ΥΠΟΛΟΓΙΣΜΟΣ FGLT ΣΤΗΝ GPU ΜΕ ΧΡΗΣΗ CUDA

Σε αυτήν την εργασία στόχος είναι η υλοποίηση κώδικα CUDA για τον γρήγορο υπολογισμό του fast graphlet transform, όπως περιγράφεται στο paper [1] που δόθηκε στην εκφώνηση και στον αντίστοιχο κώδικα. Ζητούμενο αυτής της άσκησης είναι ο υπολογισμός των σ1, σ2, σ3, σ4.
Στην παρακάτω αναφορά δίνονται σύντομα πληροφορίες για την υλοποίηση του κώδικα και γίνεται μία παρουσίαση των αποτελεσμάτων και συμπερασμάτων που προέκυψαν. Ο κώδικας βρίσκεται στο Github link:

https://github.com/kprattis/parallel_programming_ex3

- ## Υλοποίηση Αλγορίθμου και περιγραφή Kernels

Η δομή της main καθώς και οι συναρτήσεις I/O που υπάρχουν στην βιβλιοθήκη fglt δεν αλλάχθηκαν σχεδόν καθόλου, με σκοπό αφενός την πιο εύκολη επέκταση με CUDA και αφετέρου την επικέντρωση στο ζητούμενο της άσκησης που είναι η υλοποίηση των υπολογισμών. Οι ελάχιστες τροποποιήσεις του κώδικα περιλαμβάνουν την αφαίρεση των υπολογισμών για συχνότητες πάνω από την σ4 και την προσθήκη της νέας συνάρτησης “compute” που χρησιμοποιεί την GPU.

Η συνάρτηση αυτή, δέχεται τα ίδια ορίσματα με την multithreaded εκδοχή της στη βιβλιοθήκη. Στο εσωτερικό της, αρχικά γίνεται η δήλωση και η μεταφορά μνήμης στην GPU με χρήση cudaMalloc και cudaMemcpy, στη συνέχεια καλούνται τα kernels που εκτελούν τους υπολογισμούς και με κατάλληλο συγχρονισμό μεταξύ αυτών και της GPU τα αποτελέσματα μεταφέρονται στην CPU ώστε να επιστραφούν από την συνάρτηση. Παρακάτω γίνεται μία πιο λεπτομερής παρουσίαση των kernels που υλοποιήθηκαν:

row_sum_red: Δέχεται ως είσοδο έναν πίνακα σε μορφή csr/csc και υπολογίζει το άθροισμα των στοιχείων κάθε γραμμής, αποθηκεύοντάς τα σε ένα διάνυσμα. Χρησιμοποιείται πρώτα για τον υπολογισμό του p_1 (Ae) και στην συνέχεια για τον υπολογισμό του $c_3(C_3e)$.

C3_calc: Υπολογίζει τον πίνακα $C_3\left(A \odot A^2\right)$ με την ακόλουθη διαδικασία: Επειδή ο A είναι αραιός πίνακας, ο C_3 θα είναι επίσης αραιός πίνακας. Συνεπώς, θα έχει το πολύ m μη μηδενικά στοιχεία και μάλιστα θα βρίσκονται στις ακριβώς ίδιες θέσεις με τα μη μηδενικά στοιχεία του πίνακα A. Για αυτό, αρκεί να σηκώσουμε συνολικά m threads, ώστε να υπολογίσουν το καθένα μία τιμή του αποτελέσματος. Κάθε thread, αντιστοιχίζεται σε ένα μη-μηδενικό στοιχείο του A. Έτσι, αν ένα thread έχει αντιστοιχηθεί στο στοιχείο που στη csc μορφή έχει δείκτη id, τότε εύκολα διαπιστώνουμε ότι αυτό βρίσκεται στη γραμμή ii[id]. Ο εντοπισμός του άλλου δείκτη είναι λίγο πιο περίπλοκος. Αρχικά, γνωρίζουμε ότι τα στοιχεία του πίνακα ii με δείκτες από jStart[ii[id]] -> jStart[ii[id] + 1] δείχνουν τους δείκτες των γραμμών στα οποία υπάρχουν μη μηδενικά στοιχεία στην στήλη ii[id]. Επειδή ο A είναι συμμετρικός, το τελευταίο είναι ισοδύναμο με το "σε ποιες στήλες υπάρχουν μη μηδενικά στοιχεία στην γραμμή ii[id]". Εκμεταλλευόμενοι και την ταξινόμηση των δεικτών του πίνακα ii, αρκεί να βρούμε ποιος δείκτης στο παραπάνω εύρος ικανοποιεί πρώτος το $id < jStart[col],\ col \in \{jStart[ii[id]], jStart\left[ii[id]+1\right]\}$. Για αυτό γίνεται μία διαδικασία δυαδικής αναζήτησης. Τέλος, έχοντας βρει τις συντεταγμένες του μη μηδενικού στοιχείου, υπολογίζουμε το $\sum_{\lambda} \alpha_{i\lambda}\alpha_{\lambda j}$ όπου i, j είναι οι δείκτες που υπολογίστηκαν παραπάνω.

p2: Αυτή η ρουτίνα, όπως και οι επόμενες περιμένει το p1 και υπολογίζει αρχικά το Ap_1 και στη συνέχεια το $Ap_1 - p_1$.

d3: Αυτή η ρουτίνα, εκτελεί τον υπολογισμό του p2 παίρνοντας ως είσοδο το p1 με ένα απλό if έλεγχο για το p1 - 1.

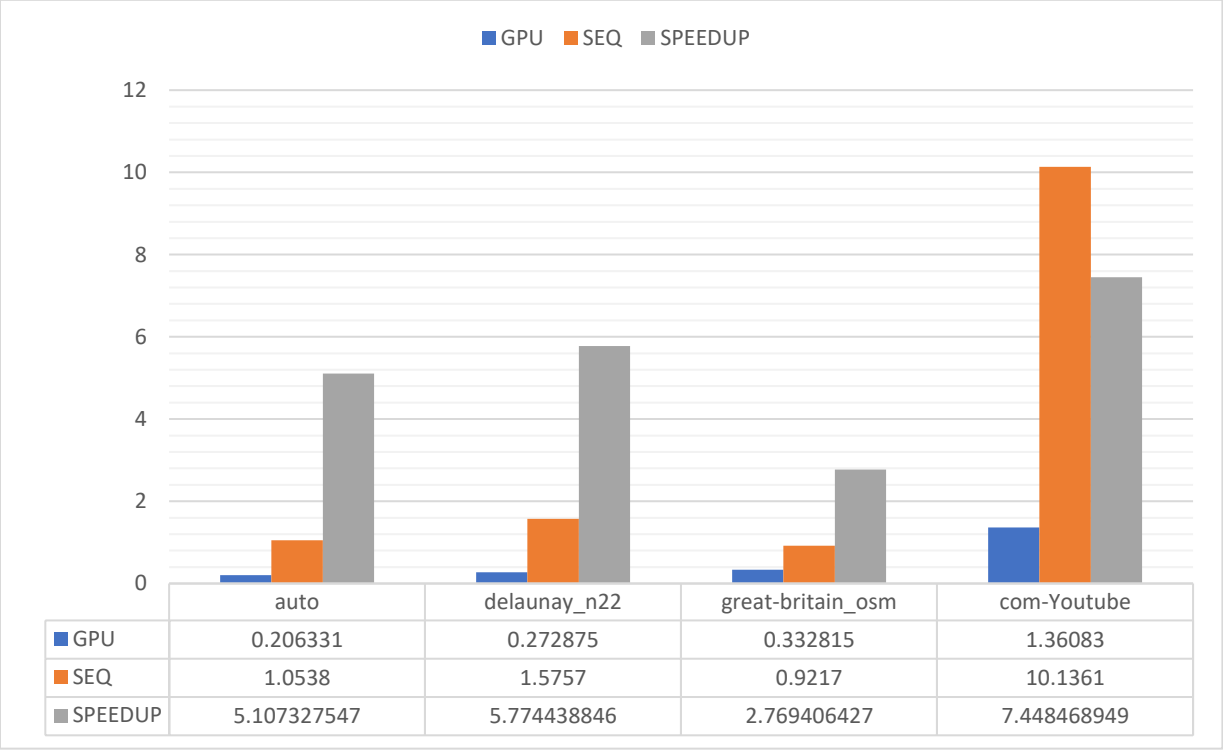
raw2net: Πραγματοποιεί την μετατροπή από raw frequencies σε net.

Η μεγαλύτερη βαρύτητα δόθηκε στο c3 καθώς φάνηκε να είναι ο πιο «βαρύς» υπολογισμός που ήθελε την μεγαλύτερη προσοχή.

- ## Παρουσίαση Αποτελεσμάτων

Για την αξιολόγηση των υλοποιήσεων χρησιμοποιήθηκαν οι ακόλουθοι γράφοι από το http://sparse.tamu.edu/: auto, delaunay_n22, great-britain_osm και com-Youtube. Επίσης, για την παρουσίαση της απόδοσης χρησιμοποιήθηκαν εργαλεία της NVIDIA και συγκεκριμένα το NVIDIA nsight compute για την μέτρηση του memory & computation throughput και το NVIDIA visual profiler για την παρουσίαση του timeline of execution.

Παρακάτω τα αποτελέσματα των runs:

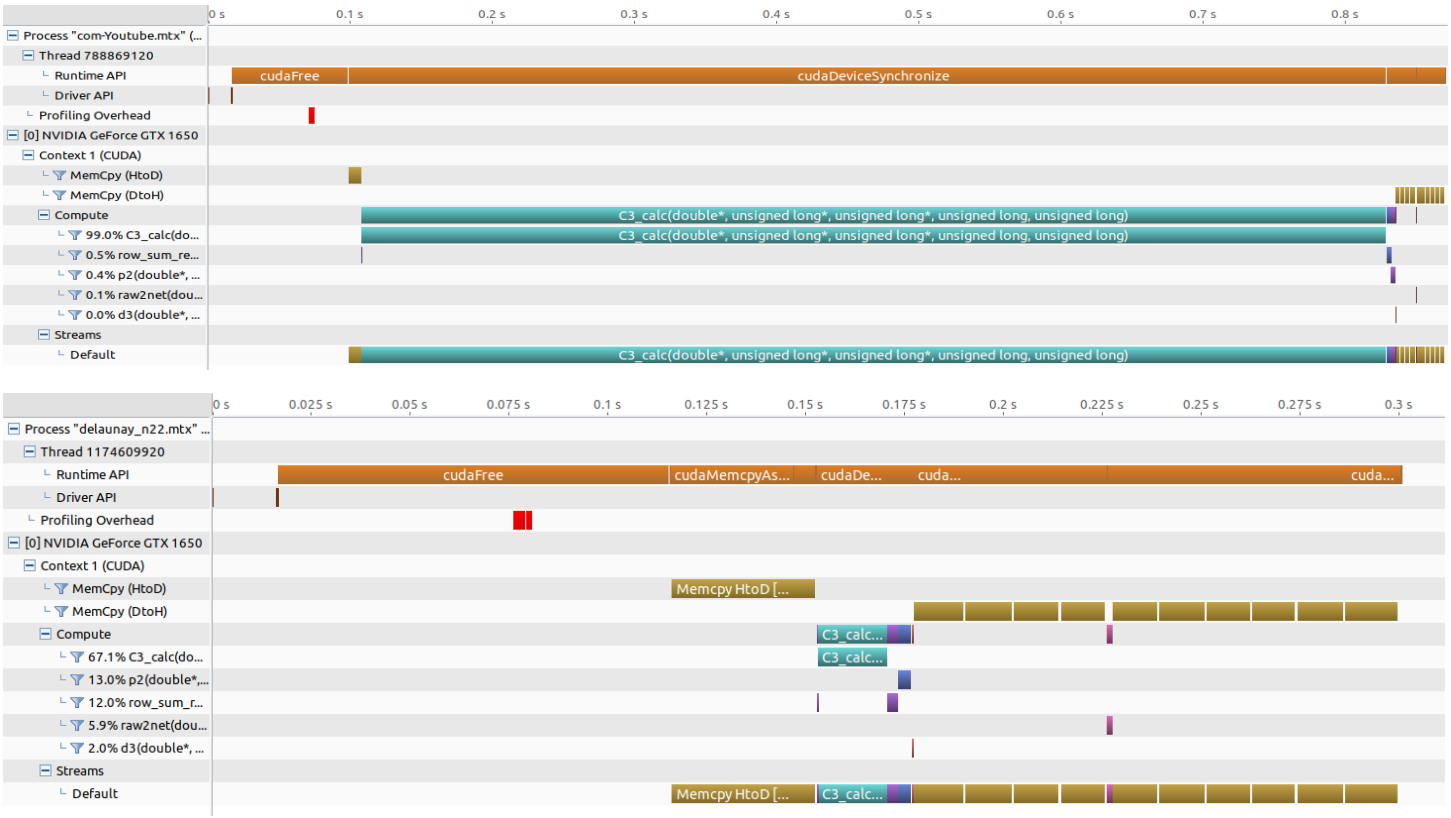


	auto	delaunay_n22	great-britain_osm	com-Youtube
GPU	0.206331	0.272875	0.332815	1.36083
SEQ	1.0538	1.5757	0.9217	10.1361
SPEEDUP	5.107327547	5.774438846	2.769406427	7.448468949

Εικόνα 1: Σύγκριση με την σειριακή υλοποίηση. Οι χρόνοι έχουν μετρηθεί σε NVIDIA GEFORCE GTX 1650 (GPU) και για το σειριακό σε AMD Ryzen 5 5000H (CPU).

	row_sum_red(p1)		C3_calc		row_sum_red(c3)		p2		d3		raw2net	
auto	29.58	86.08	2.93	34.48	17.1	90.62	19.44	80.07	81	59.51	22.58	81.59
delaunay_n22	28.85	90.76	31.48	44.39	40.9	74.64	56.28	74.94	83.65	65.81	22.45	82.99
great-britain_osm	28.75	90.9	24.42	46.17	57.82	90.59	38.87	70.07	83.67	66.24	21.93	83.23
com-Youtube	29.19	89.17	46.29	23.2	34.12	8.75	43.72	33.11	82.66	63.5	22.52	82.41

Πίνακας 1: Οι πράσινες στήλες δείχνουν το compute throughput (%) και οι κόκκινες το memory throughput (%) όπως αυτά υπολογίστηκαν από το NVIDIA NSIGHT COMPUTE για κάθε γράφο και για κάθε kernel. Όπως φαίνεται την μεγαλύτερη βελτίωση χρίζει ο υπολογισμός των C3 & c3.



Εικόνα 2: Execution Timeline για το com-Youtube (πάνω) και το delaunay_n22 (κάτω). Φαίνεται ότι στην πρώτη περίπτωση ο περισσότερος χρόνος ξοδεύεται στον υπολογισμό του C3, ενώ στην δεύτερη σε memory transfers μεταξύ host και device. Αυτό εξηγείται καθώς ο πρώτος γράφος είναι μικρότερος σε μέγεθος αλλά πιο πυκνός, ενώ ο δεύτερος είναι μεγαλύτερος σε μέγεθος αλλά πιο αραιός.

Παρατηρούμε ότι υπάρχει μία σημαντική επιτάχυνση σε αυτά τα γραφήματα, που όμως ποικίλει ανάλογα το μέγεθος και την «πυκνότητα» δηλαδή τον αριθμό των non-zero στοιχείων του γραφήματος. Ενδιαφέρον παρουσιάζει ότι αν και ο γράφος com-Youtube είναι ο πιο αργός, παρουσιάζει την μεγαλύτερη επιτάχυνση.

Επιπλέον, από τα ενδεικτικά χρονοδιαγράμματα εκτέλεσης, παρατηρούμε ότι αυτό που διαρκεί το μεγαλύτερο ποσοστό του χρόνου εκτέλεσης είναι το kernel που υπολογίζει το C3. Ακόμα, φαίνεται να είναι και αυτό που χρησιμοποιεί λιγότερο αποδοτικά την GPU, από τον πίνακα με τα ποσοστά διεκπεραίωσης.

- ## Συμπεράσματα – Προτάσεις για βελτίωση

Το πιο χρονοβόρο κομμάτι των υπολογισμών είναι ο υπολογισμός του c3. Στην παρούσα εργασία έγινε μία προσπάθεια για μία υλοποίηση που αποβλέπει πρώτα στον υπολογισμό του C3 και συγκεκριμένα μόνο των μη μηδενικών στοιχείων του. Με αυτόν τον τρόπο αποφεύγεται η ανάγκη για υπολογισμό ολόκληρου του A * A, που δεν είναι κατά ανάγκη αραιός πίνακας. Πιθανές βελτιώσεις για αυτό το πρόβλημα είναι η εκμετάλλευση της συμμετρίας του A ώστε να υπολογίζονται μόνο τα μισά αποτελέσματα, κάποιο tiling για εκμετάλλευση του shared memory ενώ κάτι που θα βοηθούσε αρκετά αλλά θα κόστιζε σε επιπλέον μνήμη είναι η αποθήκευση των col indices ώστε να μην καταναλώνεται χρόνος στην εύρεση αυτών των δεικτών.

Ένα άλλο σημείο είναι η μεταφορά δεδομένων μεταξύ CPU και GPU. Αν και σε πιο μεγάλους γράφους είναι ανεπαίσθητη, σε μικρότερους είναι αυτή που δημιουργεί την μεγαλύτερη καθυστέρηση. Για την βελτίωση σε αυτό το κομμάτι θα μπορούσαν να χρησιμοποιηθούν ίσως ασύγχρονες επικοινωνίες, οι οποίες όμως θα προσέθεταν επιπλέον initialization overhead.

Τέλος, αν και λόγω τεχνικού ζητήματος δεν έγινε η σύγκριση της έκδοσης Cilk με την έκδοση CUDA που υλοποιήθηκε, θα παρουσίαζε ενδιαφέρον η μελέτη των διαφορών στους χρόνους των δύο προσεγγίσεων.

- ## Βιβλιογραφία

1. **Floros Dimitris, Pitsianis Nikos and Sun Xiaobai** Fast Graphlet Transform of Sparse Graphs [Book Section] // IEEE High Performance Extreme Computing Conference. - 2020. (https://arxiv.org/abs/2007.11111)