Finding the Strongly Connected Components of a directed graph in a shared memory multiprocessor

Τόλιας Φίλιππος ΑΕΜ:10252

Κώδικας: https://github.com/superkolios/parallel-programming-erg-1

Δομή δεδομένων

Η μορφή αποθήκευσης του του γράφου είναι ένας αραιός πίνακας στην μορφη CSC(Compressed Sparse Column). Αυτό έχει υλοποιηθεί σε κώδικα με ένα struct.

```
struct Csc{
    int nz;
    int n;
    int m;
    int* row_index;
    int* col_index;
    bool* valid_nodes;
    int remaining;
};
```

`nz`: non zero elements δηλαδή ο αριθμός των nodes `n`,`m`: μέγεθος πίνακα δηλαδή αριθμός των edges

`valid_nodes`: `True` αν το αντίστοιχο node δεν έχει μπει ακόμα σε κάποιο SCC(Strongly Connected Component) απο τον αλγόριθμο ή False αν έχει μπει. `remaining`: Ο αριθμός των nodes που δεν έχουν μπει ακόμα σε κάποιο SCC απο τον αλγόριθμο.

Τρόπος λειτουργίας του αλγόριθμου

Αρχικά το πρόγραμμα καλεί την συνάρτηση `int my_csc_mtx_to_csc(struct Csc *csc, char* file)` για να μετατρέψει το αρχείο `.mtx` (MatrixMarket) στην δομή CSC. Στη συνέχεια καλείται ο αλγόριθμος `int* my_coloring_scc_algorithm(struct Csc *csc)`, ο οποίος είναι υπεύθυνος για τον εντοπισμό των SCC. Επιστρέφει ένα array που δείχνει σε ποιο SCC ανήκει κάθε node. Στην αρχή

Στρατηγικη παραλληλοποίησης

Παρραλληλοποίησα τα for loops για τα όποια οι επαναληψεις δεν κάποια εξάρτηση μεταξύ τους και τα for loops για τα οποία η εξάρτηση μπορούσε να λυθεί με reducers. Επισης παρραλληλοποίησα και το BFS που είχε εξαρτηση λόγο του array `to_visit` μαζί με δεικτη θεσης τα οποια λειτουργουν ως ενα FILO(First In Last Out) queue στο οποίο ειναι πολυ πιθανο να γινει κάποιο data race. Για να το αντιμετοποισω αυτό το έβαλα σε ενα critical section. Επειδή τα critical sections καθυστερούν την ρόη του προγράμματος εκτελείται μονο οταν είναι απολύτως απαραίτητο, για αυτό και η τιμη visted[i] ελέγχεται και πριν μπει critical section.

```
if (colors[node] == c && !visited[node]){
    #pragma omp critical
    {
        if (!visited[node]){
            to_visit[to_visit_prt] = node;
            to_visit_prt++;
            visited[node] = true;
            (*size)++;
            csc->valid_nodes[node] = false;
        }
    }
}
```

Επίσης ενα σημαντικό κομμάτι που μειώνει συμπαντικά τον χρονο εκτέλεσης (και στο παράλληλο και στο σειριακό) είναι η μείωση των αριθμων που γινονται memory allocations.

Αποδοση αλγορίθμου

Graph	n	m	ncc	serial	openMp
Newman/celegansneural	297	2345	57		
Pajek/foldoc	13356	120238	71	0.003121s	0.021194s
Tromble/language	399130	1216334	2456	0.097528s	0.596788s
LAW/eu-2005	862664	19235140	90768	0.612506s	1.498135s
SNAP/wiki-topcats	1791489	28511807	1	7.928697s	11.442782s
SNAP/sx-stackoverflow	2601977	36233450	953658	2.295690s	4.845562s
Gleich/wikipedia-20060925	2983494	37269096	975731	23.086584s	23.838741s
Gleich/wikipedia-20061104	3148440	39383235	1040035		
Gleich/wikipedia-20070206	3566907	45030389	1203340	29.036076s	25.950829s
Gleich/wb-edu	9845725	57156537	4269022		
LAW/indochina-2004	7414866	194109311	1749052		
LAW/uk-2002	18520486	298113762	3887634		
LAW/arabic-2005	22744080	639999458	4000414		
LAW/uk-2005	39459925	936364282	5811041		
SNAP/twitter7	41652230	1468365182	8044728		

Αρχικά πίστευα ότι ο κώδικας σε OpenMp ήταν πολύ πιο γρήγορος άπο τον σειριακό αλλά στο τέλος

παρατήρησα οτι στον σειριακό δεν είχα κάνει μια βελτίωση που μειώνει τον αριθμό τον allocations.