Συστήματα Παράλληλης Επεξεργασίας

Εργαστηριακή αναφορά 4
Ταυτόχρονες Δομές Δεδομένων

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

Δεκέμβριος 2022

1 Εισαγωγή

1.1 Σκοπός άσκησης

Σκοπός του συγκεκριμένου ερωτήματος της άσκησης είναι η εξοικείωση με την εκτέλεση εφαρμογών σε σύγχρονα πολυπύρηνα συστήματα και η αξιολόγηση της επίδοσής τους. Συγκεκριμένα, μας δίνονται διάφορες ταυτόχρονες υλοποιήσεις μίας απλά συνδεδεμένης ταξινομημένης λίστας και καλούμαστε να εκτελέσουμε κάποια πειράματα με αυτές με στόχο την αξιολόγηση της επίδοσής τους κάτω απο διαφορετικές συνθήκες.

1.2 Περιγραφή παραγωγής αποτελεσμάτων

Πιο συγκεκριμένα, μας δίνονται οι παρακάτω ταυτόχρονες υλοποιήσεις μίας απλά συνδεδεμένης λίστας:

- Coarse-grain locking
- Fine-grain locking
- Optimistic synchronization
- Lazy synchronization
- Non-blocking synchronization

Εκτελούμε πρόγραμμα που μας δίνεται στο οποίο δοκιμάζουμε τις παραπάνω υλοποιήσεις, και την σειριακή έκδοση, για διαφορετικό ποσοστό ευρέσεων, προσθηκών και διαγραφών των στοιχείων. Πιο συγκεκριμένα εκτελούμε για:

- 100-0-0,

- 80-10-10,
- 20-40-40,
- 0-50-50.

[% ευρέσεων-% προσθηκών-% διαγραφών]

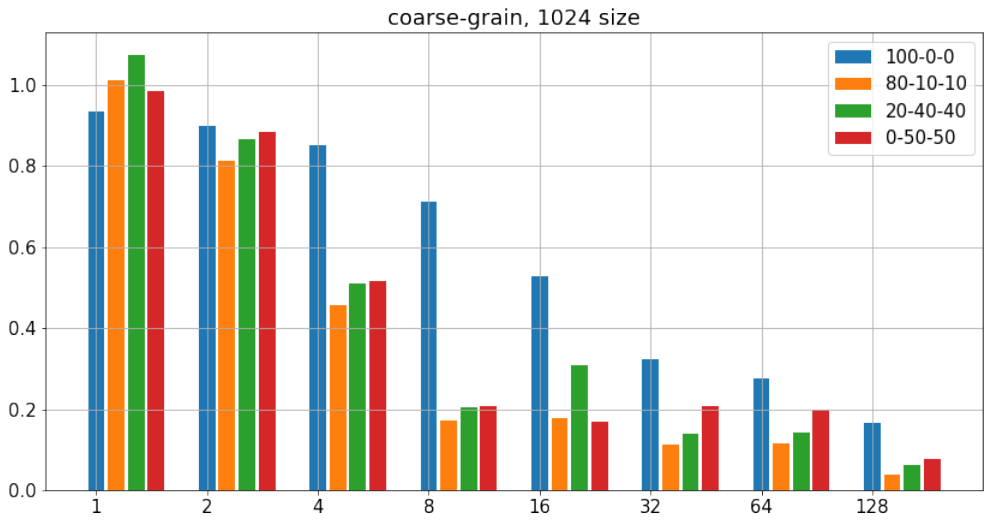
Επίσης την κάθε παραπάνω εκτέλεση την δοκιμάζουμε για 2 μεγέθη λίστας (1024 και 8192). Η κάθε εκτέλεση κρατάει 10 second και μας δίνεται ο αριθμός των `Kops/sec`, δηλαδή το throughput. Κατασκευάζουμε τα διαγράμματα διαιρούντας το throughput της εκάστοτε εκτέλεσης με το αντίστοιχο throughput της σειριακής. Έτσι κατασκευάζουμε τα διαγράμματα επιτάχυνσης (speedup graphs) των εκτελέσεων για αριθμό νημάτων 1, 2, 4, 8, 16, 32, 64 και 128. Σε αυτό το σημείο να αναφέρουμε πως μέσω της μεταβλητής περιβάλλοντος `MT_CONF` αναθέτουμε κάθε νήμα σε ξεχωριστό πυρήνα του μηχανήματος. Για τις εκτελέσεις με 128 νήματα αναθέτουμε 2 νήματα σε κάθε πυρήνα, γιατί το μηχάνημα εκτέλεσης είναι 64-πύρηνο.

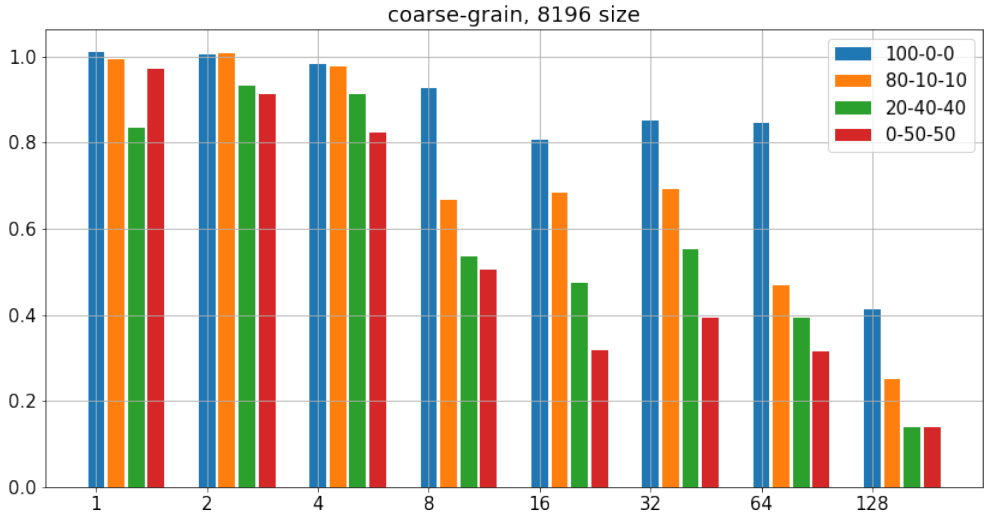
2 Coarse-grain Synchronization

2.1 Χαρακτηριστικά

Στον τρόπο συγρονισμού `Coarse-grain` χρησιμοποιείται ένα κλείδωμα για όλη την δομή της απλά συνδεδεμένης λίστας. Για αυτόν τον λόγο, ενώ η υλοποίηση είναι εύκολη, περιορίζεται η παράλληλη πρόσβαση νημάτων στη δομή.

2.2 Αποτελέσματα





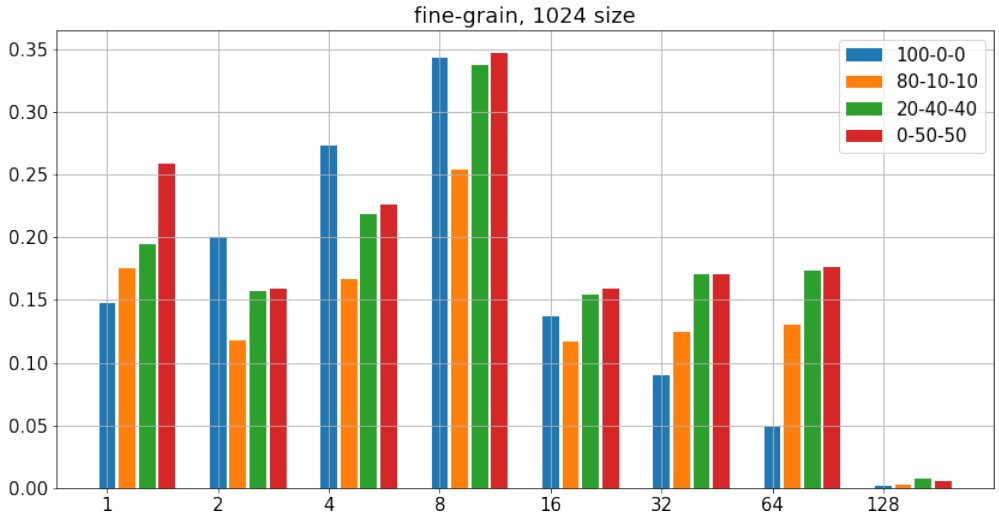
Παρατηρούμε πως ειδικά για την μικρή λίστα δεν υπάρχει βελτίωση στην επιτάχυνση όσο αυξάνονται τα νήματα. Μάλιστα, η παράλληλη επίδοση είναι χειρότερη από την σειριακή. Παρατηρούμε, επίσης, πως ελάχιστα καλύτερη επίδοση από τα υπόλοιπα έχει το ποσοστό λειτουργιών `100-0-0`. Αυτό συμβαίνει καθώς η λειτουργία της εύρεσης στοιχείου καταλαμβάνει για πολύ μικρότερο χρονικό διάστημα το κλείδωμα της δομής και επομένως δεν υπάρχουν μεγάλες ουρές αναμονής νημάτων.
Για το μεγαλύτερο μέγεθος της λίστας φαίνεται πως ο ρυθμός επιδείνωσης της επίδοσης είναι μικρότερος.

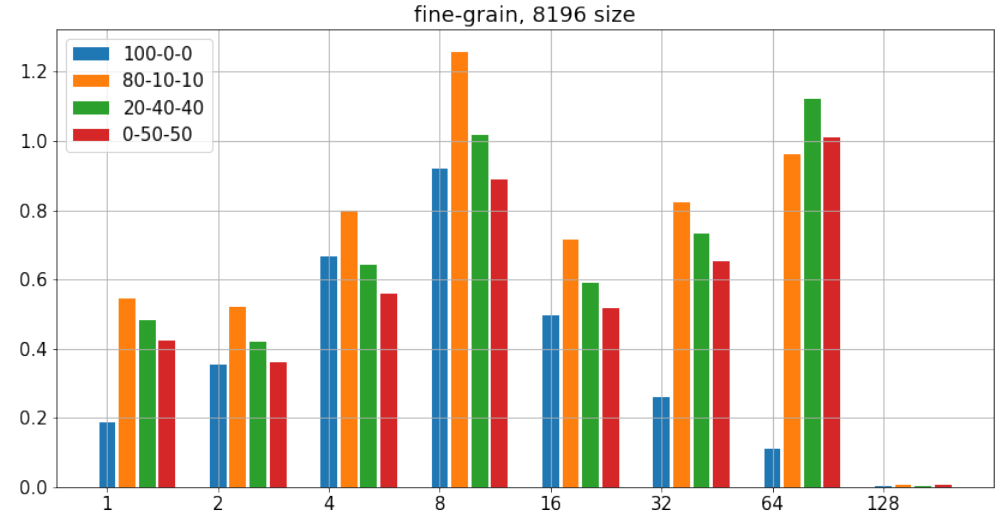
3 Fine-grain Synchronization

3.1 Χαρακτηριστικά

Στο `fine-grain synchronization` έχουμε πολλαπλά κλειδώματα σε τμήματα της δομής και έτσι εξασφαλίζεται η παράλληλη πρόσβαση στη δομή από πολλαπλά νήματα. Ωστόσο η υλοποίηση είναι σημαντικά δυσκολότερη και υπάρχει υψηλό κόστος απελευθέρωσης των κλειδωμάτων. Θεωρητικά, περιμένουμε καλύτερες επιδόσεις από το `Coarse-grain synchronization`.

3.2 Αποτελέσματα





Αρχικά, φαίνεται πως η πρόβλεψή μας πως η fine-grain σημειώνει βελτίωση σε σχέση με την coarse-grain ηταν λανθασμένη, καθώς η μέγιστη επιτάχυνση για μέγεθος λίστας 1024 είναι 0.35 στο fine-grain, ενώ περίπου 1 στο coarse-grain (παραπάνω από διπλάσια). Σε κάποιο βαθμό αυτό είναι αναμενόμεν: ο συγχρονισμός συνεπάγεται ένα κόστος που πρέπει να πληρώσει η CPU. Ο χρόνος που αφιερώνεται στον συγχρονισμό, λόγω της δημιουργίας πολλών κλειδωμάτων τα οποία δημιουργούν φραγμούς μνήμης, αφαιρείται από τον συνολικό διαθέσιμο χρόνο για τα νήματα να εκτελέσουν τις λειτουργίες.
Ωστόσο, παρατηρούμε πως οι δύο μέθοδοι έχουν διαφορετική παράλληλη επίδοση. Η `coarse-grain` φαίνεται να μην επωφελείται από την παραλληλοποίηση, ενώ η `fine-grain` σημειώνει βελτίωση μέχρι και τα 8 threads, χωρίς όμως να ξεπερνάει κατά πολύ την σειριακή επίδοση. Αξιοσημείωτη

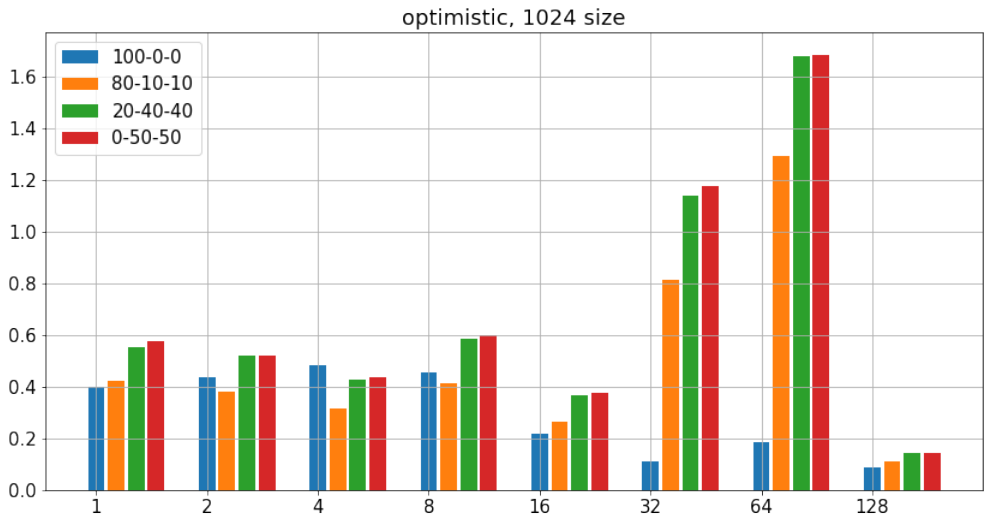
είναι, επίσης, η κάκιστη επίδοση των 128 νημάτων. Σε αυτή την μέθοδο φαίνεται πως η ταυτόχρονη ύπαρξη δύο νημάτων στον ίδιο επεξεργστή δυσχεραίνει κατά πολύ την επίδοση.

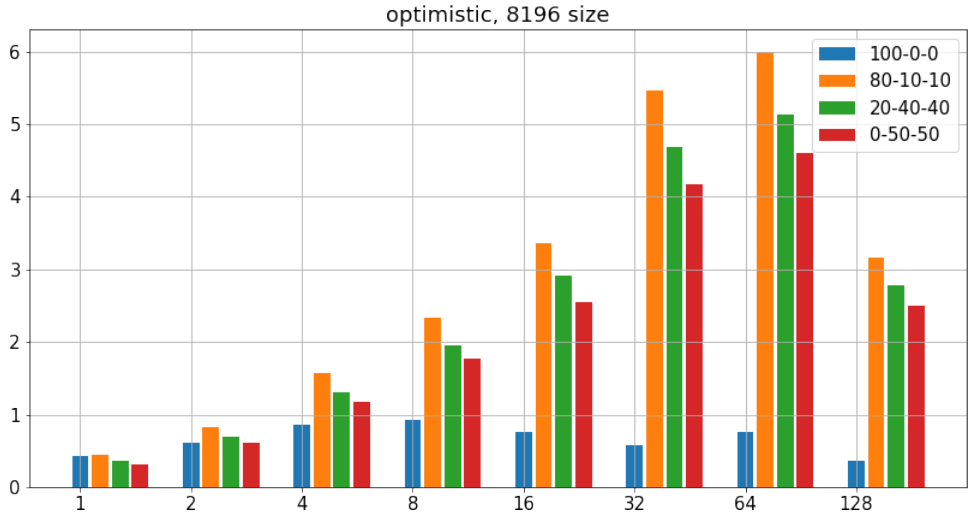
4 Optimistic Synchronization

4.1 Χαρακτηριστικά

Στην `optimistic` μέθοδο συγχρονισμού σκεφτόμαστε αισιόδοξα, δηλαδή οι πιθανότητες τα πράγματα να πάνε στραβά είναι λίγες. Πιο συγκεκριμένα, δεν κλειδώνουμε από την αρχή τους κόμβους ενδιαφέροντος κατά την εισαγωγή ή διαγραφή. Αντιθέτως, πρώτα αναζητούμε τα στοιχεία χωρίς κλείδωμα, έπειτα κλειδώνουμε και ελέγχουμε αν η δομή παραμένει συνεπής. Αν δεν παραμένει, απαλευθερώνουμε τα κλειδώματα και ξαναπροσπαθούμε.

4.2 Αποτελέσματα





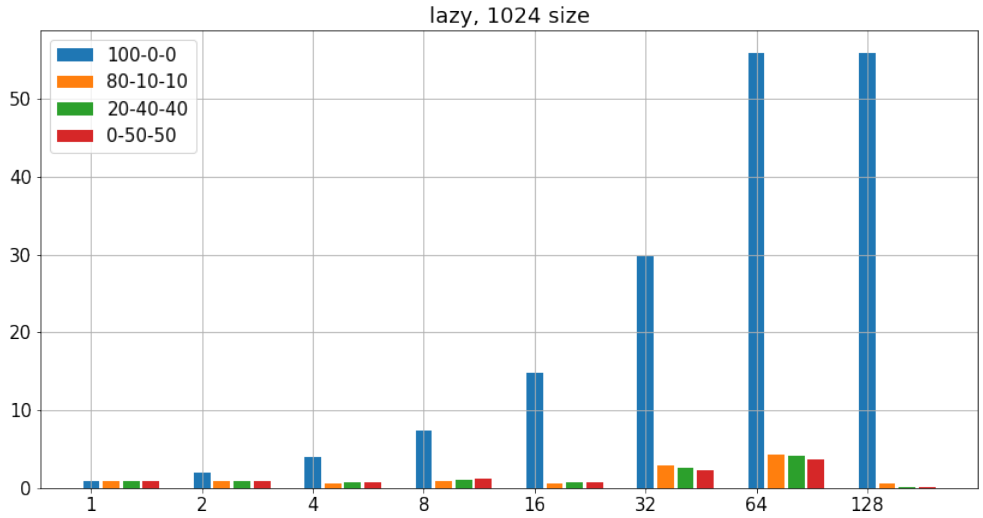
Για μικρά μεγέθη λίστας η `optimistic` εμφανίζει ελάχιστα καλύτερες επιδόσεις από την σειριακή -με ποσοστό από 20% έως 60%- για 32 και 64 νήματα, ενώ για μεγάλα μεγέθη έχει συνεχή κλιμάκωση με άνω όριο τα 64 νήματα και μέγιστο συντελεστή βελτίωσης x6. Παρ'όλα αυτά, η αισιόδοξη μέθοδος φαίνεται να είναι ακατάλληλη για αποκλειστική λειτουργία εύρεσης στοιχείων (100-0-0). Περιμέναμε, άλλωστε, να μην υπάρχει βελτίωση σε αυτό το ποσοστό λειτουργιών, καθώς όπως περιγράψαμε και παραπάνω, η `optimistic` βελτιώνει τις λειτουργίες προσθήκης και διαγραφής στοιχείων με την αποφυγή επιπρόσθετων κλειδωμάτων. Η λειτουργία εύρεσης, ωστόσο, χρησιμοποιεί κλειδώματα. Στην επόμενη μέθοδο, δεν χρησιμοποιούνται κλειδώματα στην εύρεση.

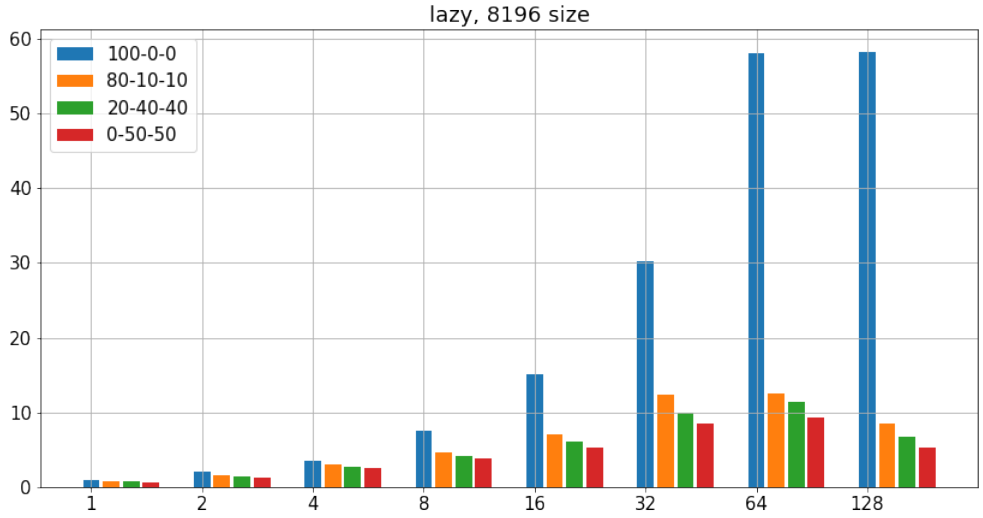
5 Lazy Synchronization

5.1 Χαρακτηριστικά

Η `lazy` αποτελεί βελτιωμένη έκδοση της `optimistic` αφαιρώντας τα κλειδώματα στην `contains` και αποφαεύγοντας την διάσχιση όλης της λίστας για στον έλεγχο συνέπειας (`validate`). Αυτές οι δύο αλλαγές επιτυγχάνονται μέσω της αντιστοίχισης μίας Boolean μεταβλητής σε κάθε κόμβο που δείχνει αν ο κόμβος βρίσκεται στην λίστα ή έχει διαγραφεί (λογικά, όχι φυσικά). Έτσι, η contains δεν χρειάζεται κλείδωμα, αφού απλώς ελέγχει την Boolean. Η validate κάνει απλώς τοπικούς ελέγχους στους κόμβους ενδιαφέροντος. Σημαντική λεπτομέρεια αυτής της μεθόδου είναι πως η λειτουργία διαγραφής (`remove`) είναι `lazy`, δηλαδή λειτουργεί σε δύο βήματα. Πρώτα, αφαιρεί λογικά τον κόμβο (με αλλαγή της Boolean σε false) και έπειτα αφαιρεί φυσικά τον κόμβο επαναθέτοντας τους δείκτες.

5.2 Αποτελέσματα





Η βελτίωση της λειτουργίας `contains` είναι φανερή, αφού παρουσιάζεται αισθητή κλιμάκωση επιτάχυνσης καθώς και μεγάλοι συντελεστές (τουλάχιστον 50πλάσια βελτίωση από την σειριακή για 64 και 128 νήματα. Επίσης, παρατηρούμε βελτίωση και στις υπόλοιπες λειτουργίες, ειδικά για μεγάλα μεγέθη λίστας, αφού για 32 και 64 νήματα έχουμε συντελεστή 10, ενώ στην προηγούμενη μέθοδο είχαμε μέγιστο συντελεστή 6. Ωστόσο, η contains έχει αρκετά πολύ καλύτερη επίδοση από τις υπόλοιπες λειτουργίες. Αυτο υποδηλώνει πως η βελτίωση της `add()` και της `remove()` έχει πλαφόν. Αξιοσημείωτη είναι η επίδοση της λίστας μεγέθους 1024 με 128 νήματα, όπου ενώ οι υπόλοιπες λειτουργίες έχουν γονατίσει, τείνοντας σε μηδενικό συντελεστή βελτίωσης, η λειτουργία `contains` μένει ανεπηρέαστη. Πιθανολογούμε πως αυτή η συμπεριφορά οφείλεται στην συχνότητα των flushes των cache των επεξεργαστών στους οποίους εργάζονται δύο νήματα

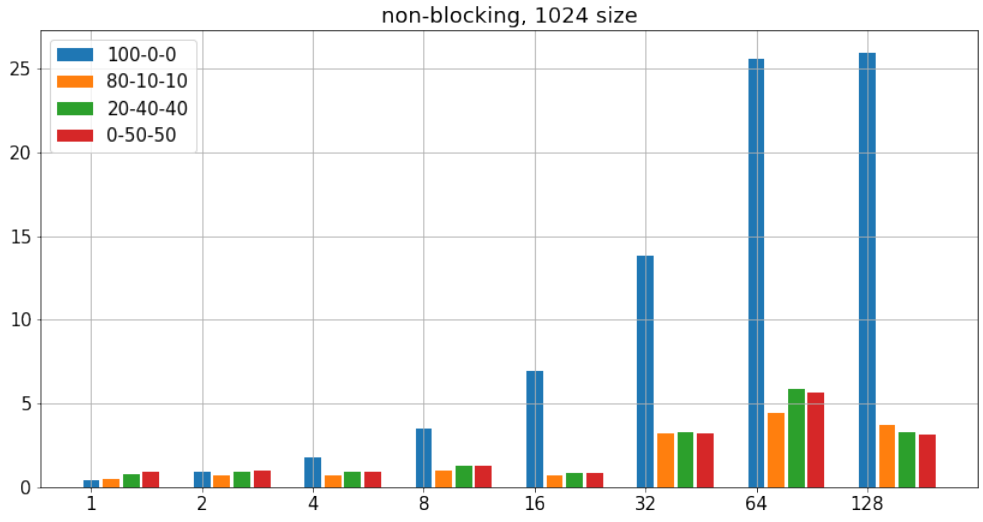
ταυτόχρονα [οι cache περιέχουν blocks πληροφορίας που σχετίζονται με τα κλειδώματα]. Όπως, θα δούμε στην συνέχεια, αυτό δεν συμβαίνει στην μέθοδο του `Non-blocking syncronization` αφού τα κλειδώματα παραλείπονται πλήρως από όλες τις μεθόδους.

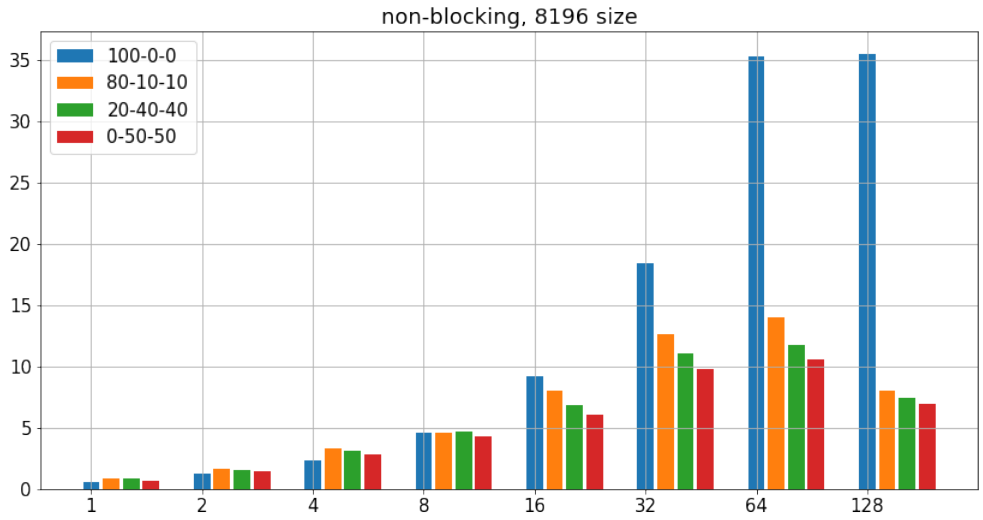
6 Non-blocking Synchronization

6.1 Χαρακτηριστικά

Σε αυτή την μέθοδο, συμπυκνώνονται τα πεδία address και marked field σε μία packed μορφή την οποία ο επεξεργαστής μπορεί να χειριστεί ατομικά. Αυτό επιτρέπει σύνθετες εντολές λογικής. Έτσι, μπορούμε να απαλείψουμε πλήρως τα κλειδώματα και από τις 3 μεθόδους. Συγκεκριμένα, οποιαδήποτε προσπάθεια ενημέρωσης του πεδίου `next` όταν το `marked` ελθβαθ true (ο κόμβος είναι λογικά διεγραμμένος) θα αποτύχει. Η αφαίρεση ενός κόμβου περιλαμβάνει την ενημέρωση του `marked` και μία μόνο απόπειρα να ενημερωθούν οι διευθύνσεις. Κάθε νήμα που διατρέχει την λίστα εκτελώντας την `add()` και `remove()` αφαιρεί φυσικά τους κόμβους που συναντά και έχουν σβηστεί. Η αποτυχία για ατομική ενημέρωση οδηγεί στην επαναπροσπάθεια με διάτρεξη από την αρχή της λίστας. Η `Non-blocking` αποτελεί `Wait-free` μέθοδο.

6.2 Αποτελέσματα





Παρατηρούμε ελάχιστες βελτιώσεις σε σχέση με την προηγούμενη μέθοδο. Αρχικά, όπως αναφέραμε και ρπιν, στα 128 νήματα δεν υπάρχει σημαντική χειροτέρευση, εφόσον δεν χρησιμοποιούνται κλειδώματα. Οι μέθοδοι `add()` & `remove()` μπορεί να μην έχουν αυτή την φορά κλειδώματα, ωστόσο δεν βελτιώνεται αρκετά η επίδοσή τους, αφού μπορεί να είναι `lock-free` αλλά με `retries`. Η μέθοδος `contains()` όπως και πριν έχει πολύ καλύτερες επιδόσεις από τις υπόλοιπες λειτουργίες, αλλά σε σχέση με την `lazy` μέθοδο, παρουσιάζει μικρότερα ποσοστά επιτάχυνσης.