Συστήματα Παράλληλης Επεξεργασίας

Εργαστηριακή αναφορά 3

Αμοιβαίος Αποκλεισμός - Κλειδώματα

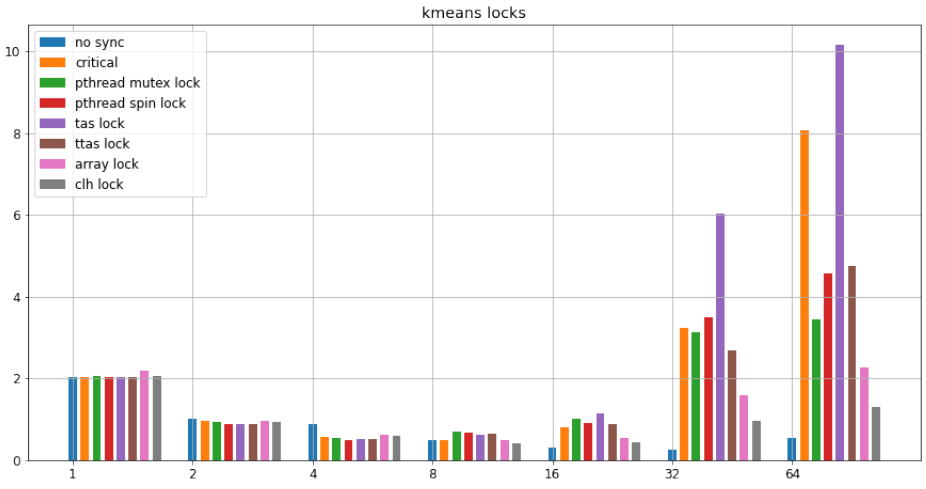
Γιάννος Παράνομος - 03118021
Νικήτας Τσίννας - 03118187
Καούκης Γιώργος - 03119006

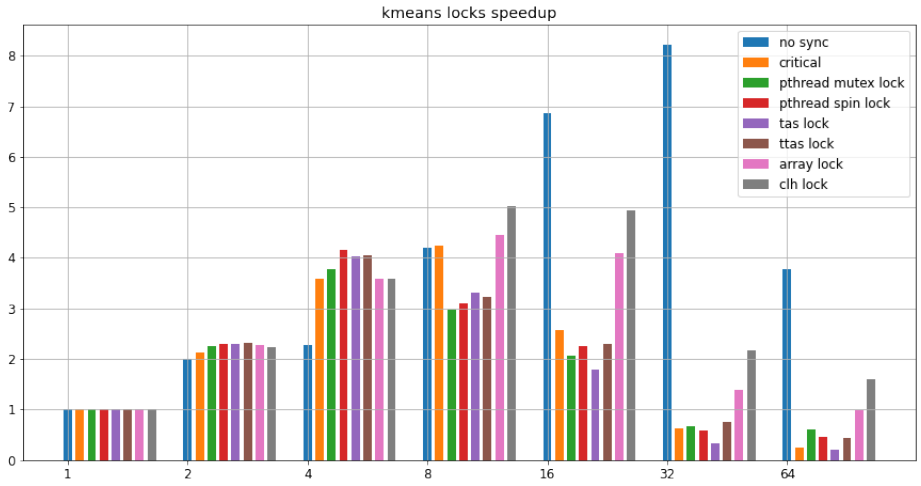
Δεκέμβριος 2022

1 Εισαγωγή

Σε αυτή την εργαστηριακή άσκηση, εξετάζουμε τους χρόνους εκτέλεσης διαφόρων κλειδωμάτων, τους συγκρίνουμε με αυτούς που εμφανίζει η υλοποίηση με `OpenMP Critical` και ερμηνεύουμε τα αποτελέσματα που παίρνουμε.

2 Παρουσίαση Αποτελεσμάτων





Αγνωόντας την επίδοση που επιτυγχάνουν τα `no sync locks`, τα οποία δεν επιτελούν λειτουργία συγχρνισμού και άρα παράγουν λανθασμένα αποτελέσματα, παρατηρούμε ότι την βέλτιστη επίδοση εμφανίζουν τα `clh locks`, τα οποία αποτελούν κλειδώματα που στηρίζονται στη χρήση συνδεδεμένης λίστας.
Σημειώνουμε ότι λόγω των παραμέτρων που έχουμε βάλει
(`{Size, Coords, Clusters, Loops} = {16, 16, 16, 10}`), είναι αναμενόμενο το scalability να "σπάει" για 32 νήματα και πάνω, καθώς πρακτικά δεν απαιτούνται παραπάνω πόροι (έχουμε 16 στοιχεία που μπορούμε να παραλληλοποιήσουμε), παρόλαυτά τα αποτελέσματα που παρατηρούμε είναι χρήσιμα για να κατανοήσουμε τη συμπεριφορά των κλειδωμάτων μας. Ειδικότερα, αυτό σημαίνει ότι οι δύο τελευταίες μετρήσεις (32 και 64 νήματα) μας επιτρέπουν να εστιάσουμε μόνο στις καθυστερήσεις που εισαγάγει το κάθε κλείδωμα!

3 Σχολιασμός Κλειδωμάτων - Υλοποιήσεων

OpenMP Critical
Η δομή αυτή, επιτρέπει σε ένα thread κάθε φορά να εισέλθει στο `critical section`. Η λειτουργία της είναι αντίστοιχη με την χρήση locks, αλλά η υλοποίηση είναι πολύ απλούστερη, καθώς πολλά low-level tasks γίνονται από το OpenMP. Για τον λόγο αυτό περιμένουμε αντίστοιχη απόδοση με την υλοποίησης `pthread mutex lock`, με κάποιες μικρές καθυστερήσεις.

nosync lock
Η δομή αυτή, δεν αποτελεί κάποιο κλείδωμα, αλλά τη χρησιμοποιούμε για να δούμε πως θα λειτουργούσε ιδανκά το πρόγραμμά μας, αν η παραλληλοποίηση ήταν τέλεια. Φυσικά δεν θα πλησιάσουμε την επίδοση που εμφανίζεται με αυτή τη δομή, καθώς οποιαδήποτε άλλη δομή εισαγάγει καθυστερήσεις αφού επιβάλει περιορισμούς στην προσπέλαση δεδομένων. Τονίζεται ότι παρά την καλή χρονική επίδοση, τα αποτελέσματα που προκύπτουν δεν είναι ορθά.

pthread mutex lock

Η δομή αυτή, κάνει χρήση του `pthread mutex lock`, της βιβλιοθήκης Pthread. Αποτελεί ουσιαστικά τα γνωστά μας mutex locks, στα οποία ένα thread θα πάρει το lock και έτσι θα έχει αποκλειστική πρόσβαση στο κρίσιμο τμήμα. Παρατηρούμε ότι η επίδοση της δομής αυτής είναι κοντά στου critical, με εμφανώς καλύτερα αποτελέσματα σε 32 και 64 threads, τα οποία βέβαια δεν έχει νόημα να αξιολογήσουμε ιδιαίτερα, καθώς σε αυτά παρατηρείται έντονο scalability break.

pthread spin lock

Η δομή αυτή, μοιάζει πολύ με την παραπάνω όσον αφορά στην επίδοση. Η διαφορά των `spin locks` με τα `mutex locks`, έγκειται στο ότι τα πρώτα θα αναγκάσουν το thread που θα βρει το κρίσιμο τμήμα κλειδωμένο, να παραμείνει στην ενεργό αναμονή μέχρι να μπορέσει να μπεί στο κρίσιμο τμήμα. Αντίθετα ένα thread που συναντά κλειδωμένο `mutex lock`, μπορεί να πάει για ύπνο και άρα να αξιοποιηθεί και από κάποιο άλλο κομμάτι του κώδικα/ άλλο πρόγραμμα. Το ζήτημα είναι ότι η διαδικασία ύπνου-ξυπνήματος μιας διεργασίας, είναι πολύ χρονοβόρα, ενώ μια επίμονη διεργασία (spin lock), καταναλώνει πολλούς πόρους. Στην παρούσα εφαρμογή, δεν φαίνεται να υπάρχουν σημαντικα΄πλεονεκτήματα στη χρήση της μιας υλοποίησης με την άλλη. Αυτό μάλλον σημαίνει ότι ο χρόνος αναμονής στα `spin locks` είναι αντίστοιχος με τον χρόνο ύπνου-ξυπνήματος στα `mutex locks`.

tas lock

Η δομή αυτή, ονομάζεται `test-and-set lock` και αποτελεί ουσιαστικά την πιο απλή υλοποίηση της δομής `spin lock`. Διατηρεί μια κοινή δυαδική μεταβλητή, η πρόσβαση στην οποίο είναι ατομική και επιστρέφει True, αν το thread επιτρέπεται να εισέλθει στο κρίσιμο τμήμα, ενώ False αν δεν επιτρέπεται. Η επίδοσή της μέχρι τα 16 threads είναι αρκετά κοντινή με αυτή των `spin lock` και `mutex lock`. Σημειώνεται ότι σαν δομή παρόλαυτά, έχει πολύ κακό scalability, καθώς δεν διατηρεί κάποιου είδους ουρά προτεραιότητας και όπως παρατηρούμε, αυτό έχει καταστροφικά αποτελέσματα για πολλά threads.

ttas lock

Η απλούστερη υλοποίηση της δομής αυτής, επιτυγχάνει καλύτερη απόδοση καθώς επιπλέον των κινήσεων που γίνονται στο `tas`, εδώ έχουμε και ένα στάδιο ελέγχου της περίπτωσης που το κλείδωμα είναι ελεύθερο, πριν εφαρμόσει το `exchange()` του `tas`. Το παραπάνω έχει ως αποτέλεσμα λιγότερα cache invalidations τα οποία επιβραδύνουν σημαντικά την επίδοση του προγράμματος. Η απόδοση βελτιώνεται ακόμα περισσότερο αν γίνει χρήση `exponential backoff`, το οποίο βοηθάει όταν υπάρχει μεγάλη συμφόρηση σε ένα κλείδωμα. Η επίδοση έως τα 16 thread είναι αντίστοιχη με τα προηγούμενα, παρατηρούμε όμως ότι καθώς αυτά αυξάνονται, έχει πολύ καλύτερη συμπεριφορά σε σχέση με το `tas lock`.

array lock

Η δομή `array lock`, αποτελεί την πρώτη προσπάθεια πραγματικά κλιμακώσιμων κλειδωμάτων. Ουσιαστικά προσπαθεί να επιλύσει το πρόβλημα που εμφάνιζαν τα `spin locks`, αναφορικά στον συνοστισμό των thread σε μια κοινή (shared) μεταβλητή. Αυτό επιλύεται με τη δημιουργία ενός boolean array, μεγέθους ίσου με το πλήθος των thread, όπου το κάθε thread θα έχει μια αντίστοιχη θέση στον πίνακα, με τιμή True αν και μόνον αν βρίσκεται στο Critical Section. Σημειώνεται ότι δίνεται έτσι η δυνατότητα διατήρησης μιας προτεραιότητας στην κατάλλειψη

του κρίσιμου τμήματος (αυτή του First Come First Serve, καθώς όταν γίνεται unlock, το επόμενο bit στον πίνακα τίθεται σε True), άρα δεν έχουμε race conditions για την κατάλλειψη του κρίσιμου τμήματος.

clh lock

Η δομή `clh lock`, βρίσκεται στην ίδια κατηγορία με την παραπάνω `array lock`, λύνει δηλαδή τα ίδια προβλήματα που περιγράφηκαν παραπάνω όμως τώρα η δομή είναι πολύ πιο φιλική στη μνήμη `O(L+n)`, σε αντίθεση με `O(Ln)`, καθώς δεν απαιτείται στατικός πίνακας (array), αλλά γίνεται υλοποίηση με queue. Αυτό έχει και το επιπρόσθετο πλεονέκτημα της μη ανάγκης πρότερης γνώσης των thread που θα χρησιμοποιηθούν (δυναμμικότητα).