Ελευθεριάδης Χαράλαμπος Παράλληλα και Διανεμημένα ΑΕΜ: 9257 | 6936981498
elefchar@ece.auth.gr

Reverse Cuthill McKee – Serial and OpenMP Implementation in C

Github Link: https://github.com/LambisElef/reverse_Cuthill_McKee

Περιγραφή του προβλήματος:

Ως τέταρτη εργασία του μαθήματος επιλέχθηκε η υλοποίηση ενός αλγορίθμου που κάνει αναδιάταξη ενός αραιού πίνακα γειτνίασης σειριακά σε C και παράλληλα με τη χρήση OpenMP. Από τη δοθείσα λίστα επιλέχθηκε ο Reverse Cuthill McKee. Αυτός περιγράφεται επαρκώς στο αντίστοιχο λήμμα στη βικιπαίδεια.

Σειριακή υλοποίηση:

Το πρόγραμμα επιλέχθηκε να λειτουργεί με τη χρήση αρχείων csv και πινάκων σε μορφή COO (σειρά, στήλη, τιμή) και μάλιστα τα στοιχεία αποθηκευμένα κατά στήλη. Οι πίνακες που χρησιμοποιήθηκαν ήταν από το University of Florida Sparse Matrix Collection και μετατράπηκαν στην επιθυμητή μορφή με τη χρήση του Matlab. Επίσης, ως είσοδος για κάθε τέτοιο πίνακα ζητείται η διάστασή του, καθώς και το πλήθος των μη-μηδενικών στοιχείων του.

Αρχικά, υπολογίζεται ο βαθμός του κάθε κόμβου, ο οποίος ορίζεται ως το πλήθος των γειτόνων του. Κατόπιν, ο πίνακας συντεταγμένων μετασχηματίζεται σε έναν δισδιάστατο πίνακα γειτόνων, όπου η πρώτη διάσταση καθορίζει τον κόμβο τον οποίο εξετάζουμε, ενώ η δεύτερη απαριθμεί τους γείτονες του κόμβου αυτού. Οι γείτονες αυτοί είναι το περιεχόμενο του πίνακα.

Στη συνέχεια, επιλέγεται ο κόμβος με τον μικρότερο βαθμό ως περιφερειακός και οι γείτονές του προστίθενται με αύξοντα βαθμό στο διάνυσμα της αναδιάταξης. Στη συνέχεια, προστίθενται στο διάνυσμα της αναδιάταξης με αύξοντα βαθμό οι γείτονες του κάθε κόμβου που βρίσκεται ήδη μέσα στο διάνυσμα, αφού πρώτα ελεγχθεί ότι δεν περιλαμβάνονται ήδη σε αυτό. Έχει ληφθεί μέριμνα και για την περίπτωση ύπαρξης μη-συνδεδεμένων γράφων σε ένα πίνακα.

Τελικά, το διάνυσμα της αναδιάταξης αποθηκεύεται στο αρχείο <sparse_array_name>-res με την αντίστροφη σειρά, κάτι που σηματοδοτεί και το τέλος του αλγορίθμου.

Παράλληλη υλοποίηση:

Με το flag OPENMP σε τιμή διάφορη του μηδενός ενεργοποιείται η παράλληλη εκτέλεση του αλγορίθμου. Ο παραλληλισμός εφαρμόστηκε σε δύο επαναληπτικούς for βρόγχους.

Ο πρώτος αφορά τον υπολογισμό του διανύσματος των βαθμών των κόμβων. Εκεί χρειάστηκε να συνδυαστεί με reduction, ώστε κάθε thread να διατηρεί το δικό του τοπικό αντίγραφο του διανύσματος των βαθμών και στο τέλος να αθροιστούν σε έναν τελικό διάνυσμα, ώστε να αποφευχθούν race conditions και να μη χρειαστεί να εφαρμοστεί critical ή atomic section, κάτι που θα μείωνε κατά πολύ την απόδοση του παραλληλισμού.

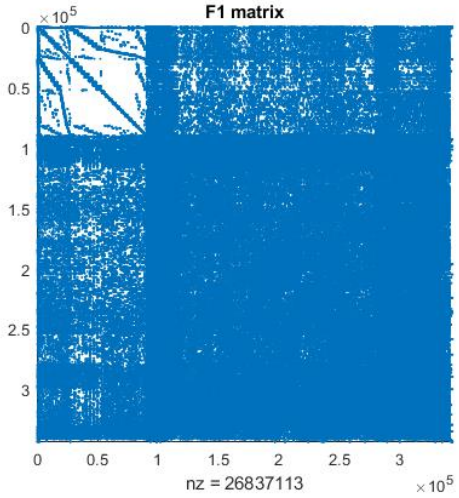
Ο δεύτερος επαναληπτικός βρόγχος αφορά τον μετασχηματισμό του πίνακα σε πίνακα γειτόνων όπως περιεγράφηκε παραπάνω. Σε συνδυασμό με το COO format του αρχικού πίνακα και την

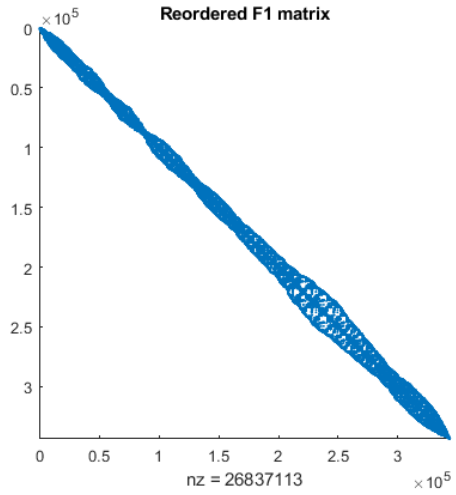
αποθήκευση των στοιχείων κατά στήλη μπορούμε να υπολογίσουμε τους γείτονες κάθε κόμβου παράλληλα.

Αξίζει να σημειωθεί ότι δοκιμάστηκε μια παράλληλη εκδοχή της merge sort για την ταξινόμηση των γειτόνων του peripheral κόμβου, ωστόσο δεν προσέφερε καμία βελτίωση στο χρόνο εκτέλεσης της ταξινόμησης, λόγω του σχετικά μικρού πλήθους στοιχείων που είχε να ταξινομήσει. Ακόμα και στους μεγαλύτερους πίνακες του UF Repository, η χρήση της παράλληλης merge sort δεν βοήθησε. Γι' αυτό το λόγο και δε συμπεριλήφθηκε στην τελική έκδοση του κώδικα.

Επικύρωση ορθής λειτουργίας:

Μιας και ο αλγόριθμος είναι ευριστικός, κυρίως λόγω των διαφορετικών τρόπων υπολογισμού του βαθμού κάθε κόμβου, αλλά και της ταξινόμησης γειτόνων του περιφερειακού κόμβου (τι γίνεται στην περίπτωση που δύο κόμβοι έχουν τον ίδιο βαθμό) δε μπορούμε απλά να συγκρίνουμε την αναδιάταξη που βγάζει η δική μου υλοποίηση με αυτή που βγάζει μία άλλη, όπως αυτή του Matlab. Ωστόσο, εξετάζοντας το αποτέλεσμα της αναδιάταξης οπτικά με τη χρήση της spy στο Matlab, ή αναλυτικά με τη χρήση της bandwidth στο Matlab, βλέπουμε την αποτελεσματικότητα του αλγορίθμου, ο οποίος πετυχαίνει το σκοπό του, δηλαδή τη μείωση του bandwidth του αναδιατεταγμένου πίνακα σε σχέση με αυτό του αρχικού.





Benchmarking:

	Μέγεθος (n)	Μη-Μηδενικά Στοιχεία	Αρχικό Bandwidth
filter3D	106.437	2.707.179	8.276
F1	343.791	26.837.113	343.754
audikw_1	943.695	77.651.847	925.946

Για τη μέτρηση της απόδοσης του προγράμματός μας χρησιμοποιήθηκαν οι πίνακες filter3D, F1 και audikw_1. Με το flag BENCH_SECTORS ενεργοποιείται μια δυνατότητα για μέτρηση του χρόνου εκτέλεσης διαφόρων κομματιών του προγράμματος όπως αυτά έχουν χωριστεί στον πηγαίο κώδικα.

Ως υφιστάμενη προς σύγκριση υλοποίηση χρησιμοποιήθηκε η συνάρτηση symrcm της Matlab. Το πρόγραμμα έγινε compile με τον gcc-10 και το O3 optimization flag. Το σύστημα στο οποίο έγιναν οι μετρήσεις είχε ως επεξεργαστή έναν AMD Ryzen 5 1600 @3.7GHz, ο οποίος στον μεγαλύτερο πίνακα οδήγησε σε επιτάχυνση του υπολογισμού των βαθμών των κόμβων κατά τουλάχιστον 5 φορές σε σχέση με τη σειριακή υλοποίηση. Οι παρακάτω χρόνοι είναι οι **μέσοι όροι 5 διαδοχικών εκτελέσεων** του αλγορίθμου σε **millisecond**.

	Matlab	**Σειριακή Υλοποίηση**	**Σειριακή σε σύγκριση με Matlab**	**Παράλληλη Υλοποίηση**	**Παράλληλη σε σύγκριση με Matlab**
filter3D	37,505	17,925	**-52%**	20,685	**-45%**
F1	224,449	125,312	**-44%**	92,420	**-59%**
audikw_1	616,017	393,272	**-36%**	238,945	**-61%**

Συμπληρωματικά, παρατίθεται ένας πίνακας που συγκρίνει τα bandwidth των πινάκων μετά την εφαρμογή του αλγορίθμου:

	Matlab	**Σειριακή Υλοποίηση**	**Παράλληλη Υλοποίηση**
filter3D	3.841	3.498	3.498
F1	10.052	19.022	19.022
audikw_1	35.102	39.815	39.815

Σχόλια:

Αξίζει να σημειωθεί ότι μεγάλο ρόλο στην απόδοση έπαιξε η μείωση των cache invalidate περιστατικών κατά τον μετασχηματισμό του πίνακα εντός του κώδικα στην τελευταία γραμμή του block με σχόλιο “Transforms the sparse matrix”. Εκεί, ο πίνακας σκανάρεται και αποθηκεύεται κατά στήλη.

Ακόμη, σε ότι αφορά την ταξινόμηση των γειτόνων του peripheral node, δοκιμάστηκαν οι δύο αλγόριθμοι quick sort και merge sort όπως ορίζονται στο sort.c. Παρόλο που η quick sort υπερτερούσε για λίγα millisecond στην περίπτωση μικρότερων πινάκων, σε μεγαλύτερους πίνακες η merge sort υπερτερούσε για δεκάδες millisecond γι’ αυτό και επιλέχθηκε αυτή, μιας και δε διέθετα το κουράγιο να μελετήσω εάν η συμπεριφορά αυτή διατηρούνταν πάντοτε, μετρώντας την απόδοση της κάθε sort για ένα μεγάλο πλήθος αραιών πινάκων.

Συμπερασματικά, η παράλληλη υλοποίηση φαίνεται να αξίζει για πίνακες με πλήθος μη-μηδενικών στοιχείων μεγαλύτερο των 26 εκατομμυρίων, δηλαδή σχετικά μεγάλους αραιούς πίνακες. Η σειριακή από την άλλη είναι γρηγορότερη για πλήθος μη-μηδενικών στοιχείων μικρότερο των 3 εκατομμυρίων, ωστόσο αυτά τα συμπεράσματα έχουν βγει από τους 3 μόνο πίνακες που δοκιμάστηκαν. Σίγουρα μπορούν να γίνουν πιο εκτενείς δοκιμές, ωστόσο θα χρειαζόταν περεταίρω αυτοματοποίηση για σύγκριση των χρόνων της σειριακής και της παράλληλης υλοποίησης.

Πηγές:

- Βικιπαίδεια
- GeeksForGeeks
- University of Florida Sparse Matrix Collection
- Matlab Sparse Matrix Reordering