Συστήματα Παράλληλης Επεξεργασίας

Άσκηση 2
Ταυτόχρονες δομές δεδομένων

Ομάδα	parlab04
Θεόδωρος Αράπης	el18028
Εμμανουήλ Βλάσσης	el18086
Παναγιώτης Παπαδέας	el18039

Ζητούμενα

Εκτελέστε πειράματα για τις ταυτόχρονες υλοποιήσεις μίας απλά συνδεδεμένης ταξινομημένης λίστας:

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

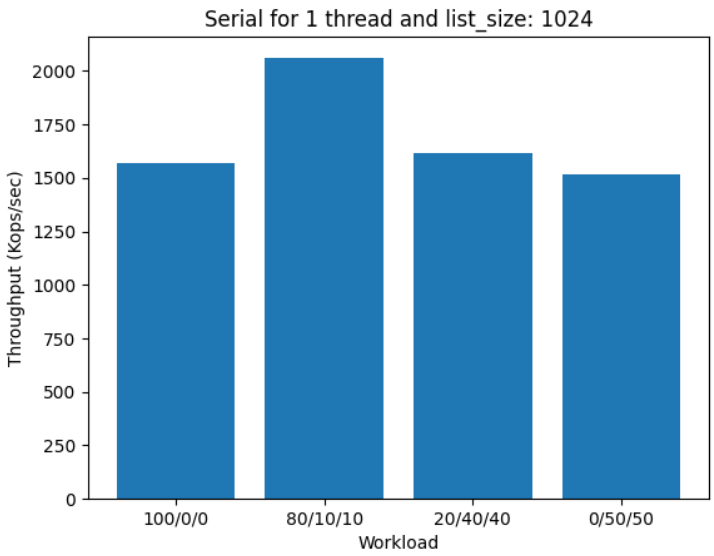
χρησιμοποιώντας τις παρακάτω τιμές για τις παραμέτρους των εκτελέσιμων:

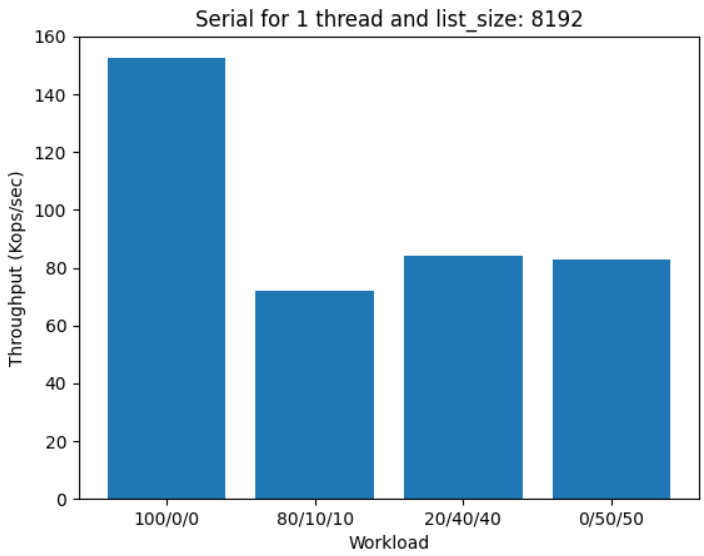
- Αριθμός νημάτων: 1, 2, 4, 8, 16, 32, 64, 128
- Μέγεθος λίστας: 1024, 8192
- Ποσοστό λειτουργιών: 100-0-0, 80-10-10, 20-40-40, 0-50-50. Ο πρώτος αριθμός υποδηλώνει το ποσοστό αναζητήσεων και οι επόμενοι δύο το ποσοστό εισαγωγών και διαγραφών αντίστοιχα.

Μετρήσεις

Serial Version

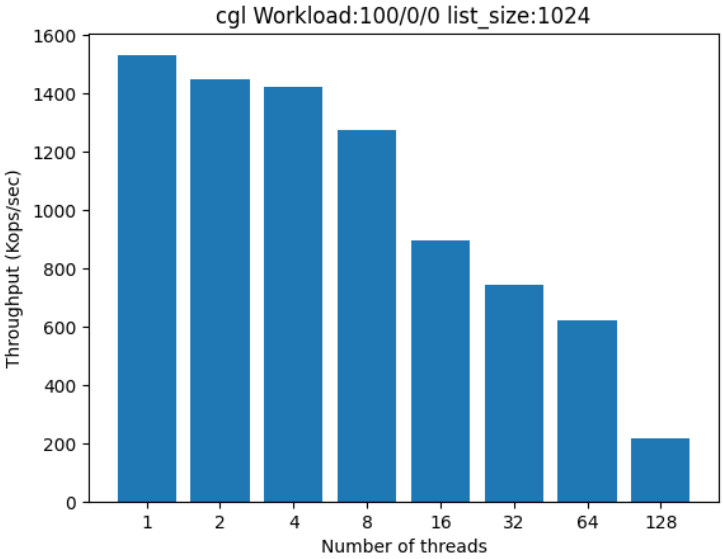
Παραθέτουμε τις μετρήσεις για την σειριακή έκδοση με σκοπό την σύγκριση των επιδόσεων με τις υπόλοιπες υλοποιήσεις:

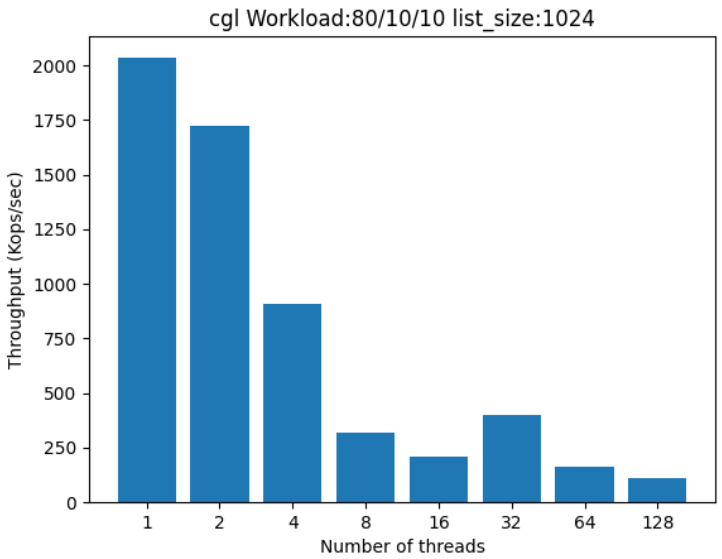


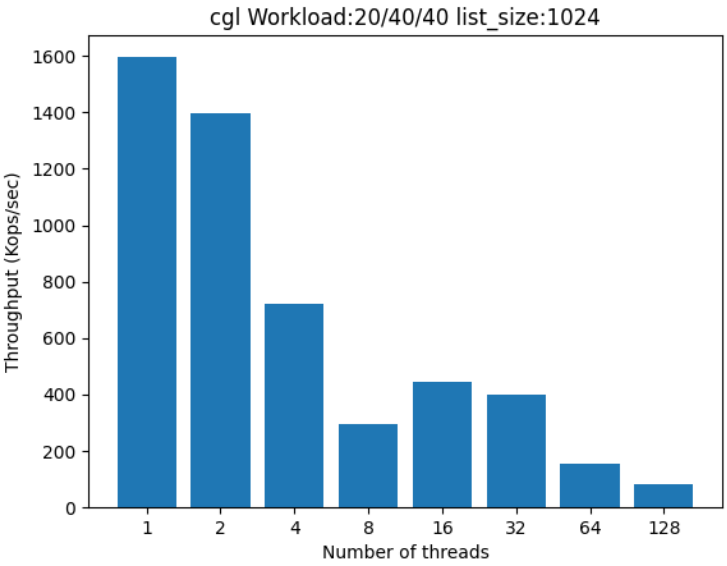


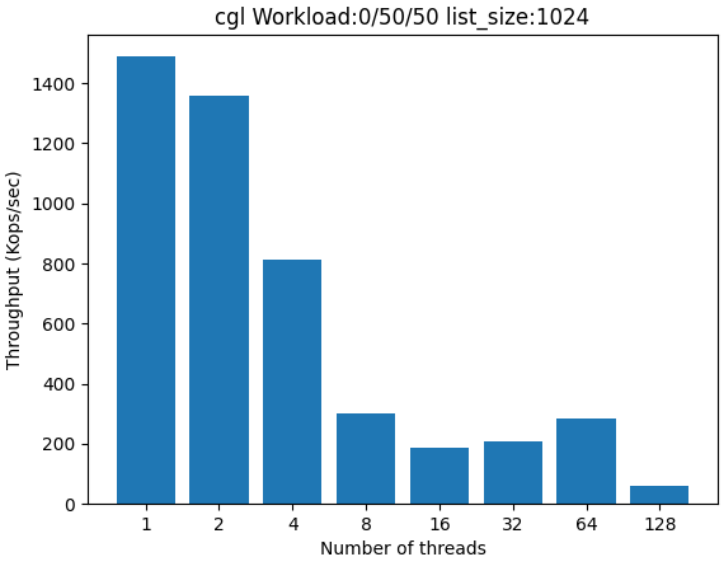
Coarse-grain locking

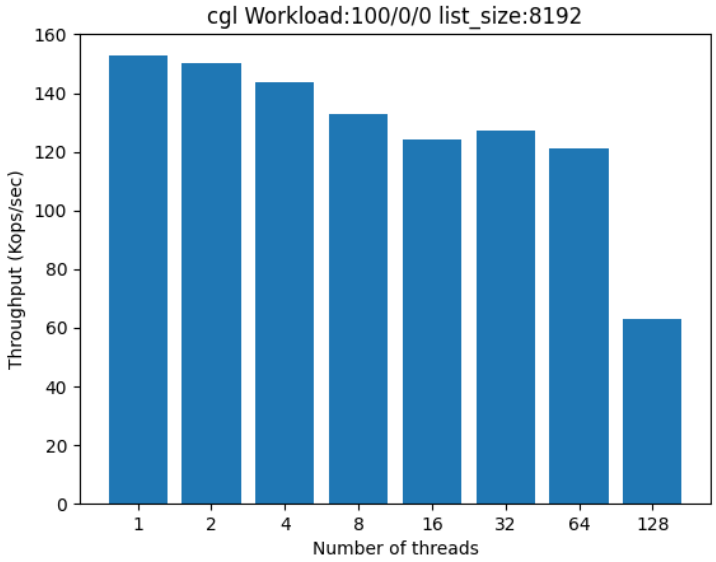
Όπως γνωρίζουμε, στην Coarse-grain locking υλοποίηση κάθε λειτουργία (contains, add, remove) διεκδικεί ένα κοινό lock για ολόκληρη την δομή. Συνεπώς, αναμένουμε βελτίωση σε σχέση με την σειριακή υλοποίηση, αφού τώρα έχουμε παραλληλισμό. Λαμβάνουμε τις εξής μετρήσεις:

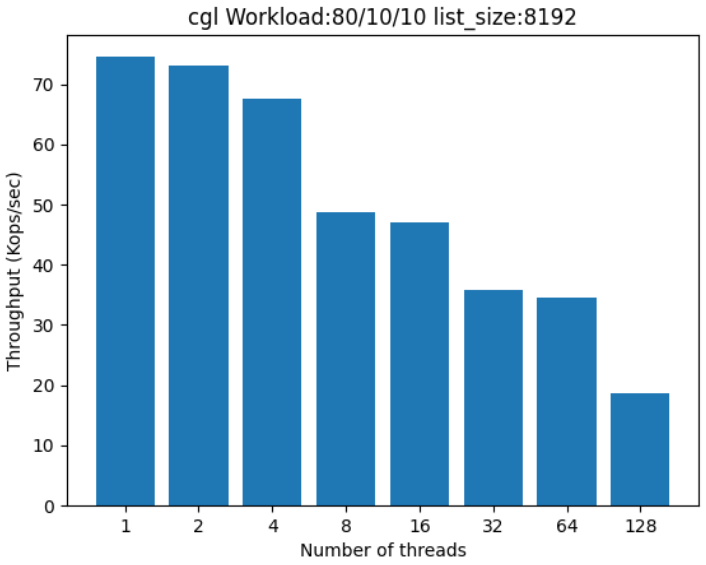


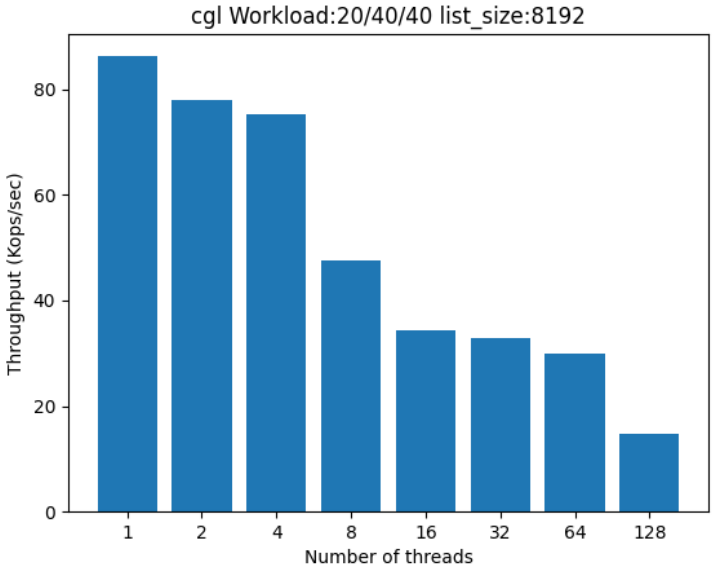


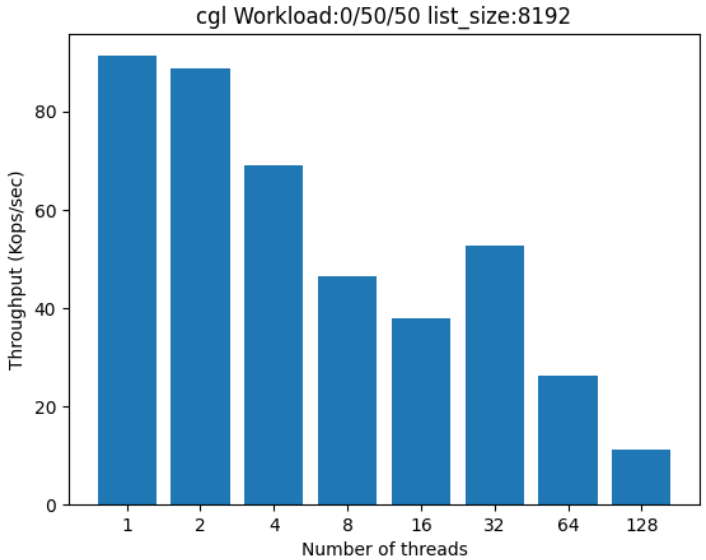

cgl Workload:20/40/40 list_size:1024
Throughput (Kops/sec)
0
200
400
600
800
1000
1200
1400
1600
1
2
4
8
16
32
64
128
Number of threads


cgl Workload:0/50/50 list_size:1024
Throughput (Kops/sec)
0
200
400
600
800
1000
1200
1400
1
2
4
8
16
32
64
128
Number of threads


cgl Workload:100/0/0 list_size:8192
Throughput (Kops/sec)
0
20
40
60
80
100
120
140
160
1
2
4
8
16
32
64
128
Number of threads


cgl Workload:80/10/10 list_size:8192
Throughput (Kops/sec)
0
10
20
30
40
50
60
70
1
2
4
8
16
32
64
128
Number of threads

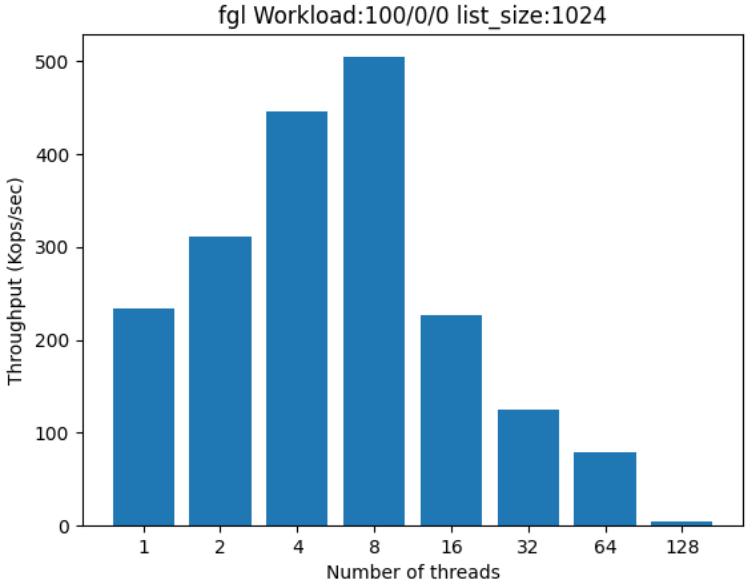

cgl Workload:20/40/40 list_size:8192
Throughput (Kops/sec)
0
20
40
60
80
1
2
4
8
16
32
64
128
Number of threads

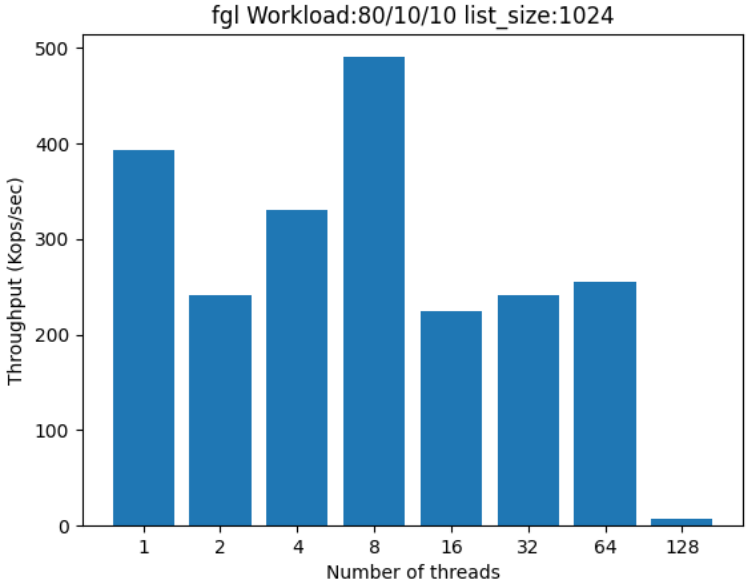

cgl Workload:0/50/50 list_size:8192
Throughput (Kops/sec)
0
20
40
60
80
1
2
4
8
16
32
64
128
Number of threads

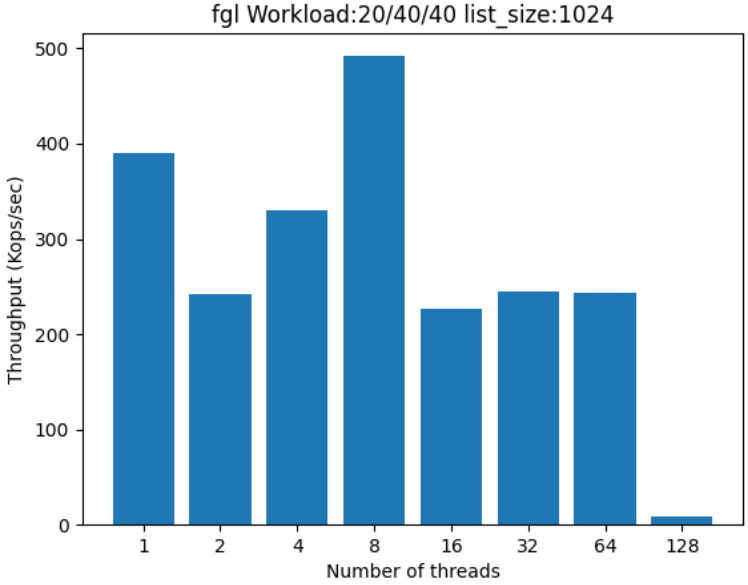
Παρατηρούμε ότι με την αύξηση των threads ελαττώνεται το συνολικό throughput (λειτουργιών ανά δευτερόλεπτο, Kops/sec). Ακόμη, δεν παρατηρούμε καμία αύξηση του throughput στην παράλληλη εκτέλεση με Coarse-grain locking. Είναι λογικό να συμβαίνει αυτό, αφού κλειδώνεται ολόκληρη η υλοποίηση της κάθε λειτουργίας, με αποτέλεσμα οι λειτουργίες να εκτελούνται διαδοχικά και η καθυστέρηση που δημιουργείται για την επικοινωνία και τον συγχρονισμό των locks καθώς και για το context switching να επιβραδύνει την εκτέλεση των λειτουργιών σημαντικά.

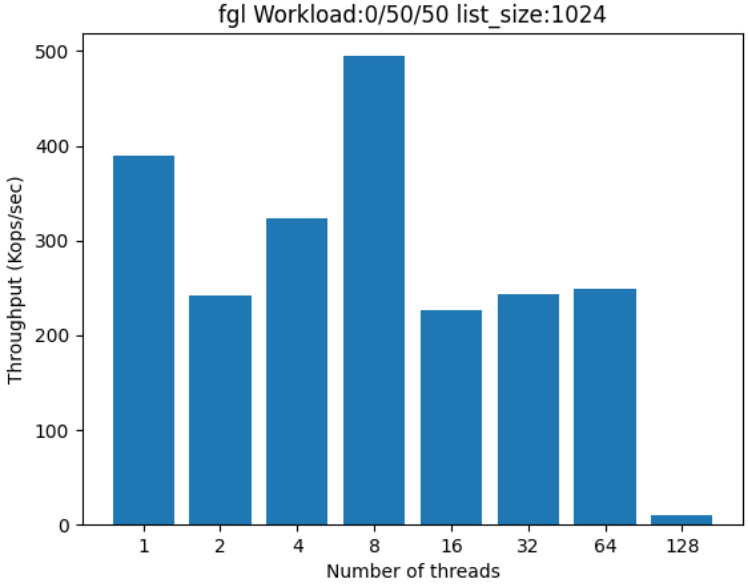
Fine-grain locking

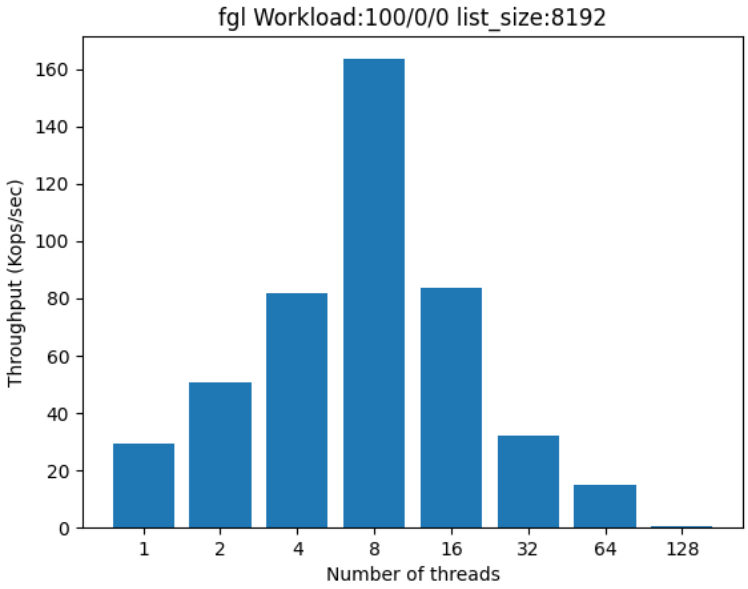
Στην συγκεκριμένη υλοποίηση, αντί να κλειδώνουμε ολόκληρη την δομή, κλειδώνουμε το στοιχείο της λίστας στο οποίο επενεργούμε καθώς και το προηγούμενό του. Επιπλέον, η λίστα διασχίζεται με την τεχνική hand-over-hand locking, όπου το τελευταίο στοιχείο που κλειδώθηκε, ξεκλειδώνει το προηγούμενό του και κλειδώνει το επόμενό του. Οι μετρήσεις που λάβαμε είναι:

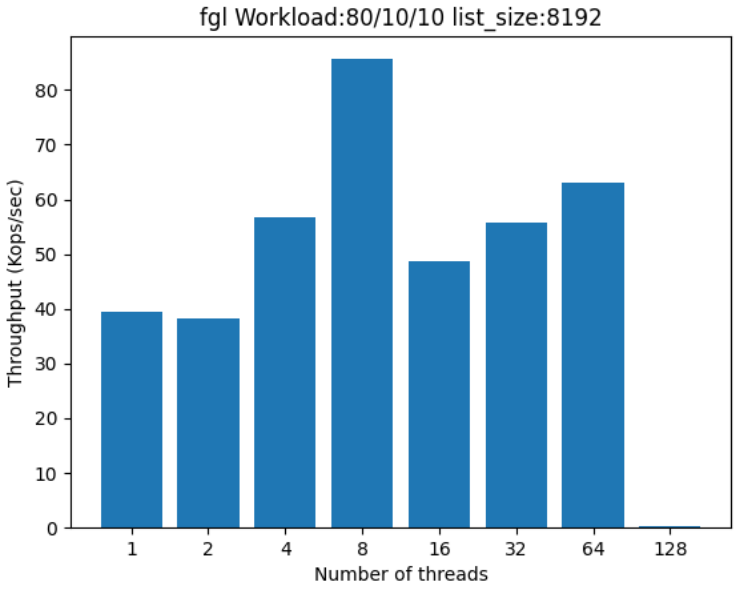


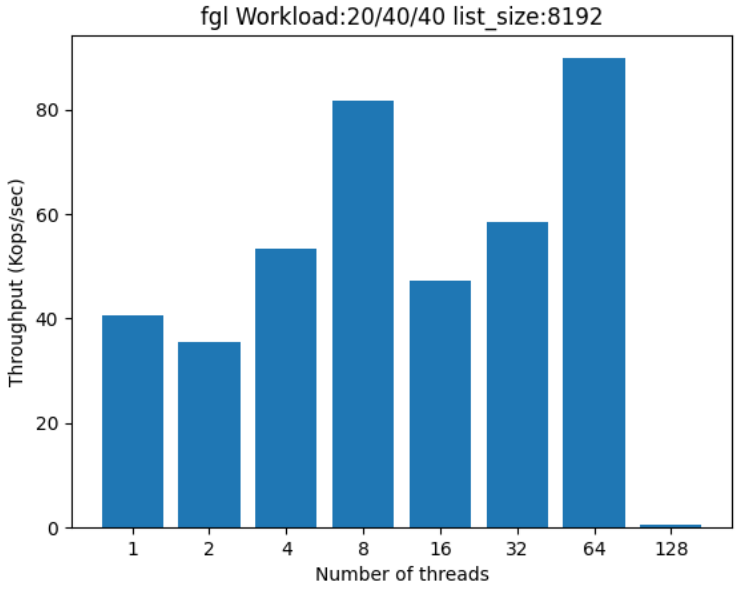


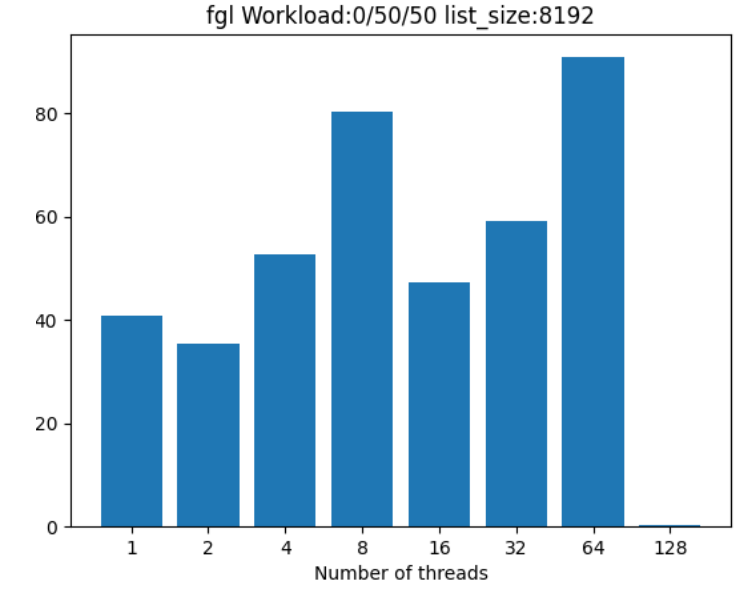








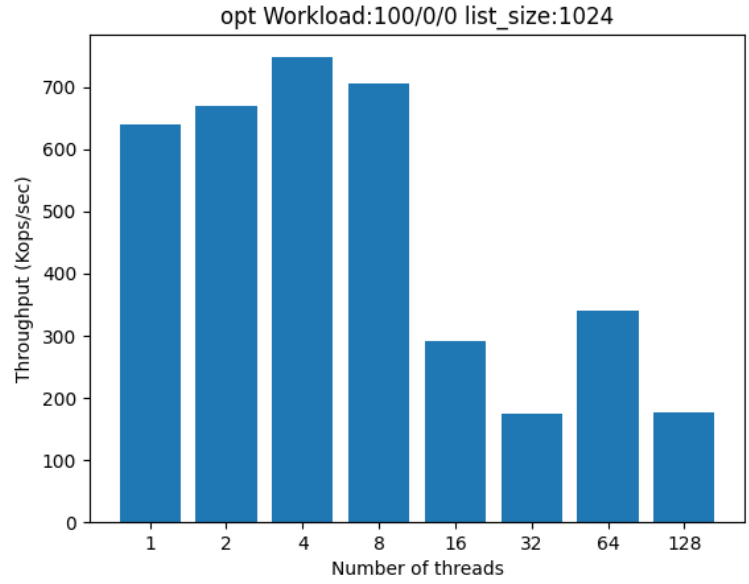


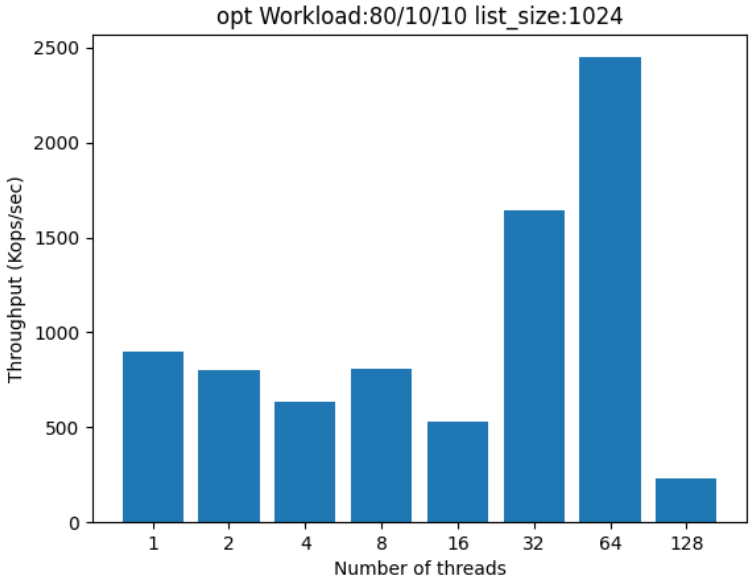


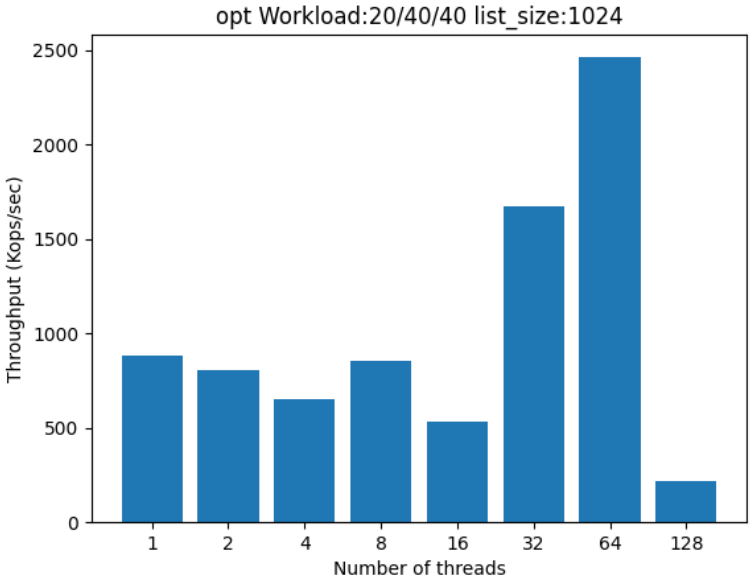
Βλέπουμε ότι, για το μέγεθος λίστας 1024, έχουμε αρκετά μικρότερο throughput σε σχέση με την σειριακή υλοποίηση. Ενδεχομένως αυτό οφείλεται στον μεγάλο αριθμό λήψεων και απελευθερώσεων κλειδωμάτων που συμβαίνουν, τα οποία μπορεί να καθυστερούν νήματα των οποίων οι λειτουργίες αφορούν στοιχεία στη μέση και το τέλος της λίστας. Ως εκ τούτου, το throughput είναι αρκετά μικρότερο και από την Coarse-grain υλοποίηση. Παρατηρούμε ακόμα ότι για 8 threads έχουμε το μέγιστο throughput και για τα δύο μεγέθη λιστών, με εξαίρεση τις περιπτώσεις 20/40/40 και 0/50/50 για μέγεθος λίστα 8192, όπου κυριαρχεί το throughput για πλήθος νημάτων 64. Τέλος, για το μέγεθος λίστας 8192 βλέπουμε ότι πετυχαίνουμε καλύτερο throughput για περισσότερα πλήθη νημάτων σε σχέση με την Coarse-grain υλοποίηση, στην οποία το μέγιστο throughput ήταν για 1 thread (όχι παραλληλοποίηση), χωρίς όμως να ξεπερνάμε την σειριακή υλοποίηση.

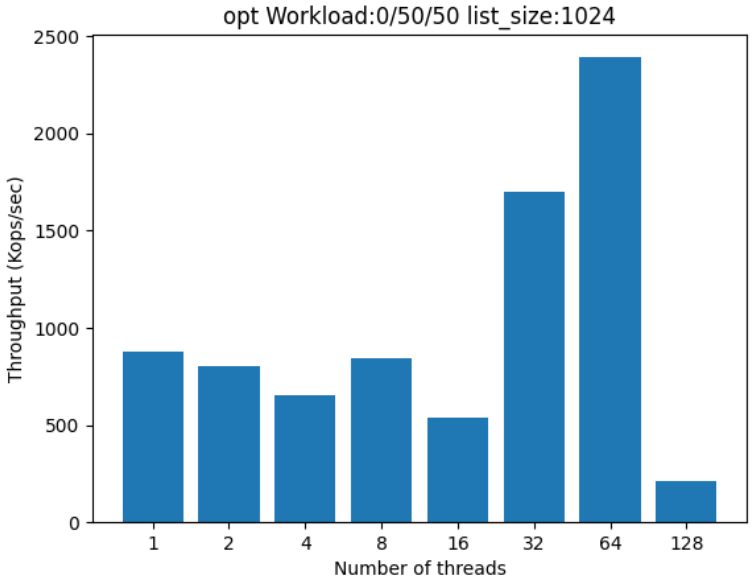
Optimistic Sychronization

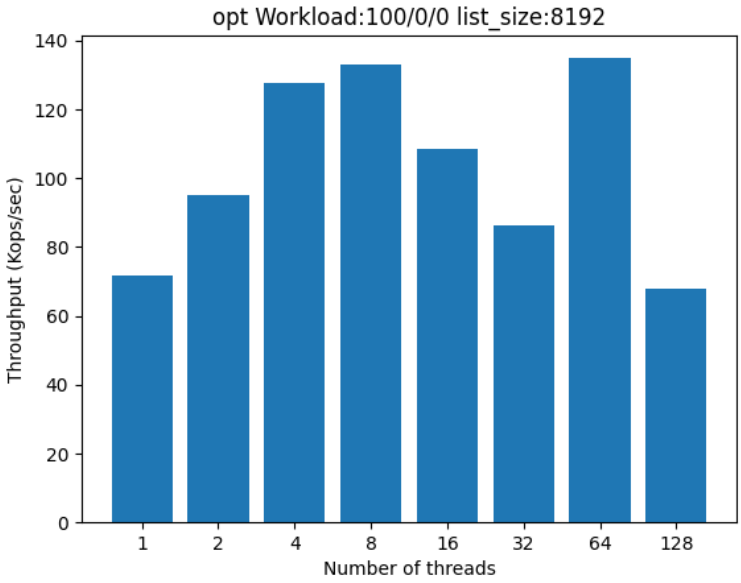
Στην Optimistic Sychronization υλοποίηση, αντί να χρησιμοποιούμε hands-over-hands τεχνική στην αναζήτηση των στοιχείων για εισαγωγή και εξαγωγή, αποφεύγουμε τα κλειδώματα κατά την διάσχιση της λίστας και κλειδώνουμε μόνο το κόμβο προς αφαίρεση (ή στον οποίο θα προστεθεί το νέο στοιχείο) και τον προηγούμενό του. Ύστερα, ελέγχουμε (διασχίζοντας ξανά την λίστα) αν το προηγούμενο στοιχείο από το τρέχον κόμβο δείχνει όντως στο τρέχον και συνεχίζουμε την εκτέλεση της λειτουργίας, αλλιώς ξαναπροσπαθούμε. Για την λειτουργία contains χρησιμοποιούμε κλειδώματα. Λαμβάνουμε τις ακόλουθες μετρήσεις:

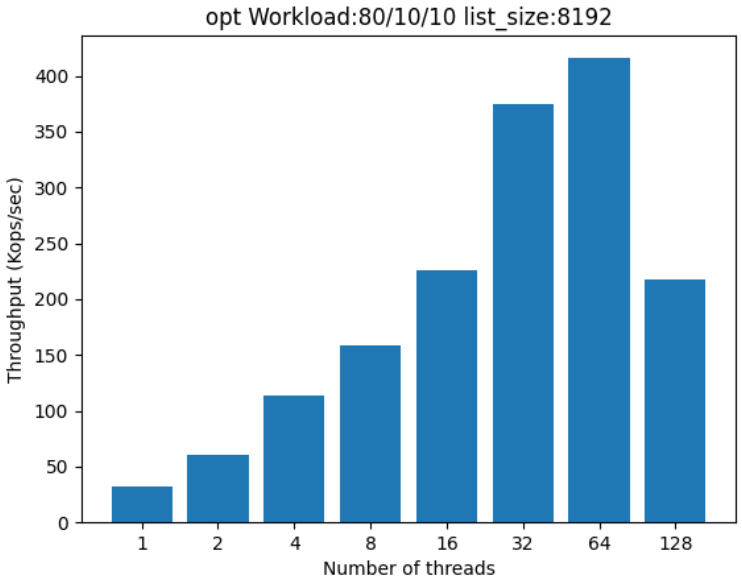


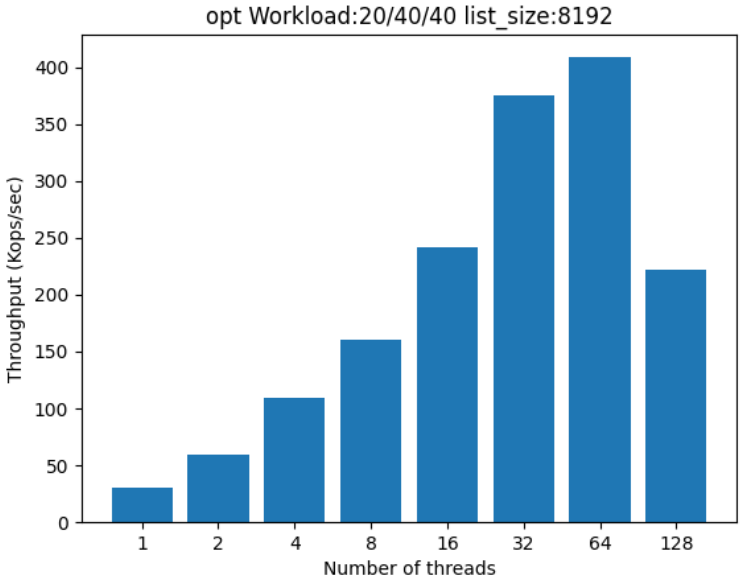


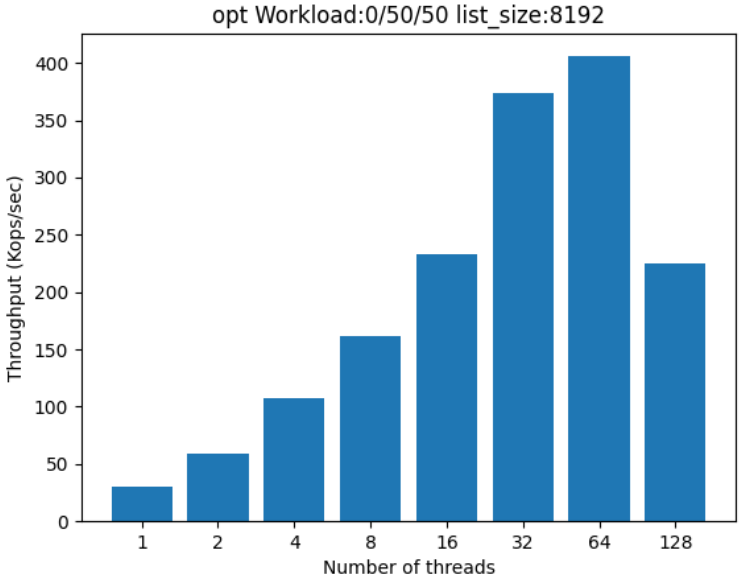








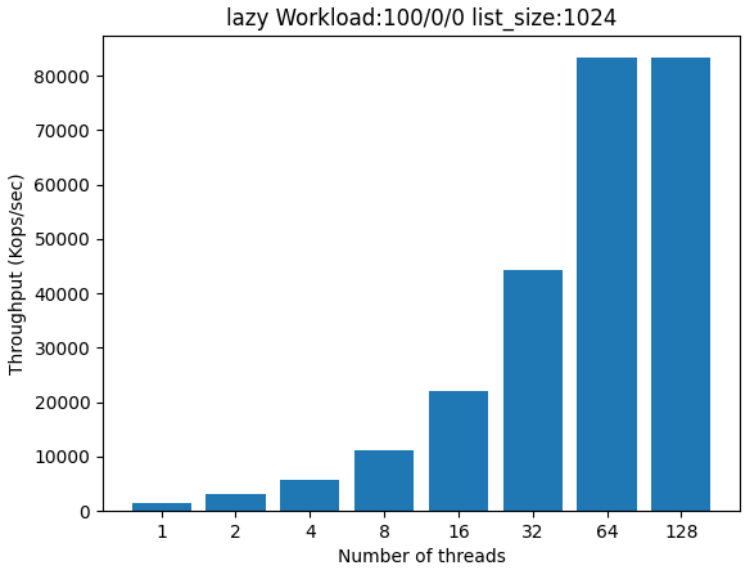


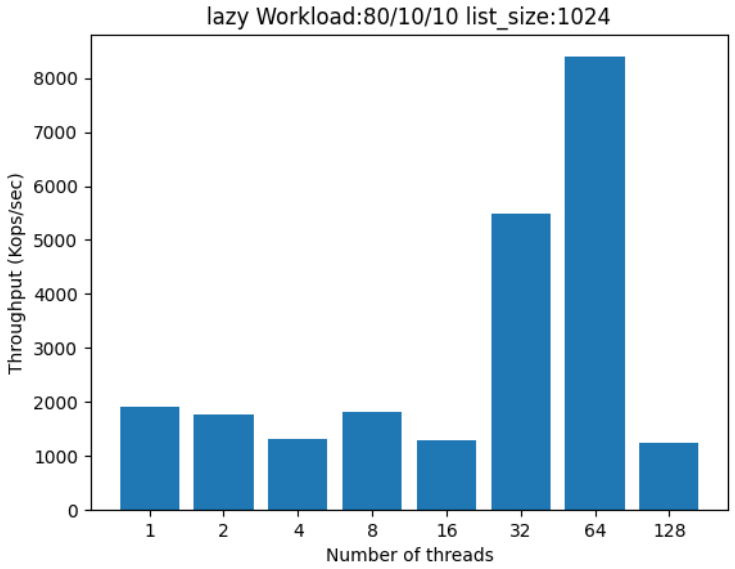


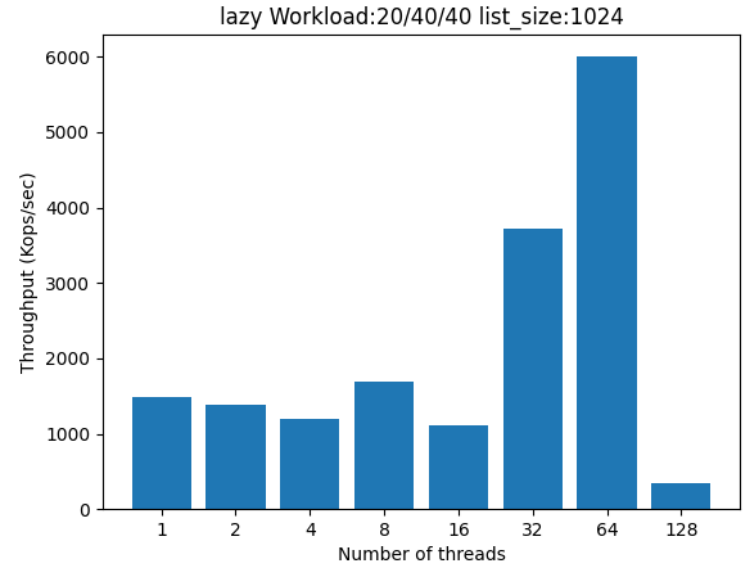
Από τα παραπάνω διαγράμματα βλέπουμε ότι για την περίπτωση που έχουμε 100% λειτουργίες contains το throughput είναι σχετικά χαμηλό, ενώ για την λίστα μεγέθους 1024 είναι πολύ χαμηλότερο της serial υλοποίησης. Όμως, στις υπόλοιπες περιπτώσεις που έχουμε λιγότερες λειτουργίες contains και περισσότερες add/remove, βλέπουμε ότι το throughput αυξάνεται σημαντικά, κυρίως για την λίστα μεγέθους 8192. Ο λόγος γι' αυτό είναι πιθανότατα το γεγονός ότι οι λειτουργίες contains περιέχουν πολλά κλειδώματα ενώ οι add/remove κλειδώνουν μόνο τους κατάλληλους κόμβους. Επιπρόσθετα, σημειώνουμε ότι το throughput αυξάνεται εκθετικά με την αύξηση των threads για την περίπτωση της λίστας μεγέθους 8192 (εκτός της περίπτωσης 100/0/0), που σημαίνει ότι το κόστος για να διατρέξουμε την λίστα μία φορά με κλειδώματα είναι μεγαλύτερο από να την διατρέξουμε 2 φορές χωρίς κλειδώματα. Τέλος, σε όλες τις περιπτώσεις, εκτός της περίπτωσης «100/0/0 list_size:1024», έχουμε μέγιστο throughput για 64 threads.

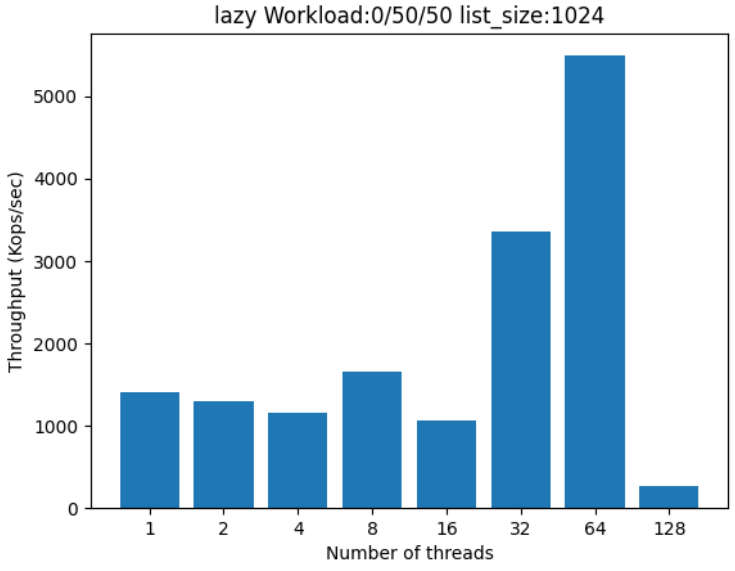
Lazy Sychronization

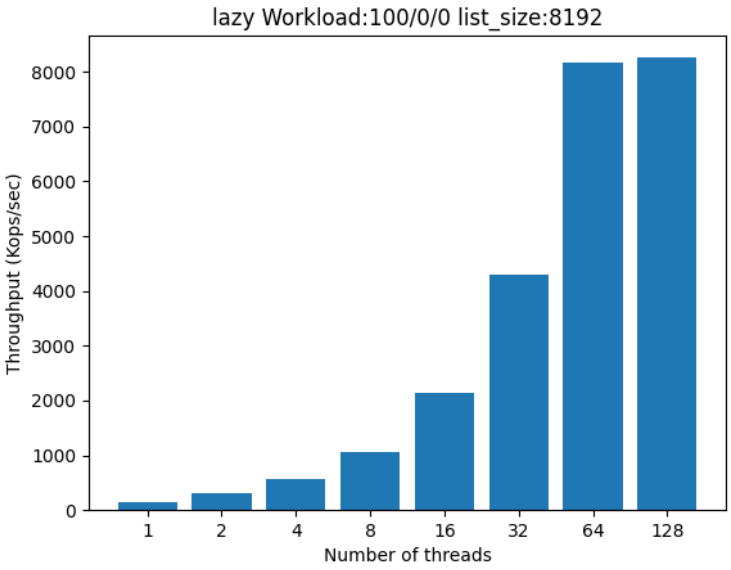
Η υλοποίηση αυτή αποτελεί βελτίωση της προηγούμενης. Πιο συγκεκριμένα, η contains διατρέχει τη λίστα χωρίς να χρησιμοποιεί κλειδώματα (wait-free), ελέγχοντας επιπλέον μία μεταβλητή boolean που επιβεβαιώνει την ύπαρξη του τρέχοντος στοιχείου στη λίστα. Η add κάνει ότι και προηγουμένως, μόνο που στο validation (έλεγχο) δεν ξανατρέχουμε την λίστα, αλλά κάνουμε τοπικούς ελέγχους στον τρέχον κόμβο και τον προηγούμενό του. Η remove τώρα λειτουργεί με έναν lazy τρόπο σε δύο φάσεις: στην πρώτη φάση θέτει την προαναφερθείσα boolean μεταβλητή ως false και στην δεύτερη αφαιρεί τον κόμβο από την λίστα. Τα αποτελέσματα των μετρήσεων είναι τα εξής:

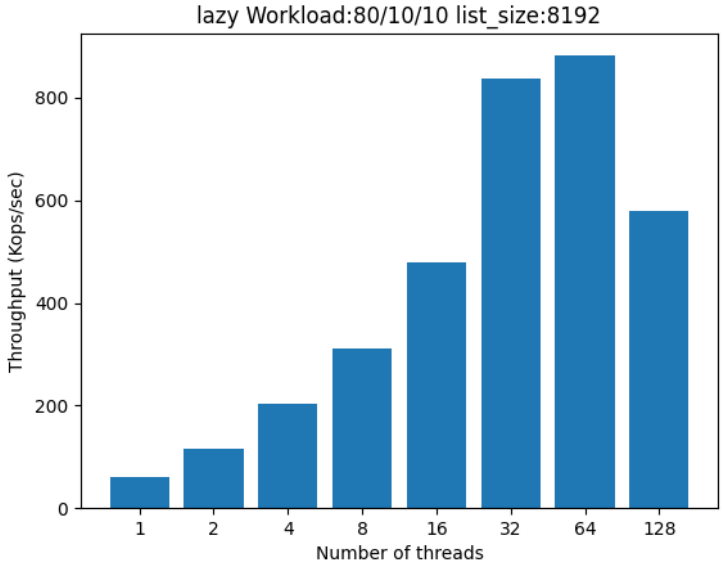


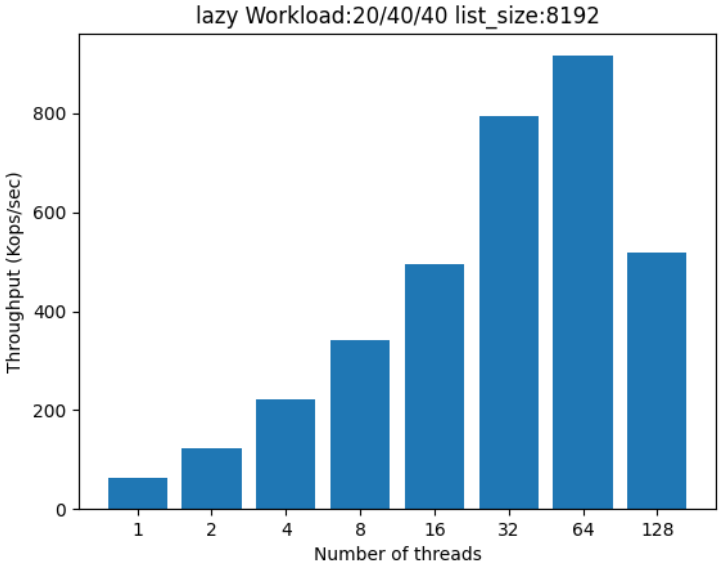


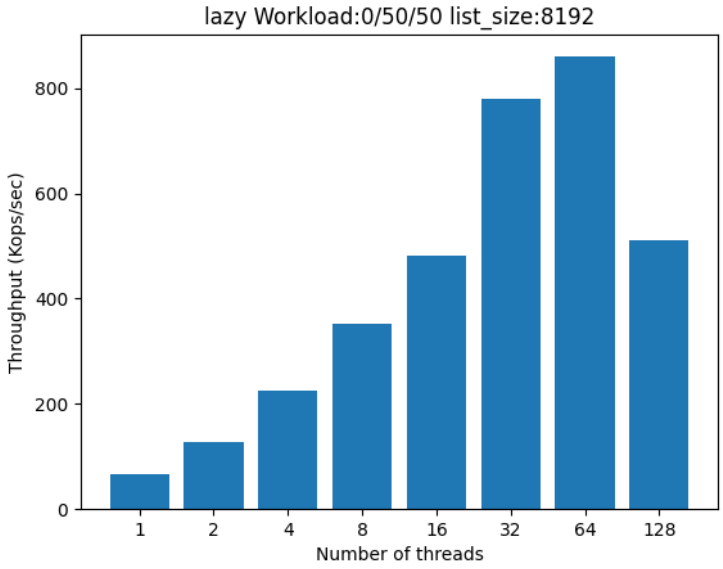








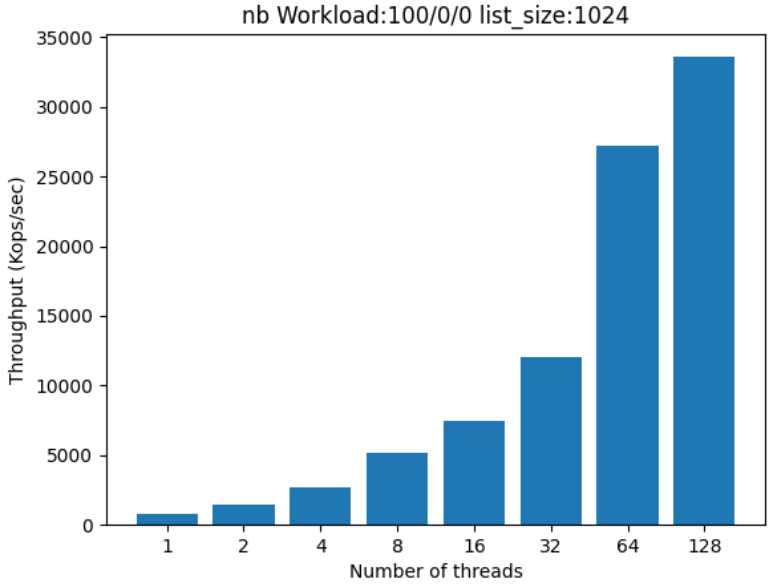


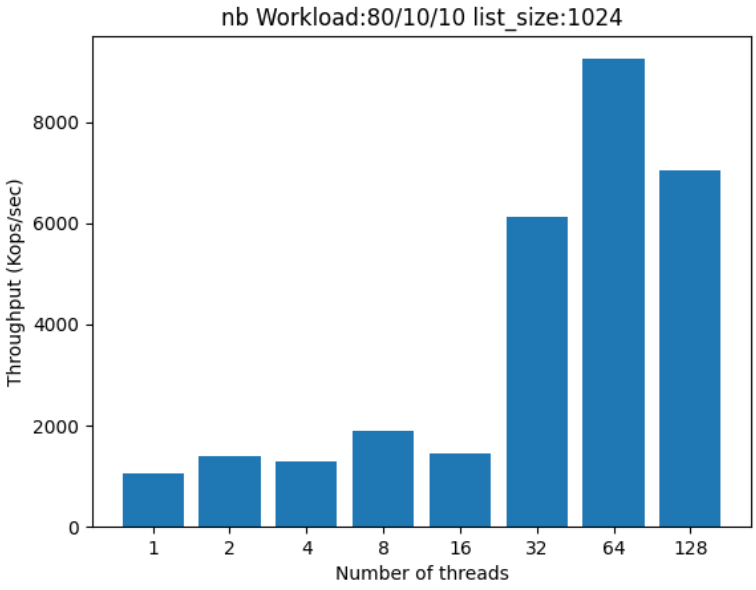


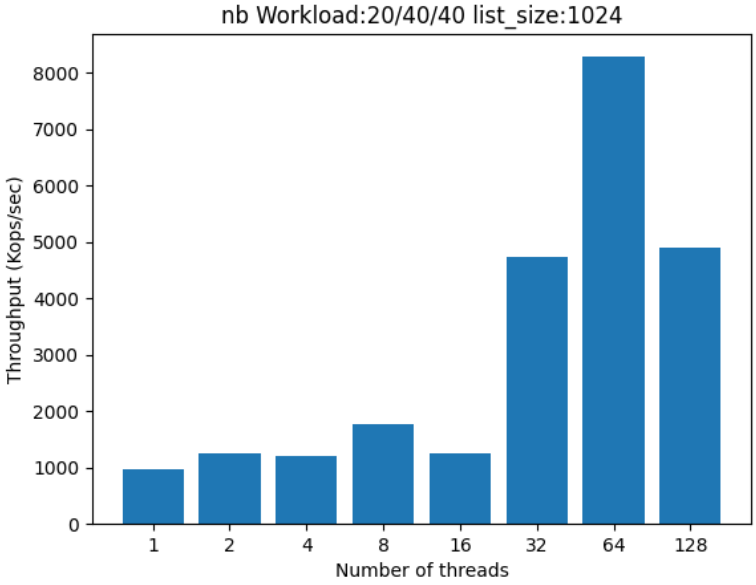
Όπως αναμέναμε, η lazy synchronization υλοποίηση έχει πολύ μεγαλύτερο throughput από όλες τις προηγούμενες υλοποιήσεις, ειδικά στην περίπτωση 100/0/0 (όπου παρατηρούμε δεκαπλάσιο throughput σε σχέση με τις υπόλοιπες περιπτώσεις της Lazy). Αυτό είναι λογικό αφού πλέον η λειτουργία contains δεν χρησιμοποιεί κλειδώματα και άρα η διαδικασία διάσχισης του πίνακα είναι σημαντικά πιο γρήγορη. Επιπλέον, το validation διαρκεί λιγότερο οπότε επιταχύνεται και η λειτουργία add. Όπως και στην περίπτωση του Optimistic Synchronization έχουμε μέγιστο throughput για 64 thread, με εξαίρεση τις περιπτώσεις 100/0/0 όπου έχουμε μέγιστο για 128 threads. Ακόμη, σε όλες τις περιπτώσεις για μέγεθος λίστας 8192, καθώς και της περίπτωσης 100/0/0 για μέγεθος λίστας 1024, παρατηρούμε εκθετική αύξηση του throughput των threads (μέχρι τα 64 threads για τις περιπτώσεις που αναφέραμε προηγουμένως).

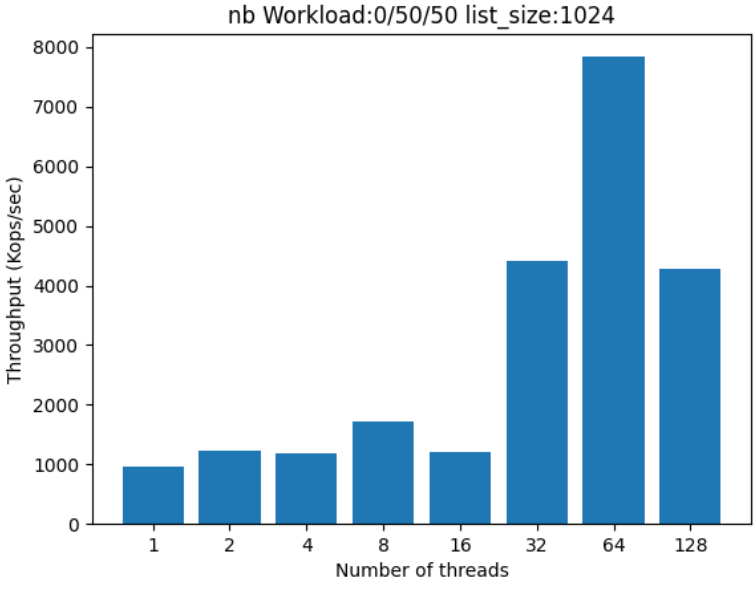
Non-blocking Sychronization

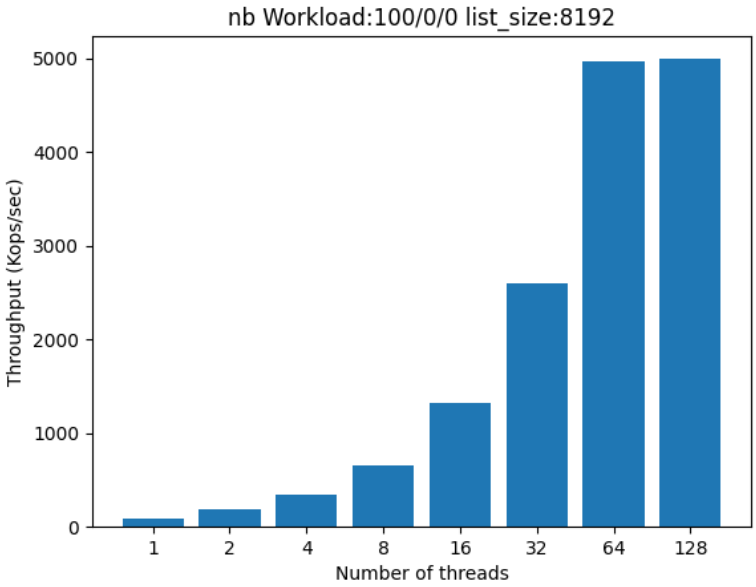
Σε αυτήν την υλοποίηση, δεν χρησιμοποιούμε locks πουθενά, δηλαδή οι add/remove είναι lock-free (with retries) ενώ η contains παραμένει wait-free. Αυτό επιτυγχάνεται συγχωνεύοντας την boolean που αναφέραμε στην Lazy υλοποίηση με το πεδίο next του κόμβου, το οποίο δείχνει στο επόμενο στοιχείο της λίστας. Έτσι, η boolean αυτή λειτουργεί τώρα ως flag (true-δεν υπάρχει/false-υπάρχει) που επιτρέπει την ενημέρωση του πεδίου next του κόμβου, εφόσον αυτός υπάρχει λογικά. Κατά την remove, ενημερώνεται η boolean και γίνεται μία μόνο προσπάθεια ενημέρωσης των διευθύνσεων. Παράλληλα, κάθε νήμα που εκτελεί λειτουργία add ή remove αφαιρεί φυσικά τους κόμβους που συναντά να έχουν σβηστεί. Ωστόσο, σε περίπτωση αποτυχίας για ατομική ενημέρωση, ξαναπροσπαθούμε, διατρέχοντας την λίστα από την αρχή. Οι μετρήσεις που λαμβάνουμε απεικονίζονται ακολούθως:

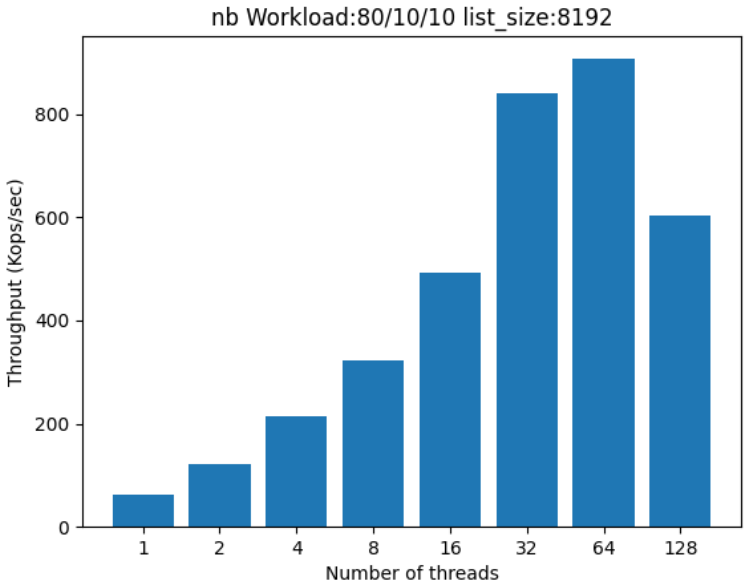


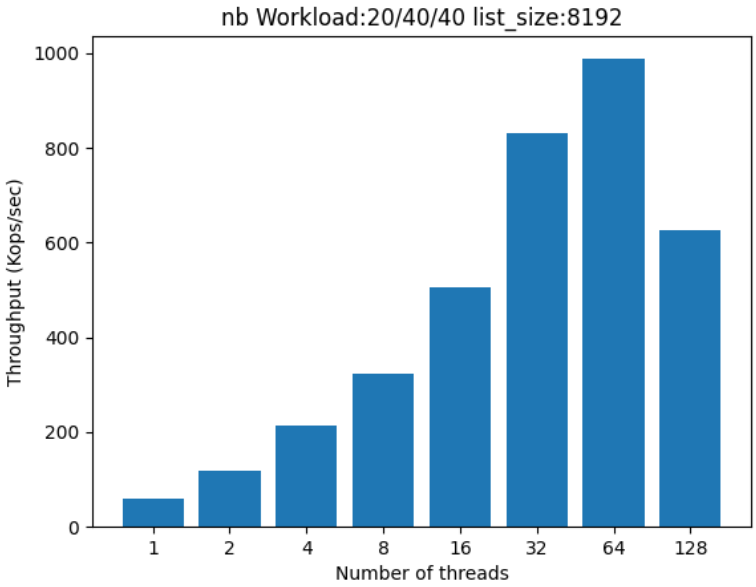


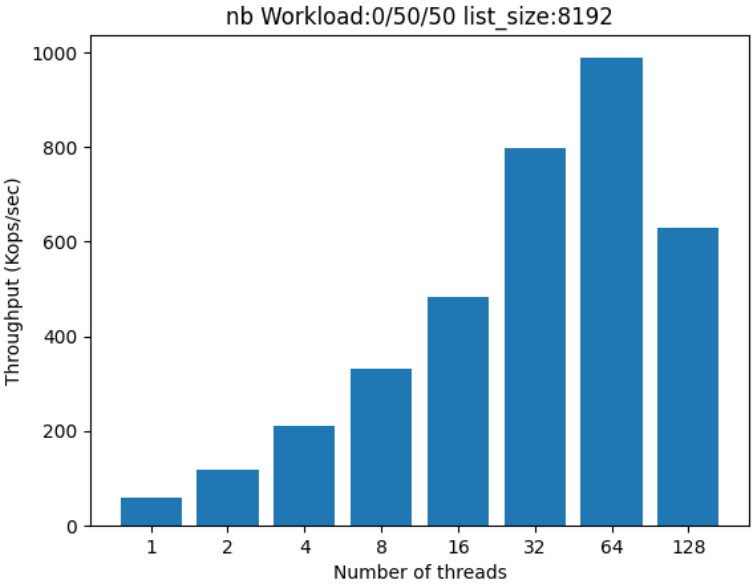












Παρατηρούμε ότι στις περιπτώσεις 100/0/0 και για τα δύο μεγέθη λίστας έχουμε εκθετική αύξηση του throughput σε σχέση με την αύξηση του αριθμού των νημάτων (μέγιστο throughput στα 128 νήματα), μόνο που συνολικά είναι μικρότερο από το αντίστοιχο throughput στην Lazy synchronization υλοποίηση. Εντούτοις, στις υπόλοιπες περιπτώσεις και για τα δύο μεγέθη λίστας, βλέπουμε πως έχουμε μια μικρή αύξηση του throughput για κάθε πλήθος νημάτων (σε σχέση με την Lazy), πράγμα αναμενόμενο αφού αποφύγαμε τη χρήση κλειδωμάτων στις add/remove. Όπως και προηγουμένως, έχουμε σε όλες τις περιπτώσεις (εκτός των 100/0/0) μέγιστο throughput για 64 νήματα, ενώ ακόμη όλες οι περιπτώσεις για μέγεθος λίστας 8192 παρουσιάζουν εκθετική αύξηση στο throughput με την αύξηση των νημάτων.