόνων των εκτελέσεων τόσο στην κάρτα γραφικών όσο και στον επεξεργαστή.

Οι υλοποιήσεις που έχουν γίνει στο CUDA API έχουν γραφτεί στην γλώσσα C και απαιτούν μια κάρτα γραφικών της Nvidia για την εκτέλεση τους. Στις δικές μας μετρήσεις χρησιμοποιήσαμε μια Nvidia GeForce GTX 970 που διαθέτει 4GB μνήμης γραφικών (VRAM) τύπου GDDR5 και 1664 πυρήνες (cuda cores), ενώ βασίζεται στην επονομαζόμενη αρχιτεκτονική «Maxwell» της εταιρείας. Από την άλλη μεριά, οι υλοποιήσεις που έχουν γίνει με την βοήθεια της βιβλιοθήκης openMP είναι επίσης γραμμένες σε C και μπορούν να εκτελεστούν για οποιοδήποτε πλήθος νημάτων, ώστε να καλύπτουν τις αρχιτεκτονικές των περισσότερων σύγχρονων επεξεργαστών. Οι δικές μας μετρήσεις πραγματοποιήθηκαν με ένα Intel Core i5 3570K που διαθέτει 4 φυσικούς πυρήνες οι οποίοι είναι χρονισμένοι στα 3.8GHz.

Τα αρχεία πάνω στα οποία πραγματοποιήσαμε τις μετρήσεις μας αφορούν οδικά δίκτυα πόλεων και πολιτειών των ΗΠΑ που είναι απεικονισμένα σε μορφή γράφου. Στον παρακάτω πίνακα μπορούμε να διακρίνουμε τα ονόματα των αρχείων αυτών καθώς και το πλήθος των κόμβων και των κορυφών των γράφων που αντιπροσωπεύουν. Στα διαγράμματα που θα ακολουθήσουν θα αναφερόμαστε σε αυτά με το όνομα τους.

Όνομα	Περιγραφή	Κόμβοι	Ακμές
NY	New York City	264.346	733.846
BAY	San Francisco Bay	321.270	800.172
COL	Colorado	435.666	1.057.066
FLA	Florida	1.070.376	2.712.798
NW	Northwest USA	1.207.945	2.840.208
NE	Northeast USA	1.524.453	3.897.636
CAL	California and Nevada	1.890.815	4.657.742
LKS	Great Lakes	2.758.119	6.885.658
E	Eastern USA	3.598.623	8.778.114
W	Western USA	6.262.104	15.248.146
CTR	Central USA	14.081.816	34.292.496
USA	Full USA	23.947.347	58.333.344

Πίνακας 1: Τα αρχεία εισόδου

2 Ο αλγόριθμος Boruvka

2.1 Δέντρα επικάλυψης ελαχίστου κόστους

Στη Θεωρία Γράφων και στην Επιστήμη Υπολογιστών, συχνά συναντάται το πρόβλημα της εύρεσης του δέντρου επικάλυψης ελαχίστου κόστους (MST)[5] ενός γράφου με βάρη στις ακμές. Το πρόβλημα συνίσταται στην εύρεση ενός δέντρου με κορυφές αυτές του γράφου και ακμές ένα υποσύνολο των ακμών του γράφου, τέτοιο ώστε το άθροισμα των βαρών τους να είναι το ελάχιστο δυνατό. Αλλιώς, το πρόβλημα μπορεί να διατυπωθεί ως πρόβλημα για την εύρεση ενός υποσυνόλου των ακμών του γραφήματος, ώστε το νέο υπογράφημα που προκύπτει να είναι συνεκτικό και το άθροισμα των βαρών των ακμών του να είναι το ελάχιστο δυνατό. Εδώ, δεν είναι σαφές αν η λύση του προβλήματος αποτελεί δέντρο. Όμως, αν το υπογράφημα που προκύπτει ως λύση έχει κύκλο, τότε αφαιρώντας μια ακμή από τον κύκλο (την βαρύτερη) το υπογράφημα παραμένει συνεκτικό και το άθροισμα των βαρών των ακμών του είναι ακόμη μικρότερο (αφού αφαιρέσαμε την βαρύτερη ακμή). Επομένως, δεν γίνεται η λύση του προβλήματος να περιέχει κύκλο, γιατί τότε δεν θα έχει το ελάχιστο δυνατό άθροισμα βαρών. Και αφού επιπλέον το υπογράφημα πρέπει να είναι συνεκτικό, εξορισμού προκύπτει ότι η λύση θα είναι ένα δέντρο. Για την επίλυση τέτοιων προβλημάτων έχουν αναπτυχθεί πολλοί αποτελεσματικοί άπληστοι αλγόριθμοι. Ένας από αυτούς, τον οποίο θα εξετάσουμε, είναι ο αλγόριθμος του Boruvka.

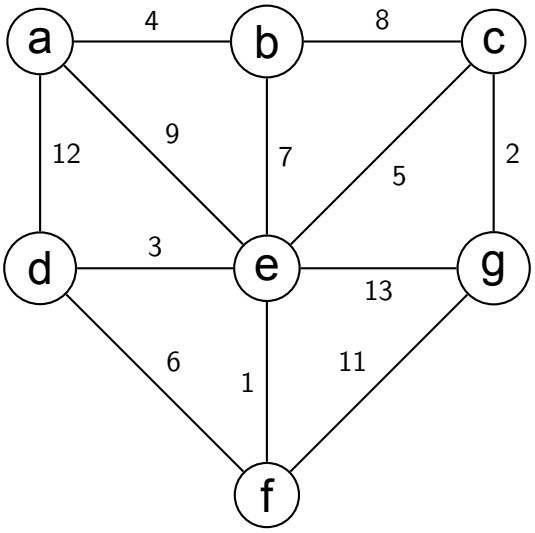
2.2 Η δομή συμπιεσμένων αραιών γραμμών (CSR)

Για την αποτελεσματικότερη παραλληλοποίηση του αλγορίθμου βασιστήκαμε στον σχεδιασμό compressed sparse row (CSR)[6]. Με την επιλογή αυτή ενισχύουμε την απόδοση του, διότι η ειδική τοποθέτηση των δεδομένων βελτιώνει την χρήση και της CPU και της GPU. Συγκεκριμένα είναι μια πολύ αποτελεσματική προσέγγιση για την συρρίκνωση του γράφου (συγχώνευση των κόμβων σε υπερκόμβους). Η δομή CSR είναι ένας συμβιβασμός μεταξύ μιας λίστας γειτνίασης και ενός πίνακα γειτνίασης και με αυτήν αναπαριστούμε τον γράφο στην κάρτα γραφικών. Σε αυτή την δομή ο γράφος αναπαρίσταται με τέσσερεις πίνακες:

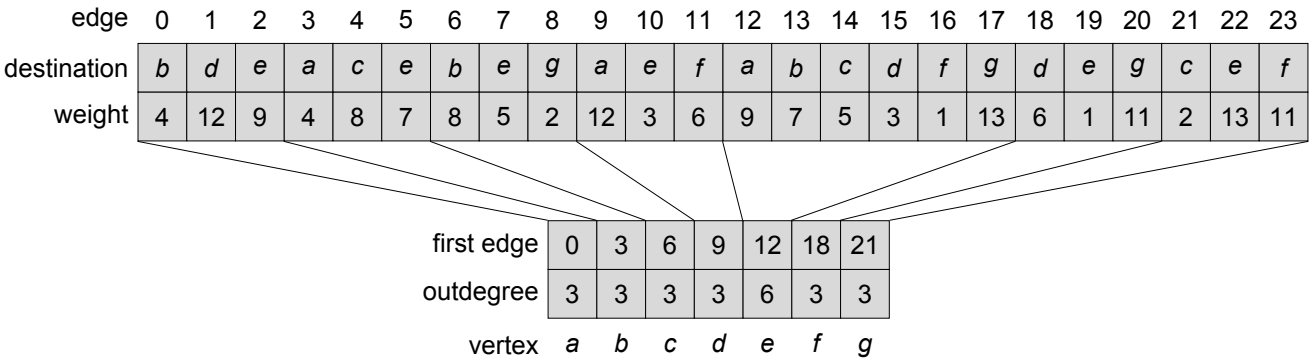
- **Destination** – πίνακας μεγέθους $|E|$, ο οποίος αντιστοιχίζει κάθε ακμή στον προορισμό της.
- **Weight** – πίνακας μεγέθους $|E|$, ο οποίος αντιστοιχίζει κάθε ακμή με το βάρος της.
- **First edge** – πίνακας μεγέθους $|V|$, ο οποίος αντιστοιχίζει κάθε κορυφή με την πρώτη της ακμή.
- **Outdegree** – πίνακας μεγέθους $|V|$, ο οποίος αντιστοιχίζει κάθε κορυφή με τον συνολικό αριθμό των εξερχόμενων ακμών της.

Για να αναπαραστήσουμε μη κατευθυνόμενους γράφους, κάθε ακμή εμφανίζεται δύο φορές ώστε να καλυφθούν και οι δυο κατευθύνσεις.

Επιπλέον, αυτή η αναπαράσταση του γράφου είναι ανεξάρτητη από την πλατφόρμα στην οποία αναπτύσσεται, δηλαδή, μπορεί να εφαρμοστεί και σε CPU και σε GPU chips ακόμα και σε κατανεμημένα συστήματα χωρίς καμία τροποποίηση.



Σχήμα 1: Μη κατευθυνόμενος γράφος



Σχήμα 2: Αναπαράσταση της δομής CSR

Στα παραπάνω σχήματα βλέπουμε την αναπαράσταση CSR ενός μη κατευθυνόμενου γράφου που περιλαμβάνει 7 κορυφές και 24 ακμές.

2.3 Περιγραφή αλγορίθμου

Στα τέλη της δεκαετίας του ’90 η επιστημονική κοινότητα είχε επικεντρωθεί στις προσπάθειες παραλληλοποίησης άπληστων αλγορίθμων παραγωγής δέντρων επικάλυψης ελαχίστου κόστους. Ο αλγόριθμος του Kruskal[7] σε αυτή την περίπτωση ήταν ο λιγότερο ελκυστικός, λόγο της ακολουθιακής μορφής του. Ο αλγόριθμος του Prim[8] ήταν αρκετά κατάλληλος για παραλληλοποίηση, αλλά οι υπερβολικά πολύπλοκες διαδικασίες υπολογισμού του, που απαιτούσαν συγχρονισμό μεταξύ τους, μείωναν την επιτάχυνση του. Ο αλγόριθμος του Boruvka[9], από την άλλη πλευρά, ήταν από την φύση του παράλληλος, με αποτέλεσμα να γίνει ο ισχυρότερος υποψήφιος για παραλληλοποίηση.

Ο αλγόριθμος αυτός δημοσιεύτηκε για πρώτη φορά το 1926 από τον Otakar Boruvka[10] ως μέθοδος κατασκευής ενός αποτελεσματικού δικτύου ηλεκτρικής ενέργειας για τη Μοραβία. Σε κάθε του βήμα παράγει έναν ενδιάμεσο γράφο ο οποίος περιλαμβάνει υπερκόμβους του αρχικού γράφου με ένα υποσύνολο των ακμών του. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να παραχθεί ένα δέντρο επικάλυψης όλων των κορυφών που περιέχει τις ακμές με το ελάχιστο βάρος.

Αλγόριθμος 1:

Είσοδος: Μη κατευθυνόμενος γράφος $G(V, E)$

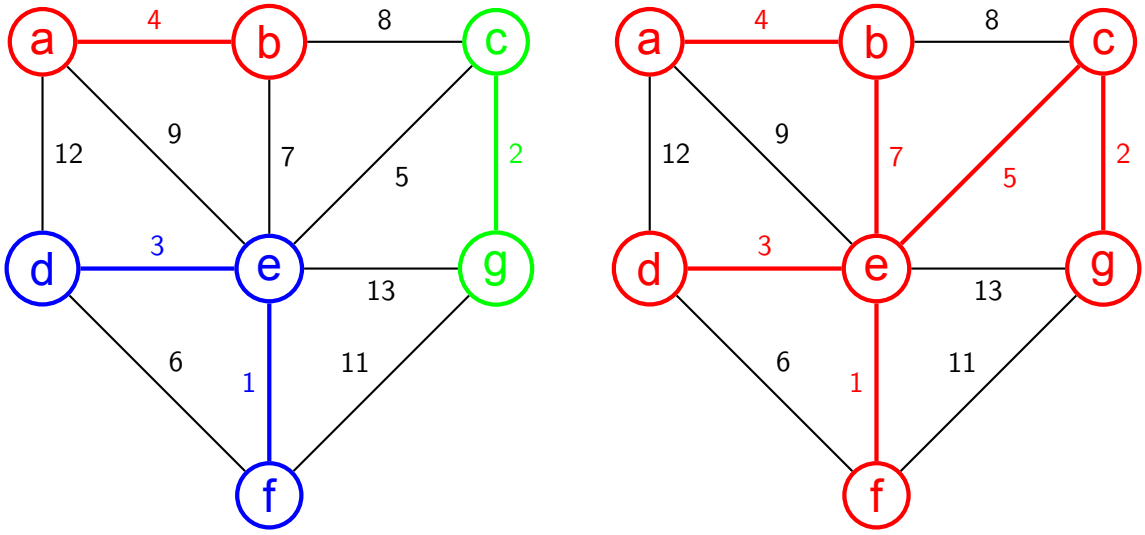
1. **Όσο** ο αριθμός των κόμβων > 1 **επανέλαβε**
2. Εύρεση ελάχιστης ακμής ανά κόμβο
3. Αφαίρεση διπλών ακμών
4. Ανάθεση χρώματος
5. **Όσο** δεν έχει συγκλίνει **επανέλαβε**
6. Διάδοση χρώματος
7. Δημιουργία νέου id κορυφής
8. Μέτρηση νέων ακμών
9. Εκχώρηση ακμών στις κορυφές
10. Εισαγωγή νέων ακμών

Για τον αλγόριθμο μας έχουμε αναπτύξει μια σειρά από συναρτήσεις που εφαρμόζονται σε κάθε κορυφή ξεχωριστά, κάτι που εξασφαλίζει την έυκολη παραλληλοποίηση του αλγορίθμου. Αυτές οι συναρτήσεις τρέχουν μέχρις ότου να παραμείνει ένας υπερκόμβος:

1. **Εύρεση ελάχιστης ακμής ανά κόμβο:** ο αλγόριθμος βρίσκει την ακμή με το ελάχιστο βάρος για κάθε κόμβο. Αν ο κόμβος έχει πολλαπλές ακμές με το ίδιο ελάχιστο βάρος, διαλέγεται η ακμή η οποία έχει το μικρότερο id κορυφής προορισμού. Το id της ακμής που επιλέγεται αποθηκεύεται στο πίνακα minedge.
2. **Αφαίρεση διπλών ακμών:** το διπλότυπο αφαιρείται αν ο προορισμός ενός κόμβου Α έχει ως προορισμό τον ίδιο κόμβο Α. Κατά τον εντοπισμό διπλής ακμής διατηρείται στον πίνακα minedge αυτή που έχει το μεγαλύτερο id κορυφής προορισμού και αφαιρείται η άλλη.

3. **Ανάθεση και διάδοση χρώματος :** προκειμένου να υπάρχει επικοινωνία στο γράφο πρέπει να εντοπισθούν τα συνδεδεμένα στοιχεία του. Τα συνδεδεμένα στοιχεία θα είναι οι υπέρκόμβοι. Για αυτό το σκοπό κάθε κόμβος αρχικοποιείται με το ίδιο χρώμα του κόμβου προορισμού του. Αν ο κόμβος δεν έχει κόμβο προορισμού επειδή έχει διαγραφτεί η ακμή στο δεύτερο βήμα, τότε ως κόμβος προορισμού θεωρείται ο εαυτός του. Η διαδικασία διάδοσης του χρώματος συνεχίζεται μέχρις ότου να χρωματιστούν όλες οι κορυφές.
4. **Δημιουργία νέου id κορυφής:** μετά την σύγκλιση των χρωμάτων, κάθε κορυφή που ο προορισμός της είναι ο εαυτός της μαρκάρεται με 1 στο πίνακα flag. Όλες οι υπόλοιπες κορυφές μαρκάρονται με 0. Στην συνέχεια, υπολογίζοντας το exclusive prefix sum του πίνακα flag παράγουμε τα καινούρια id για τις νέες κορυφές. Με την χρήση του υπολογισμού του prefix sum διασφαλίζεται ότι τα καινούρια id των κόμβων είναι σε αύξουσα σειρά. Ως εκ τούτου, οποιαδήποτε σχέση μεταξύ των κόμβων υπήρχε εξαρχής, διατηρείται.
5. **Μέτρηση, εκχώρηση και εισαγωγή νέων ακμών:** για να κατασκευάσουμε το νέο γράφο μετράμε τον αριθμό των ακμών για κάθε υπερκόμβο, συμπληρώνοντας έτσι τον πίνακα out_degree του νέου γράφου και στην συνέχεια εισάγουμε τις νέες ακμές με τα βάρη, συμπληρώνοντας και τους πίνακες destination και weight αντίστοιχα.

Ο αλγόριθμος του Boruvka χρειάζεται $O(logV)$ επαναλήψεις του εξωτερικού βρόχου μέχρι να τερματίσει και ως εκ τούτου θα εκτελεσθεί σε χρόνο $O(ElogV)$, όπου E είναι ο αριθμός των ακμών, και V είναι ο αριθμός των κορυφών του γράφου $G(V, E)$.



Πρώτη επανάληψη Δεύτερη επανάληψη

Σχήμα 3: Εκτέλεση του αλγορίθμου Boruvka

Στο παραπάνω σχήμα παρουσιάζουμε την εκτέλεση του αλγορίθμου Boruvka για τον γράφο του Σχήματος 1, η οποία τερματίζει μετά από 2 επαναλήψεις.

2.4 Υλοποίηση με Cuda

Η υλοποίηση μας βασίστηκε στην δημοσίευση των C. da Silva Sousa, A. Mariano και A. Proenca[11]. Για την αναπαράσταση του γράφου έχουμε το struct Graph που περιέχει τους τέσσερεις πίνακες destination, weight, first_edge, out_degree, καθώς και τις μεταβλητές nodes και edges, που αντιπροσωπεύουν το πλήθος των κόμβων και των ακμών.

- Για την δημιουργία του γράφου καλείται η συνάρτηση create_graph(), η οποία αρχικοποιεί τους πίνακες και τις μεταβλητές, με βάση το αρχείο εισόδου που δίνεται από τον χρήστη.
- Στην συνέχεια αντιγράφονται όλα τα δεδομένα από την RAM στην VRAM.
- Αμέσως μετά μπαίνουμε στην κεντρική επανάληψη του αλγορίθμου. Ορίζουμε το block και το grid με βάση το BLOCK_SIZE και τον αριθμό των κορυφών.
- Ακολουθεί το πρώτο βήμα του αλγορίθμου. Με την συνάρτηση find_min() βρίσκουμε τις ακμές με το ελάχιστο βάρος για κάθε κορυφή.
- Με την συνάρτηση mirrors_edge() αφαιρούμε τα διπλότυπα.
- Με την συνάρτηση initialize_colors() αναθέτουμε τα χρώματα. Συνεχίζουμε με την συνάρτηση propagate_colors(), η οποία καλείται σε μια επανάληψη και διαδίδει το χρώμα. Η επανάληψη σταματάει όταν δεν υπάρχουν πλέον αλλαγές χρωμάτων.
- Με την συνάρτηση create_new_vertex_ids() δημιουργούμε τον πίνακα flag. Στην συνέχεια, με την χρήση της συνάρτησης cub::DeviceScan::ExclusiveSum(), της βιβλιοθήκης CUB[12], που εφαρμόζεται στον πίνακα flag δημιουργούμε τα καινούρια id. Ελέγχουμε αν έχει μείνει μόνο ένας υπερκόμβος, αν ναι, τότε διακόπτεται η επανάληψη, απελευθερώνεται η μνήμη και ο αλγόριθμος τερματίζει.
- Αλλιώς, συνεχίζεται με την συνάρτηση count_edges(), όπου μετράμε τις καινούριες ακμές για κάθε κορυφή με βάση το χρώμα που έχουν πάρει στο βήμα τρία, δημιουργώντας έτσι τον πίνακα out_degree του νέου γράφου.
- Το Exclusive Prefix Sum του out_degree μας δίνει τον πίνακα first_edge για το νέο γράφο.
- Τελειώνουμε με την εισαγωγή των νέων ακμών στο καινούριο γράφο με την συνάρτηση insert_new_edges(), συμπληρώνοντας τους πίνακες destination και weight.

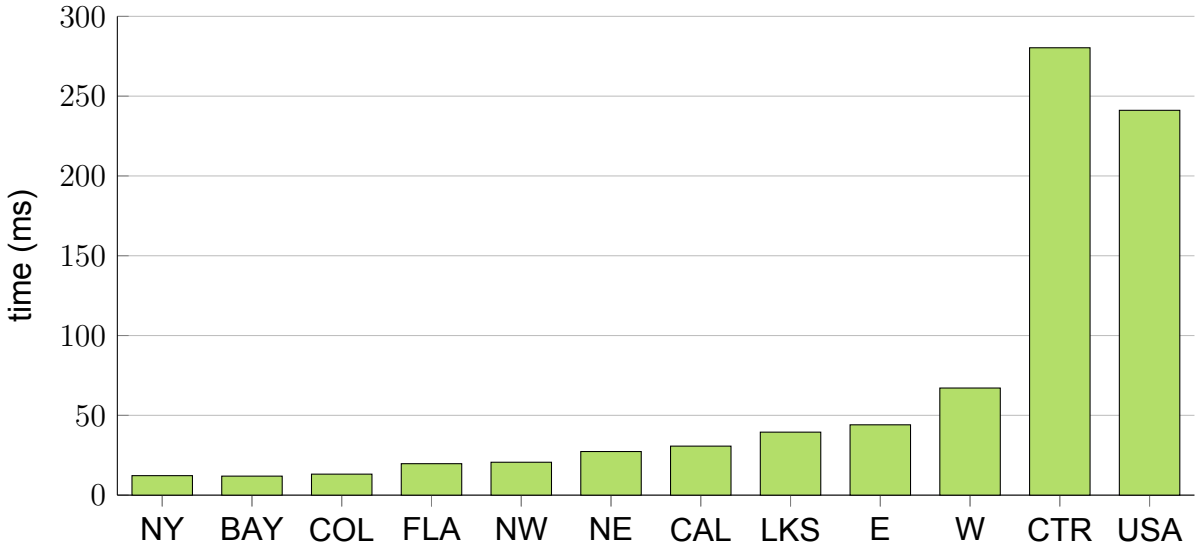
Στην υλοποίηση μας δεν κρατάμε τους ενδιάμεσους γράφους, αλλά μόνο τον τελικό. Στο τέλος κάθε επανάληψης, αφού έχει δημιουργηθεί ο νέος γράφος, διαγράφουμε τα δεδομένα του παλιού γράφου και η επόμενη επανάληψη τρέχει με το νέο.

2.5 Υλοποίηση με openMP

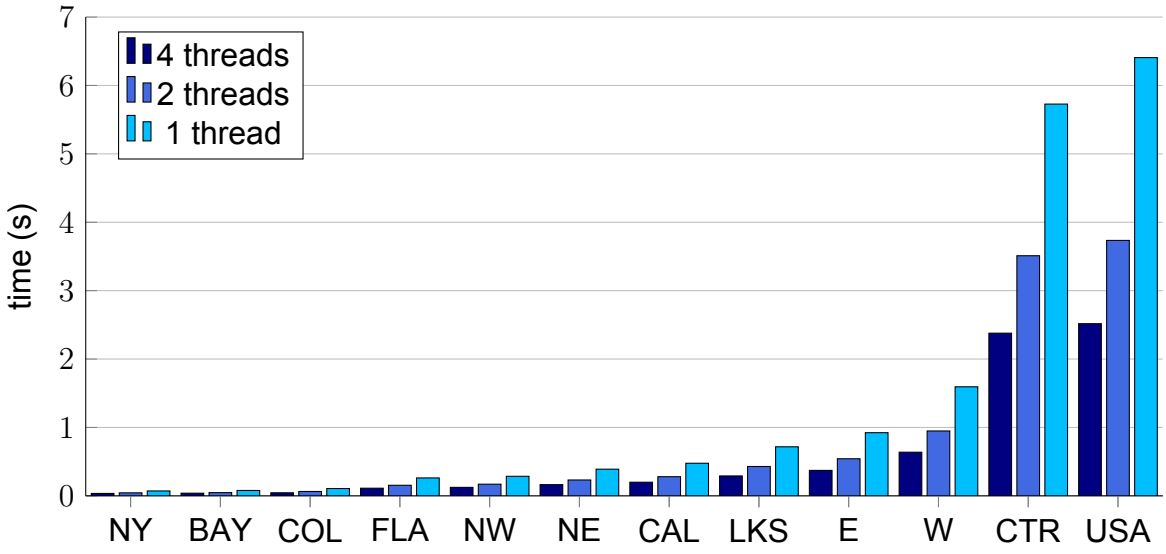
Για την παρουσίαση των αποτελεσμάτων σε σύγκριση με την CPU έχει γίνει μετατροπή του κώδικα ώστε να τρέχει και στον επεξεργαστή. Η παραλληλοποιήση έγινε με την βιβλιοθήκη openMP με την εντολή *pragma omp for*. Όλες οι δομές και οι συναρτήσεις έχουν παραμείνει ίδιες με αυτές της υλοποίησης σε CUDA. Για τον υπολογισμό του exclusive prefix sum δεν χρησιμοποιούμε την βιβλιοθήκη CUB, αλλά έχουμε φτιάξει την συνάρτηση prefix_sum()[13] που υπολογίζει το exclusive prefix sum του πίνακα που παίρνει σαν παράμετρο. Τέλος, οι συναρτήσεις initialize_colors() και propagate_colors() έχουν ενσωματωθεί στην main συνάρτηση του προγράμματος.

2.6 Μετρήσεις

Οι χρόνοι εκτέλεσης που παρουσιάζονται δεν περιλαμβάνουν το χρόνο που χρειάζεται το πρόγραμμα για το διάβασμα του αρχείου εισόδου και την δημιουργία της δομής CSR. Επίσης, οι χρόνοι εκτέλεσης στην GPU περιλαμβάνουν τον χρόνο για την μεταφορά της δομής CSR από την RAM στην VRAM.



Σχήμα 4: Χρόνοι εκτέλεσης Boruvka στην κάρτα γραφικών



Σχήμα 5: Χρόνοι εκτέλεσης Boruvka στον επεξεργαστή

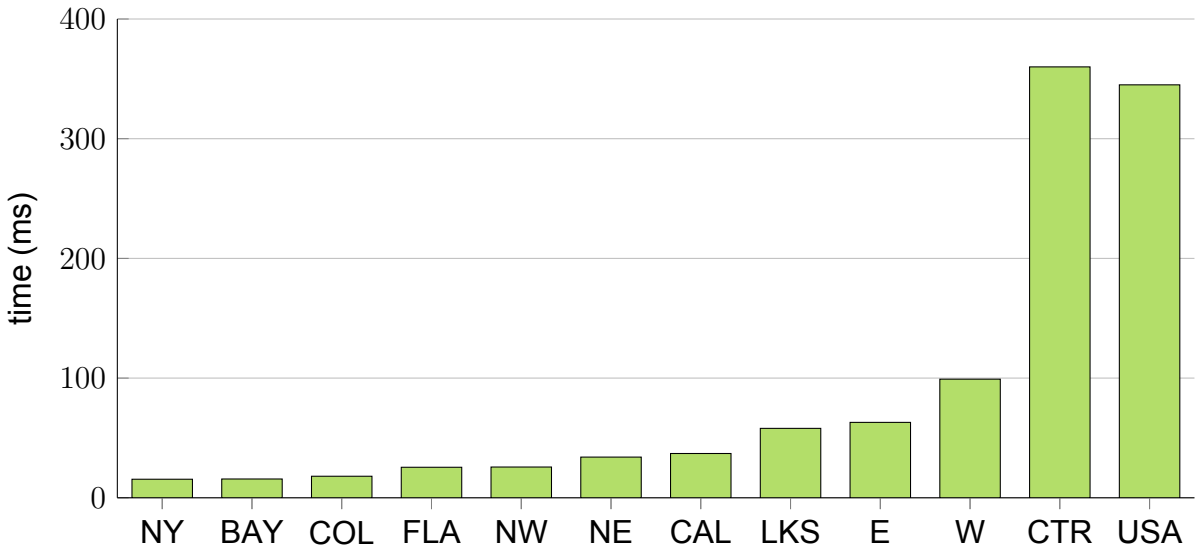
Στην υλοποίηση μας ακολουθήσαμε την τοπολογική προσέγγιση, έτσι κάθε νήμα της GPU επεξεργάζεται μια κορυφή ενώ στην CPU κάθε νήμα επεξεργάζεται ένα σύνολο κορυφών. Ο προγραμματισμός έγινε σε επίπεδο κορυφών και όχι σε επίπεδο ακμών. Σε αντίθετη περίπτωση θα είχαμε ανισόρροπη φόρτωση των νημάτων από κορυφές με πολλές ακμές και αυτό με την σειρά του θα οδηγούσε στην αύξηση του χρόνου εκτέλεσης.

Ο παρακάτω πίνακας συνοψίζει την επιτάχυνση (speedup) και την αποδοτικότητα (efficiency) των εκτελέσεων μας στην CPU και την GPU μας, σε σύγκριση με την σειριακή εκτέλεση του προγράμματος μας στην CPU για τους γράφους BAY, NE, W και USA. Η επιλογή των γράφων έγινε με βάση τα μεγέθη τους, ώστε να καλυφθεί όλη η γκάμα των μεγεθών.

	BAY		NE		W		USA	
Threads	S	E	S	E	S	E	S	E
2	1.67x	83%	1.68x	84%	1.68x	84%	1.71x	85%
4	2.03x	51%	2.38x	60%	2.49x	62%	2.54x	63%
GPU	6.59x		14.27x		23.75x		26.57x	

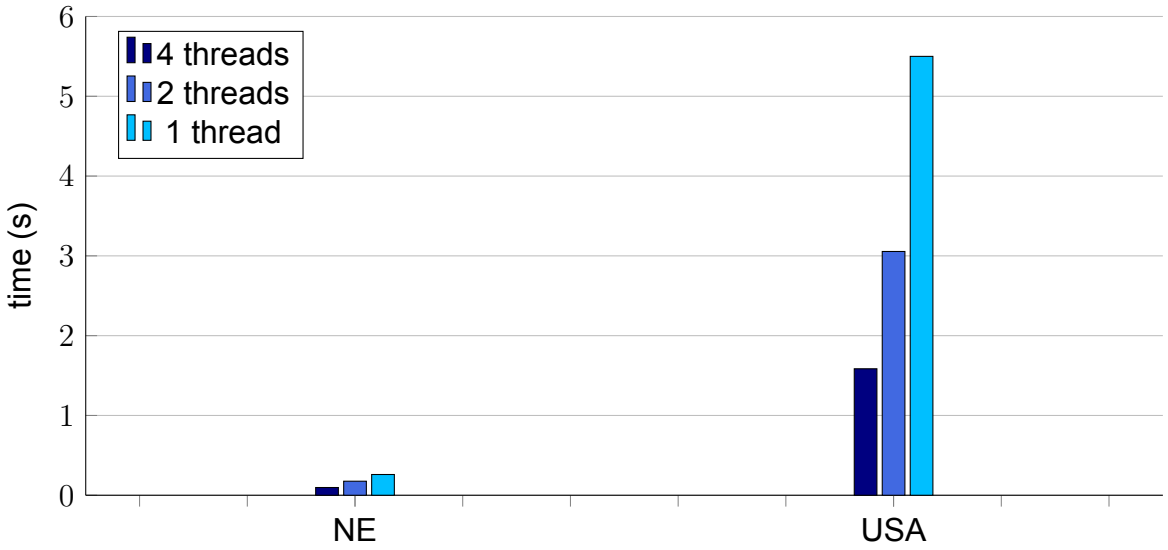
Πίνακας 2: Επιτάχυνση (S) και αποδοτικότητα (E) για 4 γράφους

Από τα Σχήματα 4 και 5 μπορούμε να διαπιστώσουμε ότι η εκτέλεση στην GPU υπερτερεί από αυτήν της CPU. Μπορεί η διαφορά στους μικρούς γράφους να μην είναι φανερή, διότι οι χρόνοι είναι πολύ κάτω από 1 δευτερόλεπτο, αλλά καθώς αυξάνετε το μέγεθος των γράφων διακρίνουμε μεγάλη απόκλιση στους χρόνους. Αυτό φαίνεται καλύτερα στο Πίνακα 2, όπου για το γράφο BAY (κόμβοι < 1 εκατομμύριο) η εκτέλεση στην GPU δεν παρουσιάζει μεγάλη επιτάχυνση, ενώ για τους επόμενους και συνάμα μεγαλύτερους γράφους παρατηρούμε ότι υπάρχει αύξηση της επιτάχυνσης, η οποία συμβαδίζει με την αύξηση του πλήθους των κορυφών του γράφου.



Σχήμα 6: Χρόνοι εκτέλεσης Boruvka στην κάρτα γραφικών NVIDIA K20m

Στα Σχήματα 6 και 7 παρουσιάζονται οι μετρήσεις απο την υλοποίηση της δημοσίευσης των C. da Silva Sousa, A. Mariano και A. Proenca στην κάρτα γραφικών και τον επεξεργαστή αντίστοιχα. Για τις μετρήσεις τους χρησιμοποιήσανε την κάρτα γραφικών NVIDIA K20m που διαθέτει 5GB μνήμης γραφικών τύπου GDDR5, 2496 πυρήνες cuda και βασίζεται στην αρχιτεκτονική «Kepler» και τον επεξεργαστή Intel E5-2670 v2 ο οποίος διαθέτει 10 πυρήνες χρονισμένους στα 2.5GHz.



Σχήμα 7: Χρόνοι εκτέλεσης Boruvka στον επεξεργαστή Intel E5-2670 v2

	NE		USA	
Threads	S	E	S	E
2	1.47x	74%	1.80x	90%
4	2.67x	67%	3.47x	87%
GPU	7.32x		15.85x	

Πίνακας 3: Επιτάχυνση (S) και αποδοτικότητα (E) για 2 γράφους των Σχημάτων 6 και 7

Συγκρίνοντας τις μετρήσεις μας με τις αντίστοιχες της προαναφερθείσας δημοσίευσης παρατηρούμε ότι οι χρόνοι εκτέλεσης στην κάρτα γραφικών μας είναι λίγο μικρότεροι. Πιστεύουμε ότι αυτό οφείλεται κατά κύριο λόγο στην διαφορά δυναμικότητας μεταξύ των δύο καρτών γραφικών που χρησιμοποιούνται, καθώς υπάρχει μια διαφορά περίπου δύο χρόνων στην κυκλοφορία τους στην αγορά και όχι εξαιτίας κάποιας αξιοσημείωτης διαφοράς στις δύο υλοποιήσεις. Αντιθέτως, οι χρόνοι εκτέλεσης στον επεξεργαστή της εν λόγο δημοσίευσης είναι ελαφρώς μικρότεροι από τους δικούς μας. Αυτό κατά πάσα πιθανότητα οφείλεται στον υπολογισμό του exclusive prefix sum ο οποίος γίνεται με την βιβλιοθήκη Intel TBB στην υλοποίηση της δημοσίευσης ενώ εμείς έχουμε αναπτύξει δικό μας κώδικα για τον υπολογισμό αυτό.

3 Ο αλγόριθμος Dijkstra

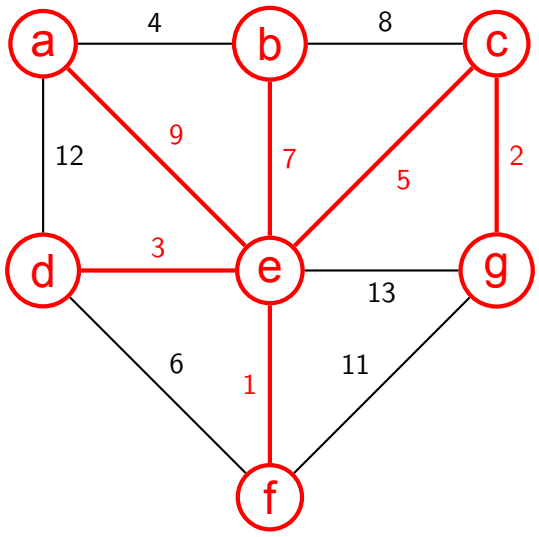
3.1 Ελάχιστα μονοπάτια

Το πρόβλημα του συντομότερου μονοπατιού (shortest path problem) είναι ένα από τα παλαιότερα διατυπωμένα και πιο σημαντικά προβλήματα στην Θεωρία των γράφων. Παρουσιάζει πληθώρα εφαρμογών στην Επιχειρησιακή έρευνα, τους αλγορίθμους δρομολόγησης ενός δικτύου, του Διαδικτύου κ.α. ενώ εμφανίζεται συχνά ως υποπρόβλημα σε πολλά άλλα ενδιαφέροντα προβλήματα.

Μας δίνεται ένα γράφος $G(V, E)$ με V αριθμημένους κόμβους και E ακμές. Κάθε ακμή (i, j) που ανήκει στον G περιέχει το κόστος μετάβασης από τον κόμβο i στον κόμβο j. Το μήκος ενός μονοπατιού αποτελείται από το μήκος των επιμέρους ακμών που το αποτελούν. Το μονοπάτι αυτό χαρακτηρίζεται ως το συντομότερο εάν έχει το μικρότερο μήκος από όλες τις εναλλακτικές διαδρομές με τους ίδιους κόμβους αφετηρίας και προορισμού. Το μήκος του συντομότερου μονοπατιού ονομάζεται επίσης και συντομότερη απόσταση.

3.2 Περιγραφή αλγορίθμου

Ο αλγόριθμος του Dijkstra[14] προτάθηκε το 1954 από τον E.W. Dijkstra[15] και αποτελεί έναν από τους πλέον δημοφιλής και ευρέως χρησιμοποιούμενους αλγορίθμους εύρεσης ελαχίστων μονοπατιών. Στηρίζεται στην παρατήρηση της βέλτιστα διασπώμενης δομής του δέντρου ελάχιστων μονοπατιών, ενώ επειδή σε κάθε του βήμα οριστικοποιεί ένα κόμβο (ή αλλιώς μια ετικέτα του κόμβου) ανήκει και στην γενικότερη κατηγορία οριστικοποίησης ετικετών.



Σχήμα 8: Εκτέλεση του αλγορίθμου Dijkstra με κόμβο αφετηρίας τον e

Ο αλγόριθμος ξεκινώντας από τον κόμβο αφετηρία σε κάθε του βήμα επιλέγει να οριστικοποιήσει τον κόμβο που απέχει την μικρότερη απόσταση από αυτόν. Η διαδικασία αυτή επαναλαμβάνεται έως ότου οριστικοποιήσουμε τον κόμβο προορισμού που έχουμε επιλέξει εκ των προτέρων ή εναλλακτικά μέχρι να οριστικοποιηθούν όλοι οι κόμβοι του γράφου. Στην πρώτη περίπτωση βρίσκουμε το ελάχιστο μονοπάτι από τον κόμβο αφετηρίας προς τον κόμβο προορισμού, ενώ στην δεύτερη, βρίσκουμε τα ελάχιστα μονοπάτια από τον κόμβο αφετηρία προς όλους τους κόμβους του γράφου. Αν δεν υπάρχει κανένα μονοπάτι μεταξύ δύο κόμβων του γράφου η μεταξύ τους απόσταση θεωρείται άπειρη.

Αλγόριθμος 2:

Είσοδος: Μη κατευθυνόμενος γράφος $G(V, E)$, κόμβος αφετηρίας B

1. Αρχικοποίησε όλους τους κόμβους ως μη οριστικοποιήμενους
2. Όρισε τα κόστη προς κάθε κόμβο ίσα με το άπειρο
3. Οριστικοποίησε τον B με κόστος μηδέν και φύλαξε τον στην μεταβλητή C
4. **Όσο** υπάρχει μη οριστικοποιημένος κόμβος **επανέλαβε**
5. **Για κάθε** εξερχόμενη ακμή $e(c, i)$ του C
6. Ανανέωσε το κόστος πρός τον κόμβο i κρατώντας την μικρότερη τιμή
7. **Για κάθε** μη οριστικοποιημένο κόμβο
8. Βρες αυτόν με το ελάχιστο κόστος m
9. Φύλαξε τον στην μεταβλητή C
10. **Αν** το m είναι ίσο με το άπειρο
11. **Σταμάτα** την επανάληψη

Ο βρόγχος της επανάληψης θα εκτελεστεί περίπου n φορές, άρα η πολυπλοκότητα του είναι $O(n)$, όπου n το πλήθος των κόμβων του γράφου. Στο εσωτερικό του βρόγχου η ανανέωση για τα κόστη προς τους κόμβους με τους οποίους συνδέεται άμεσα ο εκάστοτε C κόμβος θεωρείται αμελητέας πολυπλοκότητας, εφόσον ένας κόμβος συνδέεται άμεσα με ένα εκθετικά μικρότερο υποσύνολο των κόμβων όλου του γράφου. Η διαδικασία εύρεσης μη οριστικοποιημένου κόμβου με ελάχιστο κόστος θα χρειαστεί να γίνει για όλους τους κόμβους του γράφου. Ως εκ τούτου είναι πολυπλοκότητας επίσης $O(n)$. Επομένως, η πολυπλοκότητα του αλγορίθμου είναι $O(n^2)$.

3.3 Υλοποίηση με Cuda

Για την υλοποίηση του αλγορίθμου του Dijkstra στο CUDA API ακολουθήσαμε την ίδια δομή απεικόνισης του γράφου με αυτή που χρησιμοποιήσαμε για τον αλγόριθμο του Boruvka. Σε συνδυασμό με την CSR απεικόνιση του γράφου δημιουργήσαμε και μια δομή για την αποθήκευση των δεδομένων της εκτέλεσης του Dijkstra. Η δομή αυτή, που την ονομάζουμε Dijkstra στην υλοποίηση μας, αποτελείται από ένα πίνακα μεγέθους όσο είναι το πλήθος των κόμβων του γράφου. Σε κάθε του θέση ο πίνακας αυτός περιλαμβάνει ένα αναγνωριστικό id για τον κόμβο, μια μεταβλητή flag που υποδηλώνει αν ο κόμβος αυτός έχει οριστικοποιηθεί ή όχι (με τιμές 0 ή 1), το συνολικό κόστος με το οποίο φτάνουμε σε αυτόν από τον κόμβο αφετηρία και τέλος, το id του προηγούμενου κόμβου στο μονοπάτι ελαχίστου κόστους που δημιουργείται καθώς εκτελείται ο αλγόριθμος.

Παρακάτω θα περιγράψουμε συνοπτικά τον κώδικα που έχουμε γράψει.

- Αρχικά αποθηκεύεται όλος ο γράφος στην δομή CSR στην κύρια μνήμη RAM.
- Έπειτα αντιγράφεται η δομή CSR στην μνήμη της GPU και δημιουργείται σε αυτήν και η δομή Dijkstra.
- Ακολούθως εκτελείται ο init kernel ο οποίος, για όλους τους κόμβους του γράφου, αρχικοποιεί τις τιμές της δομής Dijkstra βάζοντας 0 στην τιμή flag (μη οριστικοποιημένος κόμβος), κόστος ίσο με UINT_ MAX (το οποίο θεωρούμε ίσο με το άπειρο μιας και η έννοια της απειρίας δεν μπορεί να οριστεί σε γλώσσα μηχανής), ενώ το id του προηγούμενου κόμβου μένει μη ορισμένο. Εξαίρεση στα προηγούμενα αποτελεί ο κόμβος αφετηρίας ο οποίος θεωρείται οριστικοποιημένος κατά την έναρξη του αλγορίθμου με κόστος επίσκεψης (στον εαυτό του) 0, έχοντας σαν προηγούμενο κόμβο τον εαυτό του.
- Στην συνέχεια, μπαίνοντας στην κύρια δομή επανάληψης της υλοποίησης, καλείται ο kernel update για όλους τους κόμβους με τους οποίους συνδέεται άμεσα ο κόμβος που οριστικοποιείται στο εκάστοτε βήμα της επανάληψης, ανανεώνοντας τα κόστη προς αυτούς και τα id των προηγούμενων κόμβων τους.
- Χρησιμοποιώντας την συνάρτηση reduce της βιβλιοθήκης CUB με δικό μας custom operator βρίσκουμε, από τους μη οριστικοποιημένους κόμβους, αυτόν με το μικρότερο κόστος, τον οριστικοποιούμε και συνεχίζουμε με αυτόν στην επόμενη επανάληψη.
- Αν δεν υπάρχει τέτοιος κόμβος, το κόστος που μας επιστρέφει η reduce είναι UINT_ MAX (άπειρο). Αυτό σημαίνει είτε ότι έχουν οριστικοποιηθεί όλοι οι κόμβοι είτε ότι οι εναπομείναντες μη οριστικοποιημένοι κόμβοι δεν είναι προσπελάσιμοι από τον κόμβο αφετηρία, δηλαδή δεν υπάρχει κανένα μονοπάτι προς αυτούς. Και στις δύο αυτές περιπτώσεις ο αλγόριθμος τερματίζει.

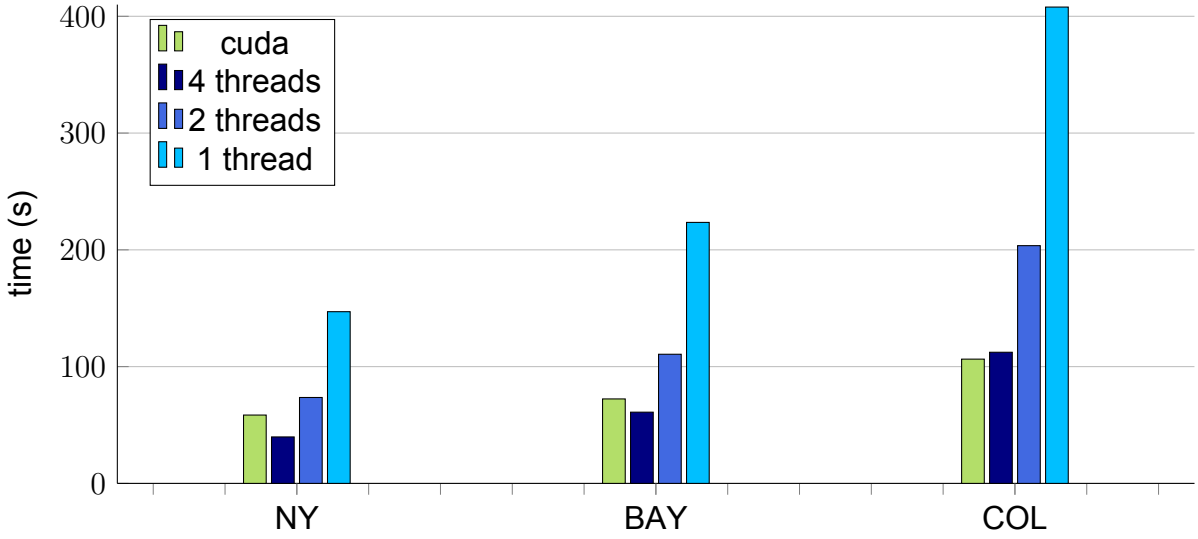
3.4 Υλοποίηση με openMP

Για λόγους πληρότητας των μετρήσεων μας αλλά και για την παρουσίαση των απαραίτητων συγκρίσεων μεταξύ των χρόνων εκτέλεσης μετατρέψαμε τον κώδικα που περιγράψαμε παραπάνω ώστε να μπορεί να τρέξει και στην CPU με την βοήθεια της βιβλιοθήκης openMP. Κατά την μετατροπή αυτή, ο κώδικας των τριών kernels μεταφέρθηκε στην main συνάρτηση του προγράμματος και πλέον εκτελείται με την εντολή *pragma omp for* για όσα νήματα (threads) καθορίσει ο χρήστης.

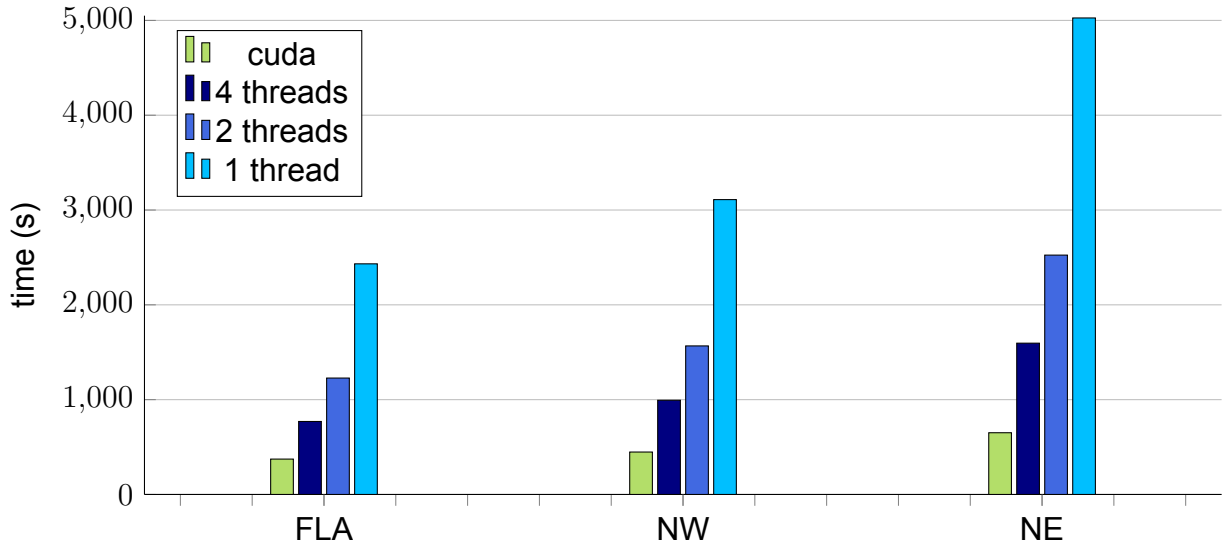
Η μόνη ειδοποιός διαφορά ανάμεσα στην υλοποίηση με openMP και σε αυτή του cuda έγκειται στον υπολογισμό του κόμβου με μικρότερο κόστος. Για να αποφύγουμε την μεγάλη καθυστέρηση που θα επέφερε το critical section για την προστασία της μεταβλητής που κρατά το μικρότερο κόστος από ταυτόχρονη πρόσβαση από διαφορετικά νήματα, χρησιμοποιούμε την εντολή *pragma omp parallel for reduction* με την οποία βρίσκουμε αποδοτικά την μικρότερη τιμή από τον πίνακα με τα κόστη.

3.5 Μετρήσεις

Στις μετρήσεις που πραγματοποιήσαμε δεν λάβαμε υπόψην τον χρόνο που απαιτούσαν η ανάγνωση του αρχείου εισόδου και η δημιουργία και αρχικοποίηση των δομών δεδομένων. Για την καλύτερη προβολή των αποτελεσμάτων παρουσιάζουμε 2 διαγράμματα χρόνων. Στο Σχήμα 9 διακρίνουμε τους χρόνους εκτέλεσης σε CPU και GPU για 3 γράφους μικρού μεγέθους (< 1 εκατομμύριο κόμβοι). Η εκτέλεση στον επεξεργαστή με 4 νήματα υπερίσχυσε οριακά αυτής του cuda στην κάρτα γραφικών για τους 3 αυτούς γράφους, κάτι που δεν συνέχισε να συμβαίνει στις επόμενες μετρήσεις, όπως θα δούμε παρακάτω.



Σχήμα 9: Χρόνοι εκτέλεσης Dijkstra (1/2)



Σχήμα 10: Χρόνοι εκτέλεσης Dijkstra (2/2)

Όσο αφορά τις μετρήσεις που πήραμε για τους 3 αμέσως μεγαλύτερους γράφους του Σχήματος 10 βλέπουμε ότι καθώς το μέγεθος του προβλήματος αυξάνεται σημαντικά (> 1 εκατομμύριο κόμβοι) οι εκτελέσεις της κάρτας γραφικών ολοκληρώνονται σε αρκετά μικρότερο χρόνο ακόμα και από αυτές του επεξεργαστή με 4 νήματα. Αυτό φυσικά διακρίνεται και στον παρακάτω πίνακα, όπου παρατηρούμε πως η επιτάχυνση των εκτελέσεων στην GPU αυξάνεται καθώς αυξάνεται και το πλήθος των κόμβων του γράφου.

	NY		COL		FLA		NE	
Threads	S	E	S	E	S	E	S	E
2	1.99x	99%	2x	100%	1.98x	99%	1.99x	99%
4	3.69x	92%	3.63x	90%	3.15x	79%	3.14x	78%
GPU	2.51x		3.83x		6.52x		7.72x	

Πίνακας 4: Επιτάχυνση (S) και αποδοτικότητα (E) για 4 γράφους

Ωστόσο, οι επιταχύνσεις που παίρνουμε με το CUDA δεν κυμαίνονται στα ίδια επίπεδα με αυτές του αλγορίθμου Boruvka που παρουσιάσαμε στην προηγούμενη ενότητα. Αυτό οφείλεται στο γεγονός ότι ο αλγόριθμος του Dijkstra έχει μια κατά πολύ πιο ακουλουθιακή μορφή από αυτήν του Boruvka, για αυτό και δεν επωφελούμαστε τα μέγιστα από το μεγαλύτερο πλήθος πυρήνων που προσφέρει η GPU.

4 Συμπεράσματα

Στις δύο προηγούμενες ενότητες αναλύσαμε και εξετάσαμε τους αλγορίθμους του Boruvka και του Dijkstra. Στόχος αυτής της μελέτης είναι να δούμε αν τελικά η απόδοση που παίρνουμε αξιοποιώντας την κάρτα γραφικών είναι τέτοια ώστε να μπορεί όχι μόνο να ανταγωνιστεί την απόδοση της CPU αλλά και να την ξεπεράσει, παίρνοντας την θέση της ως καλύτερη λύση για την επίλυση προβλημάτων τα οποία έχουν να κάνουν με την εκτέλεση των προαναφερθέντων αλγορίθμων.

Όπως είναι φυσικό, κατά την διάρκεια εκπόνησης των παράλληλων υλοποιήσεων για τους δύο αλγορίθμους αντιμετωπίσαμε ορισμένες δυσκολίες. Συγκεκριμένα, στον αλγόριθμο του Boruvka, δυσκολευτήκαμε κυρίως στην κατάνοηση του αλγορίθμου ενώ η υλοπόιηση ήταν πιο απλό έργο εξαιτίας της παράλληλης φύσης του αλγορίθμου. Αντιθέτως, ο αλγόριθμος του Dijkstra, αν και απλός στην κατανόηση, δεν μπορεί να παραλληλοποιηθεί πλήρως. Η μόνη προφανής παραλληλοποίηση που μπορεί να γίνει είναι στην εύρεση του κόμβου με ελάχιστο κοστος. Οτιδήποτε άλλο προσπαθήσαμε είτε αύξανε την πολυπλοκότητα του αλγορίθμου είτε έθετε επιπλέον ζητήματα συγχρονισμού στην υλοποίηση.

Τα συμπεράσματα που προέκυψαν από την μελέτη των μετρήσεων του αλγορίθμου του Boruvka, ο οποίος αναλαμβάνει να βρει το δέντρο ελαχίστου κόστους ενός γράφου, είναι πολύ θετικά. Το γεγονός ότι πρόκειται για ένα εκ φύσεως παράλληλο αλγόριθμο σε συνδυασμό με την εκμετάλλευση όλων των πλεονεκτημάτων που προσφέρει η απεικόνιση του γράφου με την δομή CSR μας έδωσε σπουδαίες επιταχύνσεις όχι μόνο σε σύγκριση με τους σειριακούς χρόνους εκτέλεσης αλλά και σε σύγκριση με τις αντίστοιχες επιταχύνσεις που πήραμε από την πολυνηματική εκτέλεση του αλγορίθμου στον επεξεργαστή. Συγκεκριμένα, είδαμε αφενός ότι η εκμετάλλευση των cuda cores απέφερε μια μέγιστη τιμή επιτάχυνσης της τάξεως του 27 για το πλήρες οδικό δίκτυο των ΗΠΑ και αφετέρου ότι όσο αυξανόταν το πλήθος των ακμών και των κορυφών του γράφου εισόδου τόσο αυξανόταν και το χάσμα στην διαφορά μεταξύ των επιταχύνσεων που παίρναμε με την GPU ενάντια στην CPU.

Παρόμοια συμπεράσματα προέκυψαν και από τις μετρήσεις που πραγματοποιήσαμε για τον αλγόριθμο εύρεσης ελαχίστων μονοπατιών Dijkstra. Παρά την ακολουθιακή μορφή του εν λόγο αλγορίθμου εξαιτίας της επιλογής της κορυφής με το ελάχιστο κόστος για κάθε κορυφή του γράφου μέχρι να οριστικοποιηθούν όλες, η επιτάχυνση που κερδίσαμε με τις εκτελέσεις σε cuda ήταν ικανοποιητικές αλλά όχι το ίδιο εντυπωσιακές με τις αντίστοιχες που είδαμε στον αλγόριθμο του Boruvka. Η μεγαλύτερη επιτάχυνση που μετρήσαμε στην κάρτα γραφικών ήταν 2 φορές μεγαλύτερη από αυτή του επεξεργαστή με 4 νήματα για τον γράφο με το πλήρες οδικό δίκτυο των ΗΠΑ. Θεωρητικά, σε περίπτωση που μας δινόταν η ευκαιρία να τρέξουμε με μεγαλύτερο πλήθος νημάτων (πχ 8) η επιτάχυνση τότε να ήταν σχεδόν ίση με αυτήν της κάρτας γραφικών. Ωστόσο, σε ένα τέτοιο σενάριο, ένας επεξεργαστής 8 νημάτων έχει πολύ αυξημένο κόστος αγοράς, αλλά ακόμα και τότε, αν είχαμε για είσοδο γράφο με πλήθος κορυφών και ακμών μεγαλύτερο από τον USA η επιτάχυνση της κάρτας γραφικών θα ήταν μεγαλύτερη ακόμα και από αυτήν που θα είχαμε με την εκτέλεση με 8 νήματα. Επομένως, θα μπορούσαμε να καταλήξουμε ότι και για τον αλγόριθμο του Dijkstra η κάρτα γραφικών μπορεί να μην προσφέρει εντυπωσιακή επιτάχυνση στους χρόνους εκτέλεσης, όμως δεν παύει να αποτελεί την πιο συμφέρουσα λύση από άποψη κόστους.

ΠΙΝΑΚΑΣ ΟΡΟΛΟΓΙΑΣ

Ξενόγλωσσος όρος	**Ελληνικός όρος**
Block	Τμήμα
Chip	Φύλλο ημιαγωγού
Converter	Μετατροπέας
Core	Πυρήνας
Critical section	Κρίσιμο τμήμα
Destination	Προορισμός
Edge	Ακμή
Efficiency	Αποδοτικότητα
Exclusive prefix sum	Αποκλειστικό άθροισμα προθεμάτων
First	Πρώτο
Flag	Σημαία
Grid	Πλέγμα
Keyword	Λέξη-κλειδί
Nodes	Κόμβοι
Outdegree	Βαθμός εξερχόμενων ακμών
Regular data structures	Ομοιόμορφα κατανεμημένη δομή δεδομένων
Shortest path	Συντομότερο μονοπάτι
Speedup	Επιτάχυνση
Struct	Δομή
Thread	Νήμα
Weight	Βάρος

ΣΥΝΤΜΗΣΕΙΣ–ΑΡΚΤΙΚΟΛΕΞΑ–ΑΚΡΩΝΥΜΙΑ

API	Application Programming Interface
CPU	Center Processing Unit
CSR	Compressed Sparse Row
CUB	CUDA UnBound
CUDA	Compute Unified Device Architecture
GB	GigaByte
GDDR	Graphics Double Data Rate
GHz	GigaHertz
GPU	Graphics Processor Unit
ID	Identifier
MST	Minimum Spanning Tree
OpenCL	Open Computing Language
OpenMP	Open Multi-Processing
RAM	Random Access Memory
TBB	Threading Building Blocks
VRAM	Video Random Access Memory
ΗΠΑ	Ηνωμένες Πολιτείες Αμερικής

ΠΑΡΑΡΤΗΜΑ

Η δομή CSR που έχουμε περιγράψει σε προηγούμενη ενότητα απαιτεί τα δεδομένα των ακμών του γράφου να είναι ταξινομημένα ως προς τον κόμβο από τον οποίο εξέρχεται η κάθε ακμή του γράφου. Αυτό είναι απαραίτητο διότι οι θέσεις των πινάκων first edge και out degree της δομής CSR βρίσκονται σε ένα προς ένα αντιστοιχία με τα id των κόμβων του γράφου. Για παράδειγμα, η τρίτη θέση του πίνακα out degree περιέχει το πλήθος των εξερχόμενων ακμών του κόμβου με id 3 στον γράφο.

1. Τύπος αρχείων .gr

Τα αρχεία εισόδου που χρησιμοποιήσαμε στην εργασία μας βρίσκονται στον ιστότοπο uniroma1.it. Ο τύπος των αρχείων αυτών έχει την κατάληξη .gr. Το είδος της πληροφορίας που περιέχει κάθε γραμμή ενός αρχείου τύπου .gr αναγνωρίζεται μοναδικά από κάποια γράμματα κλειδιά (keywords) που βρίσκονται ακριβώς στην αρχή κάθε γραμμής. Τα keywords αυτά περιγράφονται παρακάτω:

1. **c** – Μια γραμμή που αρχίζει με το γράμμα c αποτελεί σχόλιο και άρα το περιεχόμενο της δεν αποτελεί δεδομένα για τον γράφο που αναπαρίσταται στο αρχείο.
2. **p sp** – Μια γραμμή που αρχίζει με τα γράμματα p sp, τα οποία σημαίνουν problem shortest path, αντίστοιχα, δηλώνει την πρώτη ουσιαστική πληροφορία που αφορά το γράφο και θα πρέπει να είναι η πρώτη γραμμή στο αρχείο η οποία δεν είναι σχόλιο. Αυτός ο τύπος γραμμής πρέπει να εμφανίζεται ακριβώς μία φορά σε κάθε αρχείο. Μετά από το keyword p sp πρέπει να ακολουθούν ακριβώς δύο ακέραιοι και μη αρνητικοί αριθμοί οι οποίοι δηλώνουν το πλήθος των κόμβων και των ακμών του γράφου αντίστοιχα.
3. **a** - Μια γραμμή που αρχίζει με τον χαρακτήρα a περιγράφει μια ακμή του γράφου με τρεις αριθμούς n, m και w. Ο αριθμός n δηλώνει το κόμβο από τον οποίο εξέρχεται η ακμή, ενώ ο αριθμός m τον κόμβο στον οποίο εισέρχεται. Τέλος, ο αριθμός w υποδηλώνει το βάρος της συγκεκριμένης ακμής.

Σε ένα αρχείο .gr μπορεί να υπάρχει μη προκαθορισμένος αριθμός από γραμμές με keyword c, ενώ δεν απαιτείται κάποια μορφή ταξινόμησης στις γραμμές με keyword a που αναπαριστούν τις ακμές του γράφου. Αυτή η έλλειψη ταξινόμησης των ακμών καθιστά τα αρχεία .gr ακατάλληλα για είσοδο στις εφαρμογές που έχουμε αναπτύξει, αφού δεν μπορεί να παραχθεί άμεσα από αυτά η δομή CSR. Η κατάλληλη επεξεργασία και μετατροπή των αρχείων .gr σε κάποια άλλη μορφή είναι απαραίτητη. Η μορφή αρχείων που έχουμε επιλέξει να μετατρέπουμε τα αρχεία τύπου .gr έχει κατάληξη .edges και η δομή της περιγράφεται παρακάτω.

2. Τύπος αρχείων .edges

Ο τύπος αρχείων .edges δεν περιέχει keywords που αναγνωρίζουν μοναδικά μια γραμμή. Παρόλα αυτά πρέπει να αρχίζει με μια γραμμή που περιλαμβάνει δύο ακέραιους θετικούς αριθμούς που δηλώνουν το πλήθος των κόμβων και των ακμών του γράφου. Έπειτα, κάθε γραμμή που ακολουθεί παριστάνει μια ακμή και περιέχει τρεις αριθμούς n, m και w που δηλώνουν τον κόμβο n από τον οποίο εξέρχεται, τον κόμβο m στον οποίο εισέρχεται και το βάρος w κάθε ακμής του γράφου. Οι γραμμές αυτές θα πρέπει να ταξινομημένες με βάση τον αριθμό n. Τυχόν κενές γραμμές αγνοούνται.

Η ταξινόμηση των δεδομένων που διαθέτει ο τύπος αρχείων .edges προσφέρει άμεση παραγωγή της δομής CSR που χρησιμοποιούν οι εφαρμογές μας με μόλις μια ανάγνωση του αρχείου.

3. Μετατροπή αρχείου .gr σε .edges

Για την μετατροπή των αρχείων εισόδου από τον έναν τύπο στον άλλο έχουμε αναπτύξει έναν απλό μετατροπέα (converter). Ο μετατροπέας αυτός δέχεται στην είσοδο του ένα αρχείο τύπου .gr και παράγει στην έξοδο το αντίστοιχο αρχείο τύπου .edges. Η ταξινόμηση που πραγματοποιεί γίνεται με την γνωστή μέθοδο quick sort. Τέλος, εξασφαλίζεται ότι η αρίθμηση των id των κόμβων αρχίζει από το 0, όπως συμβαίνει και στην αρίθμηση των θέσεων στους πίνακες της γλώσσας C, στην οποία έχουμε γράψει τις εφαρμογές μας, για λόγους ευχρηστίας.

ΑΝΑΦΟΡΕΣ

[1] CUDA API,
https://en.wikipedia.org/wiki/CUDA

[2] openCL,
https://en.wikipedia.org/wiki/OpenCL

[3] Introduction to “Irregular” Algorithms, Νοέμβριος 2010,
http://www.cs.utah.edu/~mhall/cs4961f10/CS4961-L20.pdf

[4] Βιβλιοθήκη openMP,
http://openmp.org/wp/

[5] Minimum spanning tree,
https://en.wikipedia.org/wiki/Minimum_spanning_tree

[6] Sparse matrix,
https://en.wikipedia.org/wiki/Sparse_matrix

[7] Ο αλγόριθμος του Kruskal,
https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

[8] Ο αλγόριθμος του Prim,
https://en.wikipedia.org/wiki/Prim%27s_algorithm

[9] Ο αλγόριθμος του Boruvka,
https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm

[10] Otakar Boruvka,
http://tinyurl.com/zvwx67l

[11] C. da Silva Sousa, A. Mariano, A. Proenca,
“A Generic and Highly Efficient Parallel Variant of Boruvka’s Algorithm”,
2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing,
http://www.pdp2015.org

[12] Βιβλιοθήκη CUB,
https://nvlabs.github.io/cub/

[13] Prefix Sum using OpenMP,
https://codingnights.wordpress.com/2011/08/16/prefix-sum-using-openmp/

[14] Ο αλγόριθμος του Dijkstra,
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

[15] Edsger W. Dijkstra,
https://en.wikipedia.org/wiki/Edsger_W._Dijkstra