Παράλληλα και Διανεμημένα Συστήματα
Εργασία 1

Φοιτητής : Ζηκόπης Ευάγγελος
ΑΕΜ : 8808
Email : vagzikopis@gmail.com
3/11/2019

Γενικά

Στόχος της συγκεκριμένης εργασίας ήταν η κατασκευή ενος vantage point δέντρου. Η υλοποίηση του συγκεκριμένου προβλήματος έπρεπε να γίνει με τέσσερις διαφορετικούς τρόπους, μια σειριακή υλοποίηση και τρείς υλοποιήσεις που χρησιμοποιούν τεχνικές παράλληλου προγραμματισμού. Τα frameworks που χρησιμοποιήθηκαν για τις παράλληλες υλοποιήσεις είναι τα pthreads, openmp και cilk. Τα ονόματα των εκτελέσιμων που δημιουργήθηκαν είναι *vptree_sequential*, *vptree_pthreads*, *vptree_openmp*, *vptree_cilk*. Κάθε υλοποίηση θα αναλυθεί παρακάτω. Όλα τα αρχεία κάνουν include το header *vptree.h*, όπου δηλώνονται όλες οι συναρτήσεις και τα structs που χρησιμοποιούνται. Τονίζεται ότι η βάση όλων των υλοποιήσεων είναι η *vptree_sequential*. Παραθέτω τον σύνδεσμο όπου βρίσκεται ο κώδικας της εργασίας.

https://github.com/vagzikopis/vptree.git

Vptree_sequential

Κατά την κλήση της *buildvp*, οριστικοποιείται ένας πίνακας που περιέχει τα indexes όλων των σημείων και καλείται η συνάρτηση *work*, η οποία αναδρομικά κατασκευάζει το δέντρο. Τα ορίσματα της είναι το σύνολο των σημείων, ο αριθμός τους, οι διαστάσεις τους και ο πίνακας με τα indexes. Αρχικά, γίνεται έλεγχος αν ο αριθμός των σημείων είναι μηδέν ή ένα, οπότε και επιστρέφεται το κατάλληλο δέντρο για την κάθε περίπτωση. Στη συνέχεια επιλέγεται ως vantage point το τελευταίο σημείο και υπολογίζονται οι αποστάσεις των υπόλοιπων σημείων από αυτό. Μέσω της *quickselect* επιλέγεται η μέση απόσταση από το vantage point. Έπειτα, τα υπόλοιπα σημεία χωρίζονται σε δύο πίνακες ανάλογα με το αν η απόσταση τους απο το vantage point είναι μεγαλύτερη ή μικρότερη από την μέση απόσταση που έχει επιλεχθεί. Τέλος, αφού έχουν δημιουργηθεί και οι κατάλληλοι πίνακες με τα indexes των σημείων καλείται αναδρομικά η *work* για να κατασκευάσει τα υποδένδρα που πρέπει.

Vptree_pthreads

Εδώ υλοποιούμε την αναδρομή δημιουργώντας νεα threads. Ο αριθμός των threads που έχουν δημιουργηθεί αποθηκεύεται στην μεταβλητή *active_threads*. Για να περιορίζουμε τη δημιουργία νέων threads ορίζουμε ένα threshold, την μεταβλητή *max_threads*. Η τιμή που της έδωσα είναι 10 καθώς με λίγες

δοκιμές που έκανα έδινε τα καλύτερα αποτελέσματα. Ωστόσο, πιο διεξοδικά πειράματα μπορεί να οδηγήσουν σε άλλα thresholds. Έτσι, ένα thread όταν θέλει να κατασκευάσει τα υποδένδρα που πρέπει ελέγχει τον αριθμό των *active_threads*. Αν αυτός είναι ίσος με το threshold τότε δεν δημιουργεί νέο thread αλλά καλεί αναδρομικά τον εαυτό του. Επίσης για τον υπολογισμό των αποστάσεων, δημιουργούνται threads ανάλογα με τον αριθμό των σημείων προκειμένου να υπολογίσουν τις αποστάσεις παράλληλα. Γενικά όταν τα σημεία είναι λίγα, και η αναδρομή και οι αποστάσεις υπολογίζονται σειριακά καθώς η δημιουργία νέων threads προσθέτει overhead. Η χρήση των pthreads δεν είναι και τόσο user-friendly, ωστόσο με την κατάλληλη επιλογή παραμέτρων μπορούν να επιτύχουν σχετικά καλά αποτελέσματα. Τα όρια που χρησιμοποίησα για την εναλλαγή σε σειριακή υλοποίηση ήταν 10 spawned threads για τις αναδρομές και n>1000 για τον υπολογισμό των αποστάσεων.

Vptree_openmp

Με το framework της openmp παραλληλοποιήθηκαν οι αναδρομές με τη χρήση tasks. Νέα tasks καλούνταν αν ο αριθμός των σημείων ήταν μεγαλύτερος από χίλια και αν η τιμή της *omp_get_active_level*() ήταν μικρότερη από πέντε. Γενικά η άσκοπη δημιουργία thread προσέδιδε αρκετό overhead.
Για τον υπολογισμό των αποστάσεων, και πάλι μετά από κάποιο αριθμό σημείων η υλοποίηση γινόταν σειριακά και όχι παράλληλα. Ωστόσο, η βελτίωση στο χρόνο που προσέδιδε ο παράλληλος υπολογισμός των αποστάσεων ήταν ελάχιστη.

Vptree_cilk

Το framework της cilk ήταν το πιο εύκολο στη χρήση. Χρησιμοποιήθηκαν οι συναρτήσεις του και δε χρειάστηκε να μπουν κάποια thresholds μετά από τα οποία η υλοποίηση θα γινόταν σειριακά. Πιο συγκεκριμένα χρησιμοποιήθηκαν η *cilk_for*, *cilk_spawn* και *cilk_sync*.

Χρόνοι Εκτέλεσης

Παρουσιάζονται οι χρόνοι εκτέλεσης των προγραμμάτων για κάποια πειράματα που εκτέλεσα ανάλογα με τον αριθμό των σημείων και τις διαστάσεις. Το μηχάνημα που εκτελέστηκαν είναι Dell Laptop με ubuntu 16.04, 8GB RAM και επεξεργαστή intel CORE i7. Επισημαίνεται ότι οι συγκεκριμένοι χρόνοι βασίζονται σε thresholds που έχω θέσει μέσα στο πρόγραμμά μου. Η αλλαγή αυτων μπορεί να έχει αρνητική αλλά και θετική επίδραση στους συνολικούς χρόνους εκτέλεσης.

	N=1000			
	D=10	D=100	D=1000	D=10000
Sequential	0,002737	0,014583	0,1892	2,430238
Pthreads	0,008475	0,011774	0,177296	2,242385
OpenMP	0,002081	0,017837	0,193036	2,351863
Cilk	0,001903	0,011916	0,162175	2,747673

	N=100000			
	D=10	D=100	D=500	D=800
Sequential	0,389872	3,00477	16,23894	31,38543
Pthreads	0,258005	1,466923	9,387026	16,608198
OpenMP	0,302782	1,801689	10,590587	FULL RAM
Cilk	0,211939	1,390416	9,452993	17,129158

	N=10000			
	D=10	D=100	D=1000	D=5000
Sequential	0,032181	0,223779	2,900766	17,692178
Pthreads	0,020505	0,130907	1,603458	12,441081
OpenMP	0,042142	0,172847	1,904318	13,206167
Cilk	0,018853	0,129587	1,550085	13,694845

	N=1000000			
	D=10	D=20	D=50	D=100
Sequential	5,042438	7,815386	17,104769	36,774393
Pthreads	2,568896	3,927705	8,604825	17,178726
OpenMP	3,166503	3,971647	10,599389	FULL RAM
Cilk	2,494666	3,947363	8,605225	16,862541

Παρατηρούμε οτι την καλύτερη επίδοση την είχε σχεδόν πάντα η Cilk. Επίσης για μικρα N και D ο χρόνοι εκτέλεσης είναι περίπου ίδιοι. Τα σημεία όπου εντοπίζονται οι μέγιστες διαφορές είναι όταν το N παίρνει πολύ μεγάλες τιμές. Επίσης, εντύπωση κάνει το γεγονός ότι για N=1000(σχετικά μικρό) και D=10000(πολλές διαστάσεις) οι χρόνοι εκτέλεσης όλων των προγραμμάτων είναι περίπου ίδιοι και οι παράλληλες υλοποιήσεις δεν προσφέρουν κάποια μείωση. Τέλος βλέπουμε ότι σε δύο περιπτώσεις το πρόγραμμα με την OpenMP εξάντλησε τη διαθέσιμη μνήμη RAM του υπολογιστή μου(8GB). Από τα αποτελέσματα καταλαβαίνουμε ότι θα μπορούσαν να έχουν σχεδιαστεί καλύτερα οι τρόποι υπολογισμού των αποστάσεων καθώς η σχετική διαφορά των χρόνων δεν αυξάνεται ανάλογα με την αύξηση των διαστάσεων D.