Συστήματα Παράλληλης Επεξεργασίας

Μοίρας Αλέξανδρος
Α.Μ.: 03118081

1η Εργαστηριακή Άσκηση

Conway's Game of Life:

Προκειμένου να παραλληλοποιήσουμε το Game of Life, στο μοντέλο κοινού χώρου διευθύνσεων, με τη χρήση του OpenMP, καλούμαστε να παραλληλοποιήσουμε τμήμα του υπολογιστικού μέρους του αλγορίθμου το οποίο με βάση την προηγούμενη εποχή υπολογίζει την τρέχουσα. Ο υπολογισμός (στον κώδικα που μας δίνεται) δύναται να παραλληλοποιηθεί τόσο κατά γραμμές όσο και κατά στοιχείο και όχι στον χρόνο αφού ο υπολογισμός κάθε κελιού εξαρτάται μόνο από την προηγούμενη κατάσταση και όχι από τα τρέχοντα γειτονικά κελιά. Με μία πρώτη ματιά η καταλληλότερη παραλληλοποίηση που μπορούμε να εφαρμόσουμε είναι κατά γραμμές του board (i loop), καθώς έτσι αποφεύγουμε περιπτώσεις false sharing (υποθέτοντας λογικά ότι οι πίνακες είναι αποθηκευμένοι κατά γραμμές, και συνεπώς πιθανότατα διαφορετικές γραμμές ανήκουν σε διαφορετικά blocks), οι οποίες μπορεί να εμφανιζόντουσαν σε μια παραλληλοποίηση κατά στοιχείο (j loop), και θα καθυστερούσαν το πρόγραμμα λόγω coherence misses. Επίσης μια παραλληλοποίηση κατά στοιχείο (j loop) θα εισήγαγε πολύ μεγάλο overhead για την αξία της (ένα κελί ανά thread τη φορά).
Για να εφαρμόσουμε τα προαναφερθέντα πρέπει να κάνουμε include τη βιβλιοθήκη omp.h, και να προσθέσουμε την παρακάτω γραμμή κώδικα πάνω από το for loop που εξετάζει τις γραμμές του πίνακα:

```
for ( t = 0 ; t < T ; t++ ) {
        #pragma omp parallel for shared(previous, current, N) private (i, j, nbrs)
        for ( i = 1 ; i < N-1 ; i++ )
                for ( j = 1 ; j < N-1 ; j++ ) {
                        nbrs = previous[i+1][j+1] + previous[i+1][j] + previous[i+1][j-1] \
                               + previous[i][j-1] + previous[i][j+1] \
                               + previous[i-1][j-1] + previous[i-1][j] + previous[i-1][j+1];
                        if ( nbrs == 3 || ( previous[i][j]+nbrs ==3 ) )
                                current[i][j]=1;
                        else
                                current[i][j]=0;
                }
```

Με τη συγκεκριμένη εντολή καλούμε το OpenMP να μοιράσει τις εκτελέσεις του i loop μεταξύ ενός συγκεκριμένου αριθμού threads ο οποίος θα καθορίζεται δυναμικά από την environment variable OMP_NUM_THREADS. Επίσης καθορίζουμε ποιες μεταβλητές θα είναι shared μεταξύ των threads (δηλαδή θα έχουν όλα κοινή πρόσβαση σε αυτές μέσω κοινής μνήμης), και ποιες θα είναι private δηλαδή κάθε thread θα έχει δικό του αντίγραφο αυτών. Οι πίνακες previous και current πρέπει προφανώς να βρίσκονται σε κοινή μνήμη, καθώς δε συντρέχει λόγος για αντιγραφή τόσο μεγάλων πινάκων ανά thread αφού ο previous είναι read-only και στον current θέλουμε να γράφουν όλοι, κάτι το οποίο γίνεται ταυτόχρονα αφού δεν επικρατεί κάποιο race condition (κάθε thread γράφει μόνο τη δική του γραμμή). Τα i, j θέλουμε να είναι private καθώς αποτελούν loop indices τα οποία πρέπει να αυξομειώνονται ανεξάρτητα από κάθε thread. Η μεταβλητή nbrs οφείλει να είναι και αυτή private καθώς αποτελεί ενδιάμεση μεταβλητή για τον υπολογισμό κάθε κελιού, και προφανώς ο υπολογισμός ενός κελιού από ένα thread δε θα έπρεπε να επηρεάζει τον υπολογισμό ενός άλλου κελιού από άλλο thread καθώς κάτι τέτοιο θα χαλούσε την ορθότητα του προγράμματος. Ακόμα όμως και αν δεν υπήρχε το πρόβλημα της ορθότητας, το να ήταν μοιραζόμενη η μεταβλητή nbrs θα οδηγούσε σε αύξηση του χρόνου εκτέλεσης με αύξηση των threads καθώς όλα τα threads θα οδηγούνταν σε coherence miss σε κάθε πρόσβασή τους σε αυτήν, ενώ τα bus transactions που αυτά θα προκαλούσαν θα δημιουργούσαν επιπλέον καθυστερήσεις στον δίαυλο λόγω ανταγωνισμού, ειδικά για μεγάλο αριθμό threads. Πράγματι σε δοκιμές με τη μεταβλητή nbrs ως shared ο χρόνος εκτέλεσης αυξανόταν δραματικά με αύξηση των threads.

By default οι πίνακες previous, current είναι shared καθώς ορίζονται εκτός του παραλλήλου block , ενώ τα loop iteration variables είναι private, οπότε σε αυτά δε χρειαζόταν να επέμβουμε. Όμως η μεταβλητή nbrs ορίζεται εκτός του παραλλήλου block και by default είναι shared οπότε υποχρεούμαστε να τη δηλώσουμε ρητά ως private.
Παρακάτω παρουσιάζονται τόσο σε μορφή πίνακα όσο και σε γράφημα τα αποτελέσματα των μετρήσεων για 1,2,4,6,8 πυρήνες και μεγέθη ταμπλό 64x64, 1024x1024, 4096x4096 για 1000 γενιές κάθε φορά.

64x64:

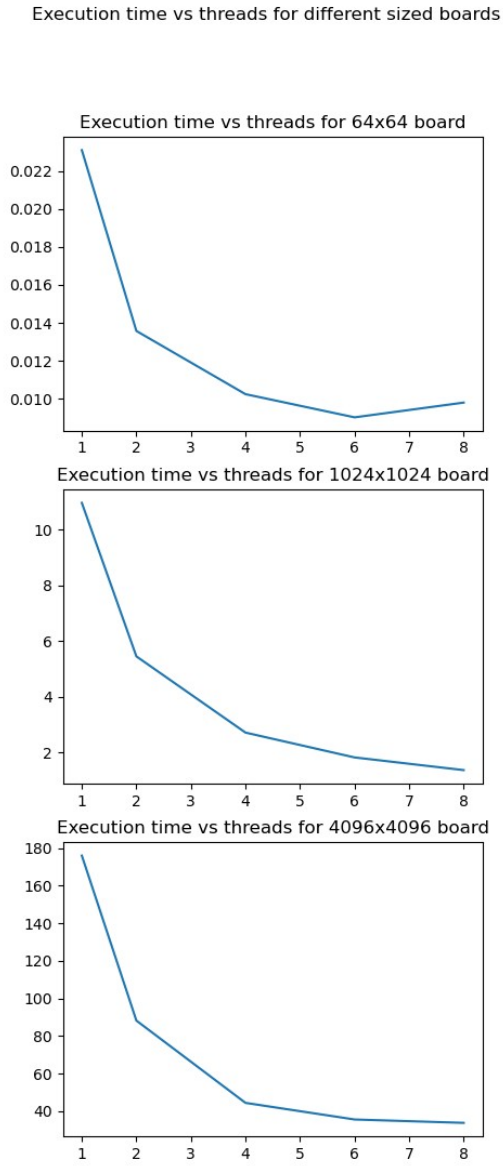
#threads	1	2	4	6	8
Time (s)	0.023095	0.013575	0.010248	0.009027	0.009804

1024x1024:

#threads	1	2	4	6	8
Time (s)	10.968725	5.458936	2.723176	1.831348	1.376639

4096x4096:

#threads	1	2	4	6	8
Time (s)	175.926421	88.220340	44.480710	35.644253	33.856838



Παρατηρούμε ότι σε κάθε περίπτωση το task μπορεί να επωφεληθεί από παραλληλία έως έναν βαθμό κάτι που είναι αναμενόμενο καθώς όπως εξηγήθηκε παραπάνω υπάρχει εκμεταλλεύσιμη παραλληλία στο Game of Life. Για όλα τα μεγέθη ταμπλό αύξηση των threads από 1 σε 2 πρακτικά υποδιπλασιάζει τον χρόνο εκτέλεσης και πρόκειται για τέλεια κλιμάκωση.

Για το μικρότερο ταμπλό (64x64) παρατηρούμε diminishing returns από εκείνο το σημείο και έπειτα καθώς το τμήμα του προγράμματος που μπορεί να παραλληλοποιηθεί δεν είναι αρκετά μεγάλο (μόλις 64 γραμμές), τα σειριακά τμήματα είναι πιο καθοριστικά για την επίδοση και το overhead που εισάγει η παραλληλοποίηση δεν μπορεί να δικαιολογήσει το κόστος του. Μάλιστα για 8 threads το overhead αυτό όχι μόνο μειώνει τη βελτίωση στην επίδοση αλλά την χειροτερεύει.

Για το μεσαίο μέγεθος (1024x1024) ταμπλό παρατηρούμε γραμμική κλιμάκωση με αύξηση των threads. Δηλαδή το πρόγραμμα είναι σε θέση να εκμεταλλευτεί στο έπακρο την παραλληλία του και όλα τα threads μπορούν να επεξεργάζονται δεδομένα ταυτόχρονα χωρίς να καθυστερούν.

Για μεγαλύτερα ταμπλό (4096x4096 στο πείραμά μας) παρατηρούμε γραμμική κλιμάκωση μέχρι τα 4 threads ενώ από εκεί και πέρα έχουμε μικρότερο speedup. Αυτό συμβαίνει λόγω συμφόρησης στο διάδρομο μνήμης. Έχουμε πολλά threads που καλούνται να εκτελέσουν κάποιους απλούς υπολογισμούς ταυτόχρονα, για πολύ μεγάλο όγκο δεδομένων το καθένα (κάθε γραμμή του πίνακα περιέχει 4096 στοιχεία). Με αυτόν τον τρόπο η προς εκτέλεση εργασία γίνεται memory bound και περιορίζεται από το throughput του διαδρόμου της κοινής μνήμης με τις υπολογιστικές μονάδες να περιμένουν δεδομένα από τη μνήμη χωρίς να εκτελούν χρήσιμες εντολές ανά διαστήματα, επομένως χάνεται η γραμμική μείωση του χρόνου εκτέλεσης με την αύξηση των threads.