ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ροή Υ : Συστήματα Παράλληλης Επεξεργασίας

1η Άσκηση

Εισαγωγή στην Παραλληλοποίηση σε Αρχιτεκτονικές Κοινής Μνήμης

Κριθαρούλας Διονύσιος 03117875

Εξάμηνο: 9ο

Νοέμβριος 2021

Εισαγωγή και Περιγραφή Προβλήματος

Στην συγκεκριμένη εργαστηριακή άσκηση υλοποιήσαμε την παραλληλοποίηση του προβλήματος Conway's Game Of Life.

Το πρόβλημα αυτό λαμβάνει χώρα σε ένα ταμπλό με κελιά δύο διαστάσεων. Το περιεχόμενο κάθε κελιού μπορεί να είναι γεμάτο (alive) ή κενό (dead), αντικατοπτρίζοντας την ύπαρξη ή όχι ζωντανού οργανισμού σε αυτό, και μπορεί να μεταβεί από τη μία κατάσταση στην άλλη μία φορά εντός συγκεκριμένου χρονικού διαστήματος. Σε κάθε βήμα (χρονικό διάστημα), κάθε κελί εξετάζει την κατάστασή του και αυτή των γειτόνων του (δεξιά, αριστερά, πάνω, κάτω και διαγώνια) και ακολουθεί τους παρακάτω κανόνες για να ενημερώσει την κατάστασή του:

- Αν ένα κελί είναι ζωντανό και έχει λιγότερους από 2 γείτονες πεθαίνει από μοναξιά.
- Αν ένα κελί είναι ζωντανό και έχει περισσότερους από 3 γείτονες πεθαίνει λόγω υπερπληθυσμού.
- Αν ένα κελί είναι ζωντανό και έχει 2 ή 3 γείτονες επιβιώνει μέχρι την επόμενη γενιά.
- Αν ένα κελί είναι νεκρό και έχει ακριβώς 3 γείτονες γίνεται ζωντανό (λόγω αναπαραγωγής).

Παραλληλοποίηση του προβλήματος

Εξετάζοντας το πρόβλημα καταλαβαίνουμε ότι προκειμένου σε κάποιο χρονικό βήμα να υπολογίσουμε την τιμή ενός κελιού χρειαζόμαστε τόσο την δικιά του όσο και τις τιμές των γειτόνων του που έχουν υπολογισθεί στο προηγούμενο χρονικό βήμα (συνολικά 9 τιμές). Η διαπίστωση αυτή μας δείχνει ότι στο συγκεκριμένο πρόβλημα δεν μπορεί να πραγματοποιηθεί καμία παραλληλοποίηση όσον αφορά τα χρονικά βήματα καθώς κάθε επόμενο χρειάζεται τις τιμές του προηγούμενου του. Για τον λόγο αυτό και ο κώδικας που είναι υλοποιημένος κρατάει δύο πίνακες , τον current που περιέχει τις τιμές των κελιών που υπολογίζουμε στο εκάστοτε χρονικά βήμα και τον previous που περιέχει τις τιμές των κελιών που υπολογίσαμε στο ακριβώς προηγούμενο χρονικό βήμα. Έτσι κάθε φορά που είναι να μεταβούμε στο επόμενο χρονικό βήμα μεταφέρουμε τα αποτελέσματα του current πίνακα στον previous ώστε να τα χρησιμοποιήσουμε στον επόμενο υπολογισμό.

Όμως μέσα σε κάθε χρονικό βήμα κάθε νέα τιμή ενός κελιού του ταμπλό μπορεί να υλοποιηθεί ανεξάρτητα καθώς εξαρτάται μόνο από τιμές των κελιών του προηγούμενου χρονικού βήματος όπως αναφέραμε. Για τον λόγο αυτό μέσα σε ένα συγκεκριμένο χρονικό βήμα το πρόβλημα μπορεί να παραλληλοποιηθεί χωρίς κανένα πρόβλημα.

Υλοποίηση της παραλληλοποίησης

Η κύρια επανάληψη της σειριακής υλοποίησης η οποία μας δίνεται κα η οποία κάθε χρονική στιγμή υπολογίζει κάθε ένα από τα κελιά του ταμπλό είναι η ακόλουθη:

```
for ( t = 0 ; t < T ; t++ ) {
        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;
                }
        }
```

Παρατηρούμε ότι το πιο εξωτερικό loop τρέχει πάνω στην χρονική στμή αυξάνοντας την κάθε φορά κατά ένα βήμα. Αντίθετα τα δύο εσωτερικά loop υπολογίζουν κατά γραμμή την νέα τιμή κάθε κελιού με βάση τους κανόνες του προβλήματος τους οποίους παραθέσαμε στην αρχή της αναφοράς.

Προκειμένου λοιπόν το πρόβλημα να παραλληλοποιηθεί προστέθηκε στο παραπάνω σειριακό πρόγραμμα η ακόλουθη γραμμή κώδικα:

```
#pragma omp parallel for shared(N,previous,current) private(i,j,nbrs)
```

Η εντολή αυτή προστίθεται μέσα από το πρώτο εξωτερικό loop καθώς όπως αναφέραμε δεν γίνεται να υπάρξει κάποια παραλληλία μεταξύ των χρονικών στιγμών. Αντίθετα προστίθεται πάνω ακριβώς από τα δύο επόμενα loop ώστε ο υπολογισμός στο εσωτερικό τους να γίνει παράλληλα. Επιπλέον ορίζουμε η μεταβλητή N που δείχνει τις διαστάσεις του ταμπλό αλλά και οι πίνακες current και previous τους οποίους την σημασία εξηγήσαμε προηγουμένως να είναι κοινές και να μοιράζονται από τα νήματα. Αντίθετα οι μεταβλητές i,j και nbrs να είναι τοπικές για κάθε νήμα.

Έτσι η κύρια επανάληψη της σειριακής υλοποίησης αποκτά πλέον την ακόλουθη μορφή:

```
for ( t = 0 ; t < T ; t++ ) {
        #pragma omp parallel for shared(N,previous,current) 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;
                }
        }
```

Ο αριθμός των νημάτων που θα παραλληλοποιήσουν την εκτέλεση του συγκεκριμένου διπλού βρόγχου ορίζεται από την μεταβλητή περιβάλλοντος OMP_NUM_THREADS . Την μεταβλητή αυτή την ορίσαμε να παίρνει επαναληπτικά τις τιμές 1,2,4,6,8 ώστε να δούμε τα αποτελέσματα της παραλληλοποίησης με διαφορετικό αριθμό νημάτων. Επιπλέον για κάθε έναν από τους παραπάνω αριθμούς νημάτων τρέξαμε το πρόβλημα έχοντας ως δεδομένα ένα ταμπλό NxN όπου το N έπαιρνε τιμές N=64,1024,4096. Το χρονικό βήμα μέχρι το οποίο θα συνεχίσει τους υπολογισμούς του ο αλγόριθμος έμεινε σταθερό για όλες τις περιπτώσεις και ίσο με 1000 βήματα. Οι παραπάνω περιπτώσεις υλοποιήθηκαν στο αρχείο run_on_queue.sh και η υλοποίηση τους είναι η ακόλουθη:

```
for thr in 1 2 4 6 8
do
        echo "Compute with $thr cores"
        export OMP_NUM_THREADS=$thr
        ./omp_game_of_life 64 1000
        ./omp_game_of_life 1024 1000
        ./omp_game_of_life 4096 1000
done
```

Μετρήσεις και Σχολιασμοί

Μετρώντας για κάθε περίπτωση τον χρόνο τον οποίο χρειάστηκε το πρόγραμμα προκειμένου να ολοκληρωθεί πήραμε τα ακόλουθα αποτελέσματα:

```
Compute with 1 cores
GameOfLife: Size 64 Steps 1000 Time 0.023136
GameOfLife: Size 1024 Steps 1000 Time 10.965857
GameOfLife: Size 4096 Steps 1000 Time 175.857121
Compute with 2 cores
GameOfLife: Size 64 Steps 1000 Time 0.013523
GameOfLife: Size 1024 Steps 1000 Time 5.454860
GameOfLife: Size 4096 Steps 1000 Time 88.227496
Compute with 4 cores
GameOfLife: Size 64 Steps 1000 Time 0.010266
GameOfLife: Size 1024 Steps 1000 Time 2.723260
GameOfLife: Size 4096 Steps 1000 Time 44.538055
Compute with 6 cores
GameOfLife: Size 64 Steps 1000 Time 0.009478
GameOfLife: Size 1024 Steps 1000 Time 1.829089
GameOfLife: Size 4096 Steps 1000 Time 37.658220
Compute with 8 cores
GameOfLife: Size 64 Steps 1000 Time 0.009370
GameOfLife: Size 1024 Steps 1000 Time 1.378009
GameOfLife: Size 4096 Steps 1000 Time 36.394241
```

Τα αποτελέσματα αυτά παρατίθενται και στον ακόλουθο πίνακα:

Αριθμός Νημάτων-Διαστάσεις ταμπλό	64	1024	4096
1	0.023136	10.965857	175.857121
2	0.013523	5.454860	88.227496
4	0.010266	2.723260	44.538055
6	0.009478	1.829089	37.658220
8	0.009370	1.378009	36.394241

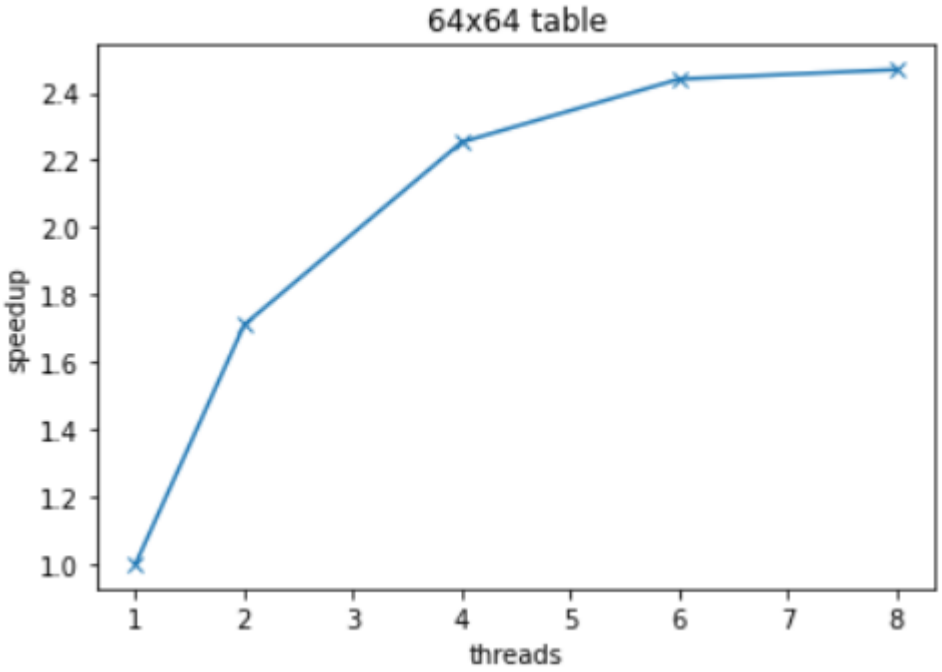
Σχολιασμοί

Παρατηρούμε ότι στην περίπτωση όπου το ταμπλό έχει μέγεθος 64x64 ο χρόνος εκτέλεσης μειώνεται λίγο συγκριτικά με την αύξηση των νημάτων. Κάτι τέτοιο οφείλεται στο γεγονός πως το μέγεθος του ταμπλό είναι αρκετά μικρό και επομένως η παραλληλία σε επίπεδο πράξεων που μπορεί να πραγματοποιηθεί δεν κάνει κάποια πολύ αισθητή διαφορά σε σχέση με την περίπτωση όπου το πρόγραμμα έτρεχε σειριακά. Αντίστοιχα οι χρόνοι που τα νήματα σπαταλούν προκειμένου να έχουν πρόσβαση στα κοινά δεδομένα (έχουμε αρχιτεκτονική κοινής μνήμης) δεν μπορούν να αμεληθούν σε αυτήν την περίπτωση όπου τα δεδομένα δεν είναι τόσα πολλά (μικρό ταμπλό) και η δυνατότητα παραλληλίας περιορισμένη.

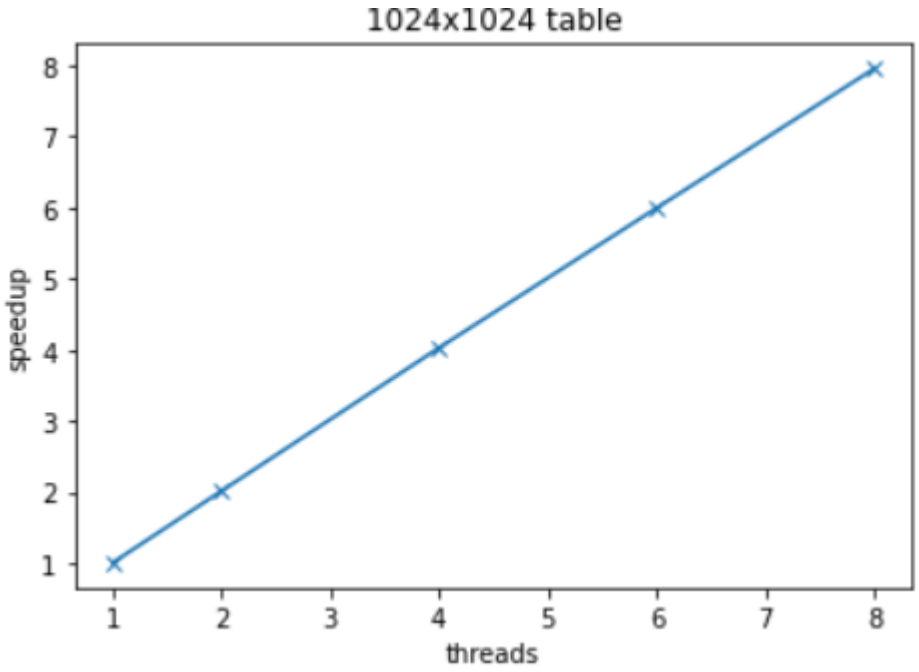
Αντίθετα στην περίπτωση όπου το ταμπλό έχει μέγεθος 1024x1024 παρατηρούμε μεγαλύτερο ποσοστό μείωσης στον χρόνο εκτέλεσης καθώς αυξάνουμε τον αριθμό των νημάτων. Κάτι τέτοιο είναι λογικό καθώς πλέον υπάρχει περισσότερη δυνατότητα παραλληλοποίησης επομένως ο χρόνος που αναφέραμε προηγουμένως της πρόσβασης σε κοινά δεδομένα δεν επιδρά τόσο αρνητικά στην μείωση του χρόνου εκτέλεσης.

Τέλος στην περίπτωση όπου το ταμπλό έχει μέγεθος 4096x4096 έχουμε μια αρκετά μεγάλη μείωση καθώς αυξάνουμε τους πυρήνες γεγονός που έρχεται σε συμφωνία με την παραπάνω μας άποψη ότι η αύξηση των διαστάσεων δίνει μεγαλύτερη δυνατότητα παραλληλίας και περισσότερο χώρο ώστε τα νήματα να μην χρειάζονται πολλές φορές να έχουν πρόσβαση σε κοινά δεδομένα. Βέβαια σε αυτήν την περίπτωση παρατηρούμε ότι πηγαίνοντας από τα 6 νήματα στα 8 η μείωση του χρόνου είναι πολύ μικρή σε σχέση με τις μειώσεις όταν π.χ. πηγαίναμε από το 1 νήμα στα 2 ή από τα 2 στα 4. Κάτι τέτοιο μας δείχνει ότι η αύξηση των νημάτων κατά 2 δεν είναι απαραίτητο ότι θα σημαίνει πάντα και την αντιστοιχεί μείωση του χρόνου εκτέλεσης. Ίσως η μεγάλη αύξηση των διαστάσεων δημιουργεί στο κάθε νήμα μεγαλύτερο χρόνο προκειμένου να λάβει πρόσβαση σε κοινά δεδομένα αφού δημιουργούνται περισσότεροι γείτονες και περισσότερα δεδομένα παύουν πλέον να είναι στην cache του κάθε πυρήνα.

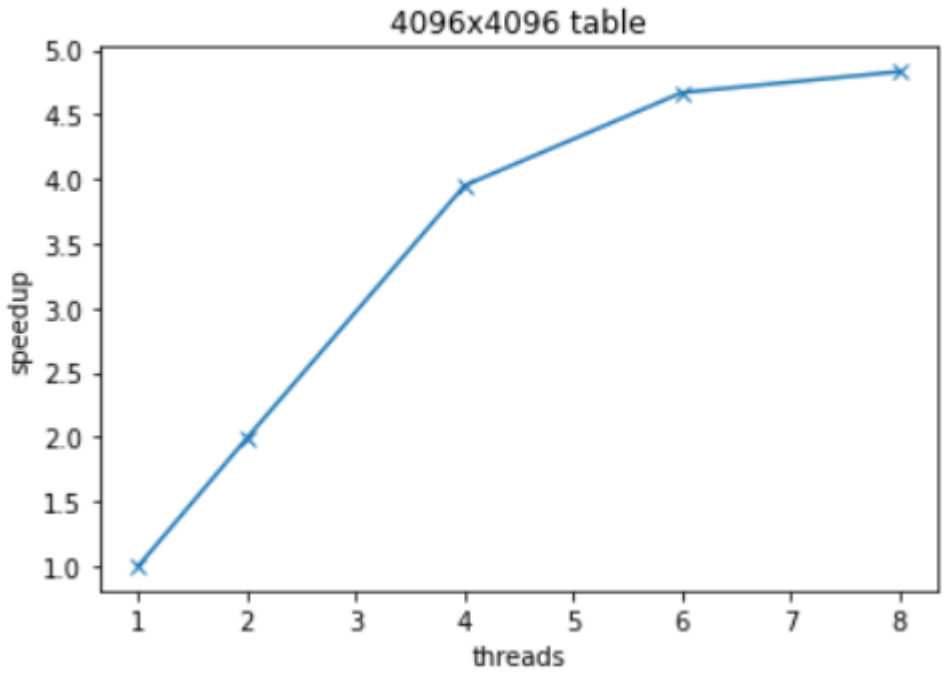
Στην συνέχεια παραθέτουμε και τρία διαγράμματα τα οποία για κάθε μέγεθος ταμπλό παραθέτουν την μεταβολή του speedup καθώς αυξάνουμε τον αριθμό των πυρήνων. Ουσιαστικά με speedup ορίζουμε τον λόγο του χρόνου εκτέλεσης του σειριακού προγράμματος προς το παράλληλο. Η τιμή αυτή θέλουμε να αυξάνεται καθώς αυξάνουμε τον αριθμό των πυρήνων (το πρόγραμμα μας άρα να γίνεται πιο γρήγορο) και μάλιστα ακόμα καλύτερα η αύξηση της αυτή να είναι ανάλογη της αύξησης του αριθμού των πυρήνων που χρησιμοποιούμε. Έτσι έχουμε:



Όπως αναλύσαμε και προηγουμένως παρατηρούμε ότι η αύξηση του αριθμού των νημάτων δεν είναι ανάλογη με την μείωση του χρόνου εκτέλεσης (αύξηση του speedup) ειδικά καθώς μεταβαίνουμε σε μεγαλύτερο αριθμό νημάτων για μέγεθος ταμπλό 64x64.



Όπως αναλύσαμε και προηγουμένως για διαστάσεις ταμπλό 1024x1024 παρατηρούμε ότι η αύξηση του αριθμού των νημάτων είναι ανάλογη της μείωσης του χρόνου εκτέλεσης (διπλασιασμός νημάτων προκαλεί υποδιπλασιασμό του χρόνου εκτέλεσης).



Όπως αναφέραμε και προηγουμένως στην περίπτωση που το μέγεθος του ταμπλό είναι 4096x4096 παρατηρούμε ότι η αύξηση των πυρήνων μέχρι το 4 προκαλεί ανάλογη μείωση του χρόνου εκτέλεσης. Αντίθετα περαιτέρω διπλασιασμός των πυρήνων δεν προκαλεί αντίστοιχο υποδιπλασιασμό του χρόνου εκτέλεσης.

Σημείωση: Η υλοποίηση των παρακάτω τριών διαγραμμάτων έγινε με την βοήθεια της python και με βάση τις τιμές του πίνακα των μετρήσεων που παραθέσαμε παραπάνω.