ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΣΥΣΤΗΜΑΤΑ ΠΑΡΑΛΛΗΛΗΣ ΕΠΕΞΕΡΓΑΣΙΑΣ

ΑΝΑΦΟΡΑ 3ης ΑΣΚΗΣΗΣ

Στοιχεία Ομάδας

- Αναγνωριστικό: parlab05
- Μέλος 1ο: Πέππας Μιχαήλ – Αθανάσιος, Α.Μ: 03121026
- Μέλος 2ο: Σαουνάτσος Ανδρέας, Α.Μ: 03121197
- Ημερομηνία Παράδοσης Αναφοράς: 20.10.2025

- **Ταυτόχρονες Δομές Δεδομένων**

1. Υλοποιήσεις

Στην άσκηση μελετάμε ταυτόχρονες υλοποιήσεις μίας ταξινομημένης απλά συνδεδεμένης λίστας με βασικές λειτουργίες contains(), add() και remove(). Οι υλοποιήσεις διαφέρουν μόνο στον μηχανισμό συγχρονισμού:

Coarse-grain locking (cgl)

Χρησιμοποιείται ένα κοινό lock (mutex) για ολόκληρη τη λίστα. Κάθε λειτουργία κλειδώνει τη δομή στην αρχή και την ξεκλειδώνει στο τέλος, άρα σε κάθε χρονική στιγμή μόνο ένα νήμα μπορεί να εκτελεί οποιαδήποτε λειτουργία στη λίστα.

Fine-grain locking (fgl)

Υπάρχει lock ανά κόμβο και η διάσχιση γίνεται με hand-over-hand (lock coupling): διατηρούνται κλειδωμένοι διαδοχικά ο πρόγονος (pred) και ο τρέχων κόμβος (curr), και καθώς προχωράμε ξεκλειδώνεται ο προηγούμενος κόμβος και κλειδώνεται ο επόμενος. Αυτό επιτρέπει παραλληλία σε διαφορετικά τμήματα της λίστας, αλλά αυξάνει το overhead από πολλά lock/unlock.

Optimistic synchronization (opt)

Οι λειτουργίες πρώτα διατρέχουν τη λίστα χωρίς locks για να εντοπίσουν το σημείο ενημέρωσης. Στη συνέχεια κλειδώνουν τοπικά τους κόμβους pred και curr και εκτελούν validation (επαλήθευση) ότι η δομή δεν άλλαξε ενδιάμεσα. Το validation περιλαμβάνει επαναδιάσχιση από την αρχή για να επιβεβαιωθεί ότι το ζεύγος (pred,curr) παραμένει έγκυρο, αλλιώς η λειτουργία επαναλαμβάνεται (retry).

Lazy synchronization (lazy)

Η contains() εκτελείται χωρίς locks και αγνοεί κόμβους που έχουν μαρκαριστεί ως διαγραμμένοι. Η remove() υλοποιείται σε δύο φάσεις: πρώτα λογική διαγραφή (θέτοντας marked=true) και έπειτα φυσική αφαίρεση (ενημέρωση pred.next=curr.next) με τοπικό locking και validation. Έτσι μειώνεται ο χρόνος που κρατιούνται locks και αποφεύγονται συγκρούσεις με αναζητήσεις.

Non-blocking (lock-free) synchronization (nb)

Δεν χρησιμοποιούνται locks. Οι ενημερώσεις γίνονται με ατομικές εντολές τύπου CAS πάνω σε δείκτες (συχνά «πακετάροντας» και το mark μαζί με το next). Οι λειτουργίες επαναπροσπαθούν (retry) όταν μια CAS αποτύχει λόγω ταυτόχρονης ενημέρωσης από άλλο νήμα. Η contains() είναι wait-free (δεν μπλοκάρει), ενώ add/remove είναι lock-free (πάντα κάποιο νήμα προοδεύει).

2. Περιβάλλον Εκτέλεσης και Ρυθμίσεις Παραμέτρων

Τα πειράματα εκτελέστηκαν στο μηχάνημα sandman (ουρά serial) με χρήση qsub, σύμφωνα με τις οδηγίες της άσκησης. Για κάθε εκτελέσιμο χρησιμοποιήθηκε η ίδια main.c διεπαφή και μετρήθηκε το throughput σε Kops/sec (χιλιάδες λειτουργίες ανά δευτερόλεπτο) για σταθερό χρόνο εκτέλεσης.

Ο αριθμός νημάτων και η αντιστοίχισή τους σε λογικούς πυρήνες καθορίστηκε αποκλειστικά από τη μεταβλητή περιβάλλοντος MT_CONF, ώστε να επιτυγχάνεται pinning και αναπαραγωγιμότητα. Σε εκτελέσεις έως 64 νήματα τα threads «δέθηκαν» σε διαδοχικούς πυρήνες (π.χ. MT_CONF=0,1,2,3 για 4 νήματα). Για 64 και 128 νήματα αξιοποιήθηκε hyperthreading και (στην περίπτωση των 128) oversubscription, σύμφωνα με τις οδηγίες της εκφώνησης.

Οι παράμετροι που εξετάστηκαν ήταν:

- Threads: 1, 2, 4, 8, 16, 32, 64, 128
- Μέγεθος λίστας: 1024, 8192
- Workloads: 100-0-0, 80-10-10, 20-40-40, 0-50-50 (contains-add-remove)

Το run_on_queue.sh φαίνεται, για λόγους πληρότητας, ακολούθως:

a4/conc_ll/run_on_queue.sh

```
#!/bin/bash

## Job Name
#PBS -N run_conc_ll

## Output and error of PBS (not the runs)
#PBS -o run_conc_ll.pbs_out
#PBS -e run_conc_ll.pbs_err

## Sandman, serial queue, 64 threads available
#PBS -q serial
#PBS -l nodes=sandman:ppn=64

## Maximum walltime (adjust if necessary)
#PBS -l walltime=01:00:00

## Go to the directory where qsub was executed
# CHANGE THIS TO YOUR ACTUAL DIRECTORY
cd $HOME/a2/conc_ll

# --- Define Core Parameters ---
IMPLEMENTATIONS="serial cgl fgl opt lazy nb"
NTHREADS="1 2 4 8 16 32 64 128"
LIST_SIZES="1024 8192"

# Workloads: (Contains, Add, Remove)
# Format: "C_A_R"
WORKLOADS="100_0_0 80_10_10 20_40_40 0_50_50"

# Directory for results
OUTDIR="results_conc_ll"
mkdir -p "$OUTDIR"

# --- Helper Function to generate MT_CONF for thread pinning ---
# Generates a comma-separated list of logical core IDs.
# Assumes sandman has 64 logical cores (0-63).
# For N > 64, it cycles through the 64 available logical cores (oversubscription).
get_mt_conf() {
    local N=$1
    local CONFIG=""
    local MAX_LOGICAL_CORES=64

    for i in $(seq 0 $((N - 1))); do
        # Core ID cycles through 0, 1, ..., 63, 0, 1, ...
        local CORE_ID=$((i % MAX_LOGICAL_CORES))

        CONFIG="${CONFIG}${CORE_ID}"
        if [ $i -lt $((N - 1)) ]; then
            CONFIG="${CONFIG},"
        fi
    done
```

```
    echo "$CONFIG"
}

# --- Main Execution Loop ---
for IMPL in $IMPLEMENTATIONS; do
    EXECUTABLE="./x.$IMPL"

    for S in $LIST_SIZES; do

        for T in $NTHREADS; do

            # For the serial implementation, only run T=1 (to establish baseline)
            if [ "$IMPL" == "serial" ] && [ $T -gt 1 ]; then
                continue
            fi

            # --- MT_CONF Setting for Thread Pinning (pthreads) ---
            if [ $T -gt 1 ]; then
                MT_CONF=$(get_mt_conf $T)
                export MT_CONF
            else
                # Unset MT_CONF for single-threaded execution (T=1)
                unset MT_CONF
            fi

            echo "Running $IMPL: ListSize=$S, Nthreads=$T, MT_CONF=$MT_CONF"

            for W in $WORKLOADS; do
                # Split the workload string (e.g., 100_0_0) into C, A, R variables
                IFS='_' read -r C A R <<< "$W"

                # Input arguments for the executable: <list_size> <contains_pct> <add_pct> <remove_pct>
                ARGS="$S $C $A $R"

                # Output files for this run
                OUT="${OUTDIR}/conc_ll_${IMPL}_S${S}_T${T}_W${W}.out"
                ERR="${OUTDIR}/conc_ll_${IMPL}_S${S}_T${T}_W${W}.err"

                # Run the program:
                #  - stdout → OUT
                #  - stderr → ERR
                $EXECUTABLE $ARGS >"$OUT" 2>"$ERR"
            done
        done
    done
done

echo "Execution finished. Results are in the $OUTDIR directory."

```

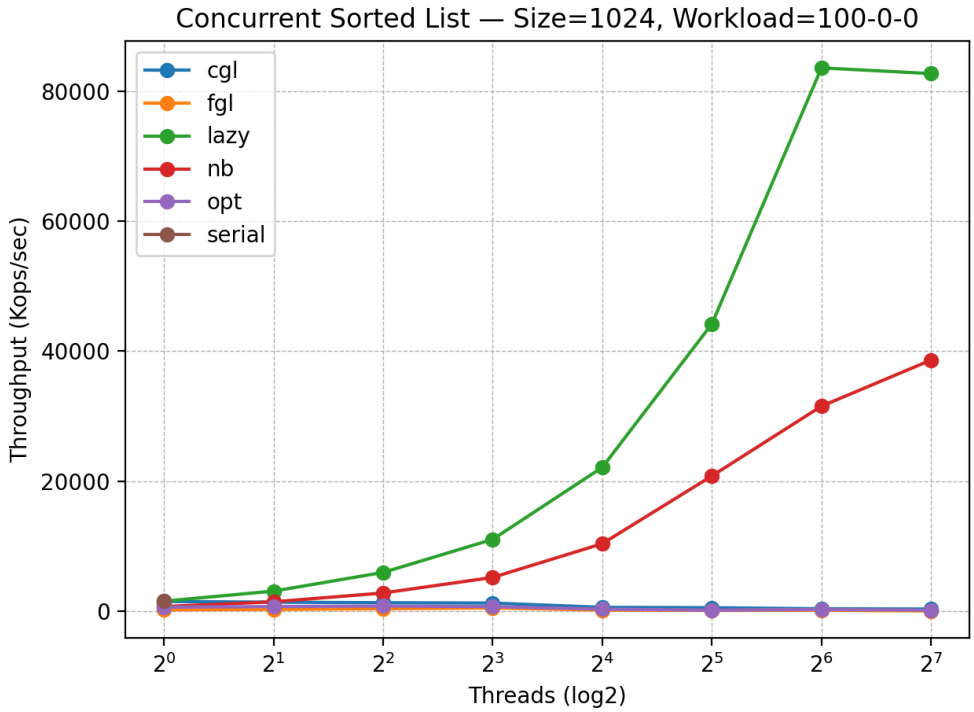
3. Αποτελέσματα και Σχολιασμός

Στα διαγράμματα που ακολουθούν παρουσιάζεται το throughput (Kops/sec), αλλά και το speedup ως συνάρτηση του αριθμού νημάτων, για διαφορετικά μεγέθη λίστας και διαφορετικά workloads. Κάθε γράφημα αντιστοιχεί σε σταθερό συνδυασμό μεγέθους λίστας και workload, ενώ οι καμπύλες αναπαριστούν τις διαφορετικές υλοποιήσεις συγχρονισμού. Για κάθε περίπτωση workload έχουμε :

A. Workload 100-0-0 (μόνο αναζητήσεις)

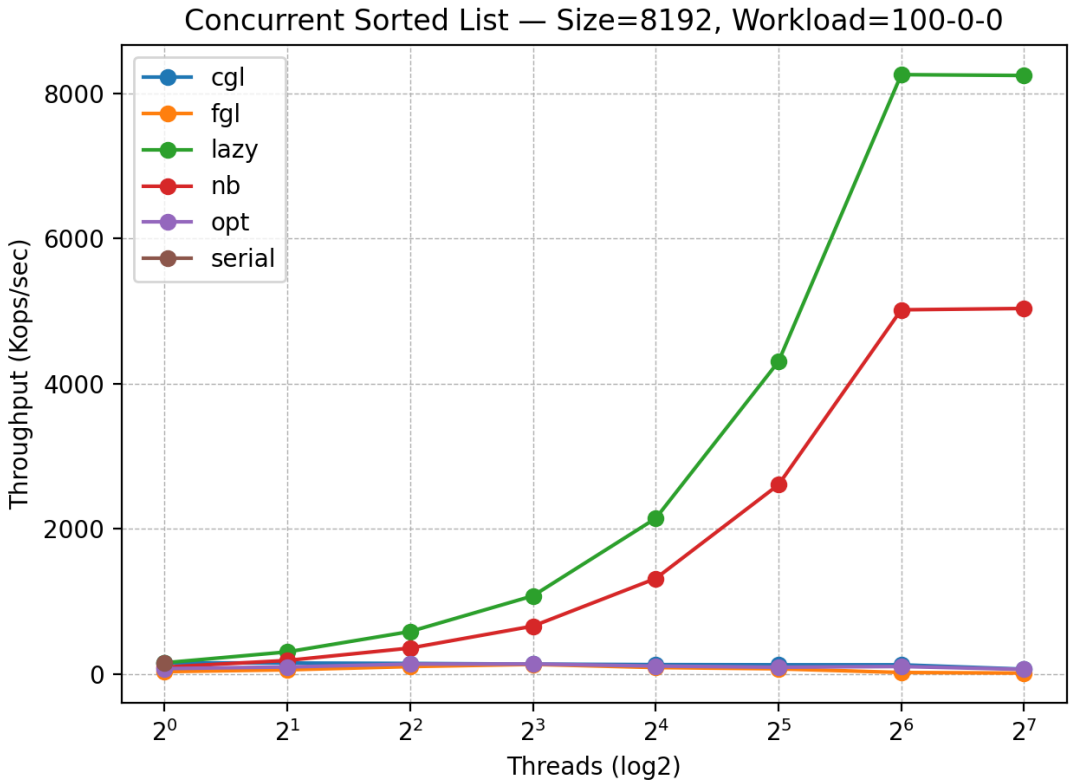
Στο συγκεκριμένο workload όλες οι λειτουργίες είναι contains(), χωρίς καμία εισαγωγή ή διαγραφή. Πρόκειται επομένως για ένα αμιγώς read-only σενάριο, στο οποίο δεν εμφανίζονται συγκρούσεις εγγραφών και η απόδοση εξαρτάται κυρίως από το αν και πόσο blocking απαιτεί η υλοποίηση για τις αναζητήσεις και το κόστος διάσχισης της λίστας (εξαρτώμενο από το μέγεθός της).

Throughput ως προς τον αριθμό νημάτων



Για μέγεθος λίστας S = 1024. Οι υλοποιήσεις lazy synchronization και non-blocking (lock-free) παρουσιάζουν τη σαφώς καλύτερη απόδοση και κλιμακώνονται έντονα με την αύξηση του αριθμού νημάτων. Το lazy synchronization επιτυγχάνει το υψηλότερο throughput, με σχεδόν γραμμική αύξηση έως περίπου 64 νήματα και ελαφρά σταθεροποίηση στη συνέχεια. Η non-blocking υλοποίηση ακολουθεί παρόμοια τάση, αν και με χαμηλότερες απόλυτες τιμές throughput.

Αντίθετα, η coarse-grain locking υλοποίηση εμφανίζει σχεδόν σταθερό και χαμηλό throughput ανεξαρτήτως αριθμού νημάτων, καθώς όλες οι αναζητήσεις σειριοποιούνται μέσω ενός καθολικού lock. Η fine-grain locking παρουσιάζει μικρή βελτίωση σε χαμηλό αριθμό νημάτων, ωστόσο η απόδοσή της υποβαθμίζεται καθώς αυξάνεται το concurrency, λόγω του αυξημένου κόστους από τα πολλαπλά lock/unlock σε κάθε βήμα της διάσχισης. Η optimistic synchronization παραμένει σε ενδιάμεσες τιμές, χωρίς να μπορεί να ανταγωνιστεί τις lazy και non-blocking υλοποιήσεις στο συγκεκριμένο σενάριο.



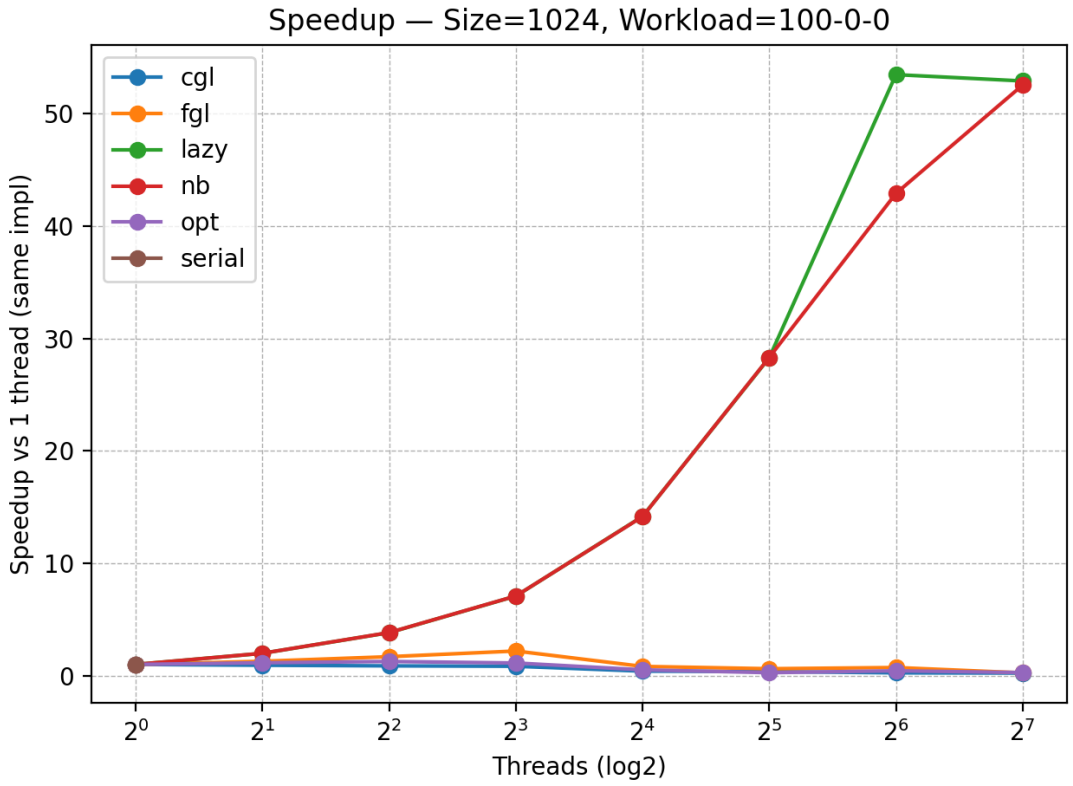
Για μεγαλύτερο μέγεθος λίστας, η ποιοτική εικόνα παραμένει ίδια, αλλά όλες οι υλοποιήσεις παρουσιάζουν σημαντικά χαμηλότερο throughput. Η αύξηση του

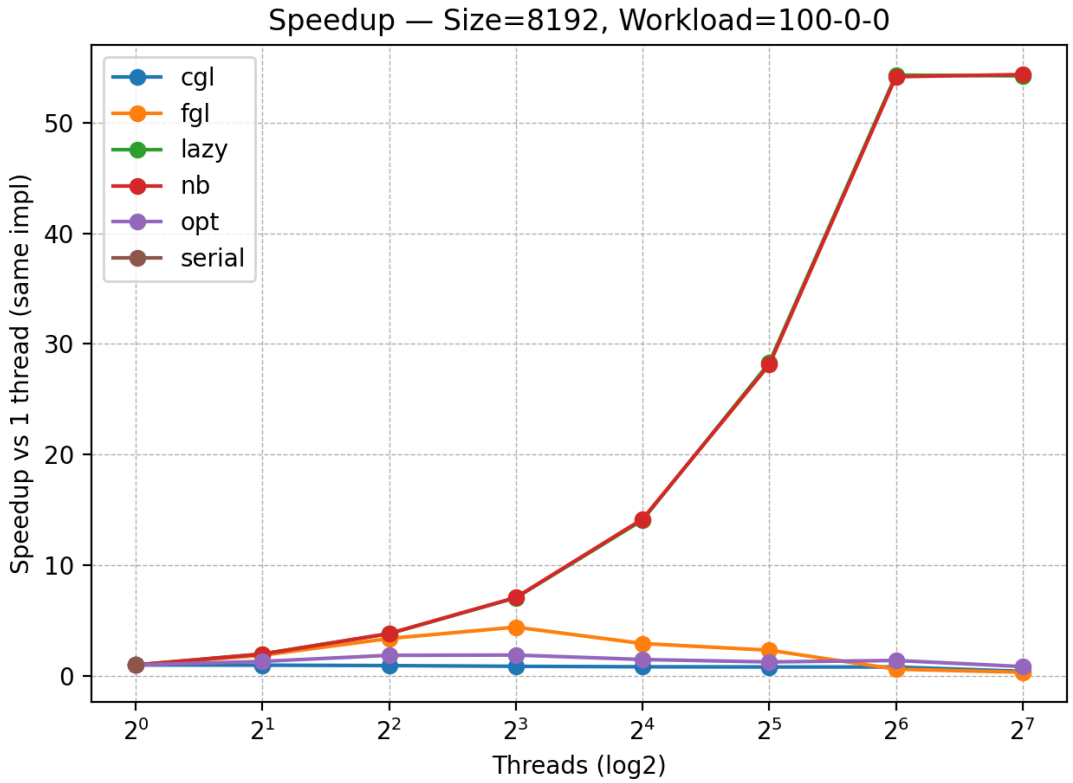
μήκους της λίστας συνεπάγεται μεγαλύτερο κόστος διάσχισης και αυξημένα cache misses, γεγονός που περιορίζει τον ρυθμό εκτέλεσης των αναζητήσεων.

Παρόλα αυτά, οι lazy και non-blocking υλοποιήσεις εξακολουθούν να υπερέχουν σημαντικά έναντι των locking-based προσεγγίσεων, επιβεβαιώνοντας ότι η αποφυγή blocking είναι καθοριστικός παράγοντας απόδοσης σε read-only workloads.

Speedup ως προς τον αριθμό νημάτων

Το speedup υπολογίζεται σε σχέση με την απόδοση ενός νήματος της ίδιας υλοποίησης και αποτυπώνει την ικανότητα κλιμάκωσης ανεξάρτητα από τις απόλυτες τιμές throughput.





Τόσο για S=1024 όσο και για S=8192, οι υλοποιήσεις lazy και non-blocking παρουσιάζουν εντυπωσιακό speedup, φτάνοντας περίπου έως 50× σε 64 νήματα. Πέρα από αυτό το σημείο, και ειδικότερα στα 128 νήματα, η επιτάχυνση σταθεροποιείται, γεγονός που αποδίδεται στη χρήση hyperthreading και oversubscription, όπου τα threads μοιράζονται τους ίδιους φυσικούς πόρους.

Οι υλοποιήσεις coarse-grain, fine-grain και optimistic εμφανίζουν περιορισμένο speedup, με τις καμπύλες τους να παραμένουν κοντά στη μονάδα ή να παρουσιάζουν μικρή μόνο βελτίωση. Αυτό υποδηλώνει ότι το κόστος συγχρονισμού και το contention υπερισχύουν των ωφελειών από την παράλληλη εκτέλεση.

Επίδραση του μεγέθους της λίστας

Η σύγκριση μεταξύ S=1024 και S=8192 δείχνει ότι το μεγαλύτερο μέγεθος λίστας επηρεάζει αρνητικά το throughput σε όλες τις υλοποιήσεις, λόγω αυξημένου κόστους διάσχισης και μνήμης. Ωστόσο, το σχήμα των καμπυλών speedup παραμένει παρόμοιο, γεγονός που υποδηλώνει ότι οι βασικοί περιορισμοί κλιμάκωσης κάθε μηχανισμού συγχρονισμού δεν αλλάζουν με το μέγεθος της λίστας, αλλά σχετίζονται κυρίως με τον τρόπο συγχρονισμού.

Συμπεράσματα για το workload 100-0-0

Στο αμιγώς read-only workload προκύπτουν τα εξής:

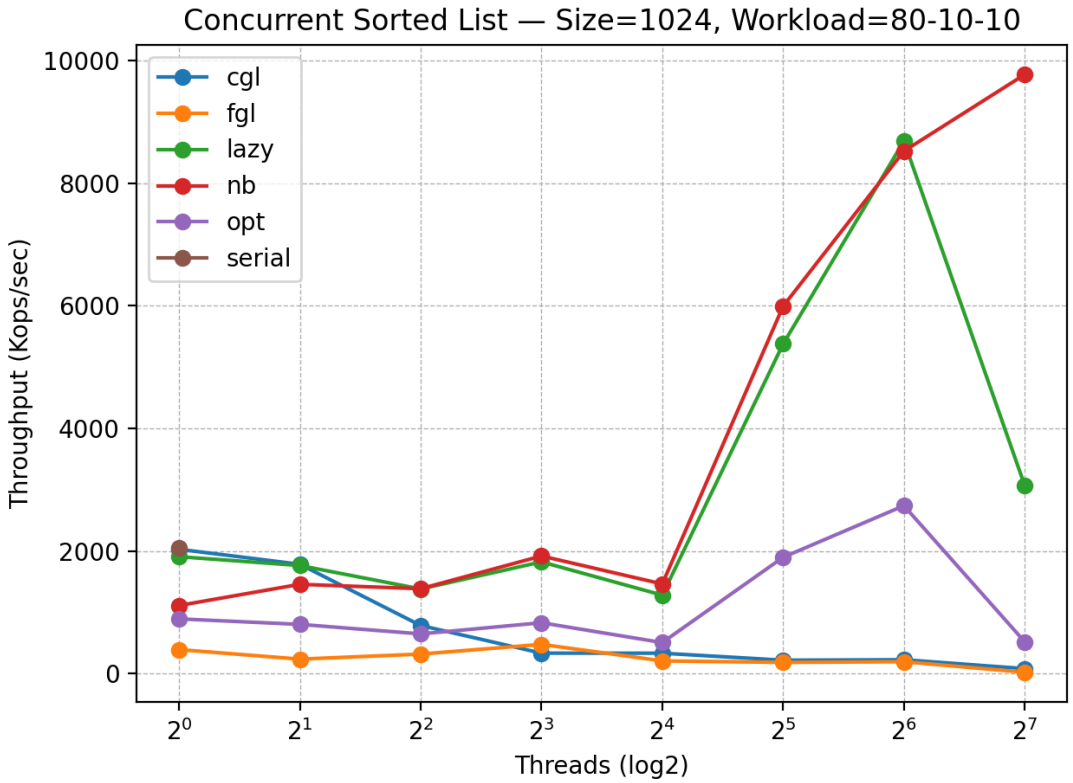
- Οι lazy synchronization και non-blocking υλοποιήσεις είναι οι πλέον κατάλληλες, καθώς οι αναζητήσεις εκτελούνται με ελάχιστο ή μηδενικό blocking.
- Η coarse-grain locking υλοποίηση δεν κλιμακώνεται, λόγω της πλήρους σειριοποίησης των λειτουργιών.
- Οι fine-grain και optimistic υλοποιήσεις παρουσιάζουν καλύτερη συμπεριφορά από την coarse-grain, αλλά παραμένουν κατώτερες σε σχέση με lazy και non-blocking λόγω αυξημένου κόστους συγχρονισμού.
- Η κλιμάκωση περιορίζεται μετά τα 64 νήματα, κυρίως λόγω αρχιτεκτονικών περιορισμών (hyperthreading και oversubscription) και όχι λόγω αλγοριθμικής αστοχίας.

B. Workload 80-10-10 (κυρίως αναζητήσεις)

Στο workload 80-10-10, το 80% των λειτουργιών είναι contains(), ενώ το υπόλοιπο 20% κατανέμεται ισομερώς σε add() και remove(). Το σενάριο αυτό προσομοιώνει ένα ρεαλιστικό read-mostly workload, όπου συνυπάρχουν αναζητήσεις και περιορισμένος αριθμός ενημερώσεων.

Σε αντίθεση με το 100-0-0, εδώ αρχίζουν να εμφανίζονται συγκρούσεις μεταξύ νημάτων λόγω των updates, γεγονός που επιτρέπει να αξιολογηθεί η συμπεριφορά των διαφορετικών μηχανισμών συγχρονισμού υπό μέτριο contention.

Throughput ως προς τον αριθμό νημάτων

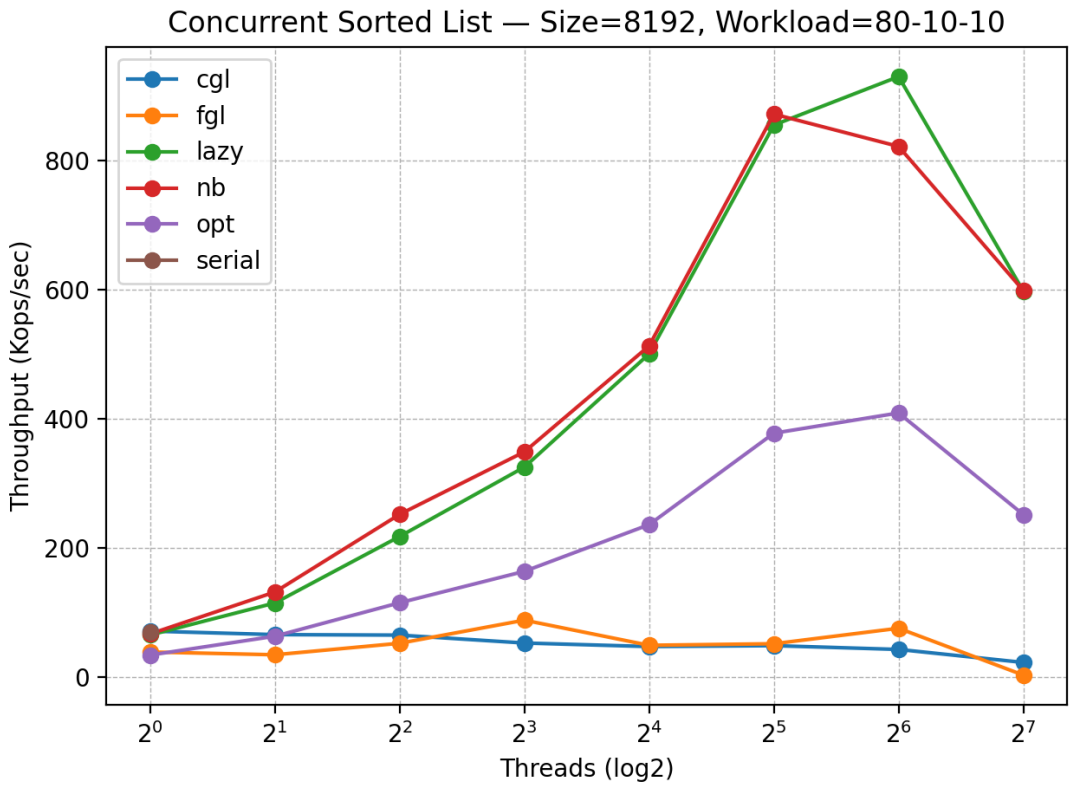


Για μικρό μέγεθος λίστας, η lazy synchronization εμφανίζει την καλύτερη συνολική απόδοση. Το throughput αυξάνεται σταθερά μέχρι τα 32–64 νήματα και στη συνέχεια παρουσιάζει ελαφρά σταθεροποίηση. Η καλή αυτή συμπεριφορά

οφείλεται στο γεγονός ότι οι contains() εκτελούνται χωρίς locks, ενώ οι ενημερώσεις υλοποιούνται με σύντομο τοπικό locking και λογική διαγραφή.

Η optimistic synchronization ακολουθεί σε απόδοση, όμως η αύξηση του αριθμού των νημάτων οδηγεί σε συχνότερα αποτυχημένα validations, με αποτέλεσμα η κλιμάκωση να περιορίζεται νωρίτερα σε σχέση με το lazy. Η non-blocking υλοποίηση παρουσιάζει καλή απόδοση σε χαμηλό και μεσαίο αριθμό νημάτων, αλλά σε υψηλότερο concurrency αρχίζει να επηρεάζεται από αποτυχημένες CAS και αυξημένο contention σε κοινά σημεία της λίστας.

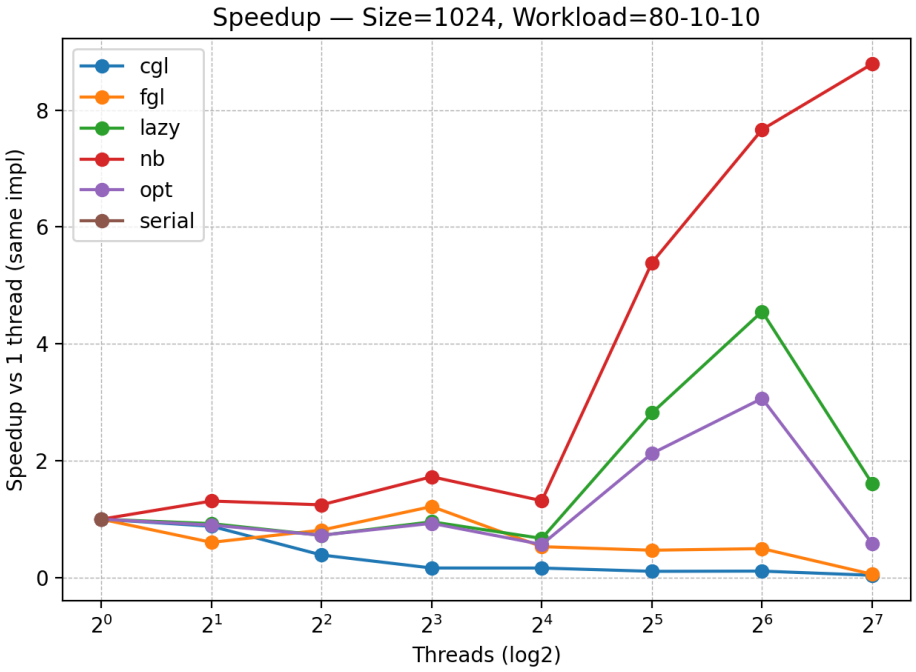
Οι υλοποιήσεις fine-grain και coarse-grain locking εμφανίζουν σαφώς χαμηλότερο throughput. Στην coarse-grain, όλες οι λειτουργίες σειριοποιούνται μέσω ενός καθολικού lock, ενώ στη fine-grain το κόστος των πολλαπλών lock/unlock γίνεται αισθητό ακόμη και με σχετικά περιορισμένο αριθμό ενημερώσεων.

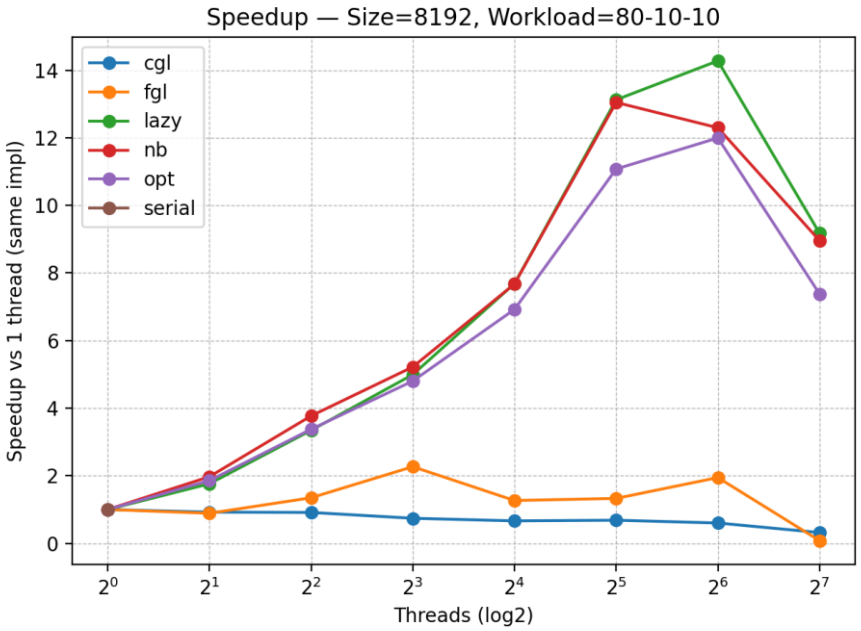


Με τη μεγαλύτερη λίστα, το συνολικό throughput μειώνεται για όλες τις υλοποιήσεις, λόγω αυξημένου κόστους διάσχισης και χειρότερης χωρικής τοπικότητας μνήμης. Παρόλα αυτά, η lazy synchronization διατηρεί ξεκάθαρο προβάδισμα, παρουσιάζοντας την πιο σταθερή συμπεριφορά σε όλο το εύρος των νημάτων.

Η optimistic synchronization επηρεάζεται περισσότερο από το μεγαλύτερο μέγεθος λίστας, καθώς κάθε αποτυχημένο validation συνεπάγεται επαναδιάσχιση μεγαλύτερου τμήματος της δομής. Η non-blocking υλοποίηση συνεχίζει να κλιμακώνει έως ένα σημείο, αλλά ο κορεσμός εμφανίζεται νωρίτερα σε σχέση με το S=1024.

<u>Speedup ως προς τον αριθμό νημάτων</u>





Η ανάλυση του speedup δείχνει ότι:

- Η lazy synchronization παρουσιάζει την καλύτερη κλιμάκωση και στα δύο μεγέθη λίστας, με σχεδόν γραμμική αύξηση έως τα 32–64 νήματα.
- Η optimistic και η non-blocking υλοποίηση εμφανίζουν μέτριο speedup, το οποίο περιορίζεται καθώς αυξάνονται τα retries (λόγω validation ή CAS αποτυχιών).
- Οι locking-based υλοποιήσεις (coarse και fine-grain) παρουσιάζουν περιορισμένη επιτάχυνση, με τις καμπύλες speedup να παραμένουν χαμηλά.
- Σε όλα τα σχήματα, η μετάβαση από 64 σε 128 νήματα δεν οδηγεί σε ουσιαστική επιπλέον επιτάχυνση, γεγονός που αποδίδεται στη χρήση hyperthreading και oversubscription.

Επίδραση του μεγέθους της λίστας

Η αύξηση του μεγέθους της λίστας από 1024 σε 8192 στοιχεία:

- Μειώνει το απόλυτο throughput σε όλες τις υλοποιήσεις.
- Επηρεάζει δυσανάλογα τις optimistic και fine-grain υλοποιήσεις, όπου το κόστος αποτυχημένων προσπαθειών (validation ή locks) αυξάνεται με το μήκος της διάσχισης.

Παρόλα αυτά, η σχετική κατάταξη των υλοποιήσεων παραμένει σταθερή, με τη lazy synchronization να αποτελεί την πιο ανθεκτική επιλογή ως προς την αύξηση του μεγέθους της λίστας.

Συμπεράσματα για το workload 80-10-10

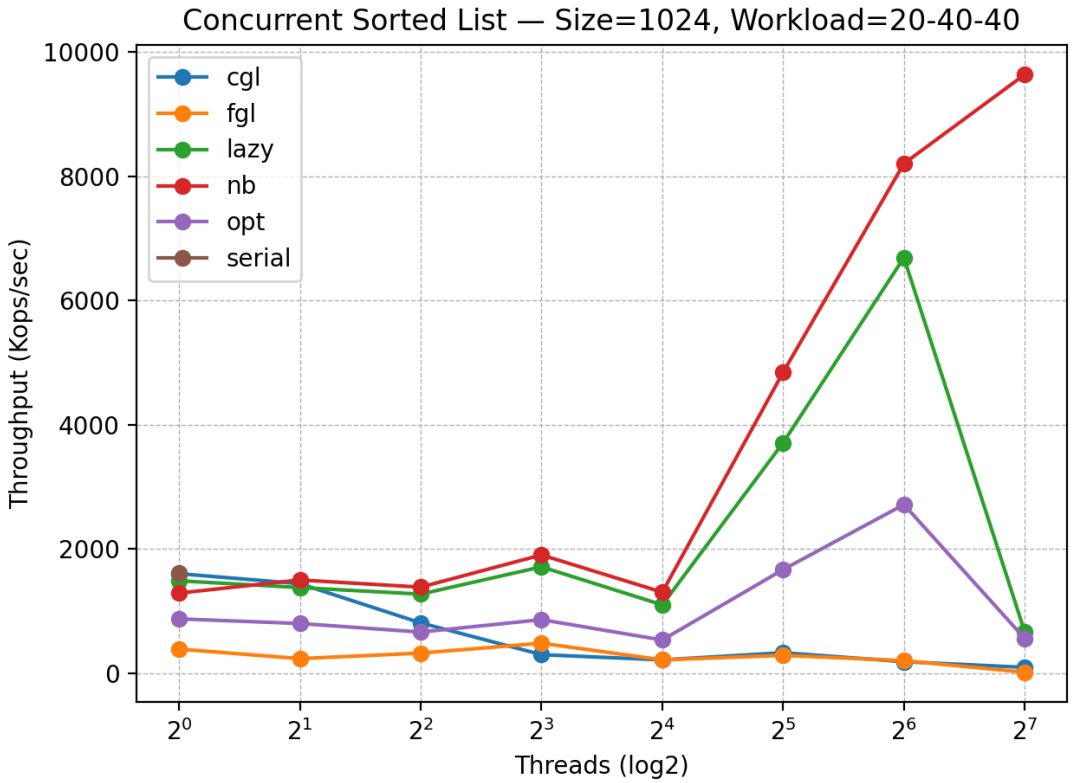
Από την ανάλυση του workload 80-10-10 προκύπτουν τα εξής:

- Η lazy synchronization αποτελεί την πιο αποδοτική και σταθερή υλοποίηση σε read-mostly σενάρια με περιορισμένα updates.
- Η optimistic synchronization λειτουργεί ικανοποιητικά, αλλά η απόδοσή της περιορίζεται από τα αποτυχημένα validations όσο αυξάνεται το concurrency.
- Η non-blocking υλοποίηση παρουσιάζει καλό scaling σε μέτριο αριθμό νημάτων, αλλά εμφανίζει κορεσμό σε υψηλό contention.
- Οι coarse-grain και fine-grain locking υλοποιήσεις υστερούν σημαντικά, επιβεβαιώνοντας ότι το blocking και το lock overhead επηρεάζουν αρνητικά την απόδοση ακόμη και όταν τα updates είναι σχετικά λίγα.

Γ. Workload 20-40-40 (κυρίως ενημερώσεις)

Στο workload 20-40-40, μόνο το 20% των λειτουργιών είναι contains(), ενώ το 80% αφορά ενημερώσεις (add() και remove()). Πρόκειται για ένα update-dominated σενάριο, στο οποίο το contention στη δομή δεδομένων είναι έντονο και η αποδοτικότητα των μηχανισμών συγχρονισμού παίζει καθοριστικό ρόλο.

Σε αντίθεση με τα read-mostly workloads, εδώ αναδεικνύεται το κόστος των locks, των αποτυχημένων validations και των επαναλαμβανόμενων atomic retries.

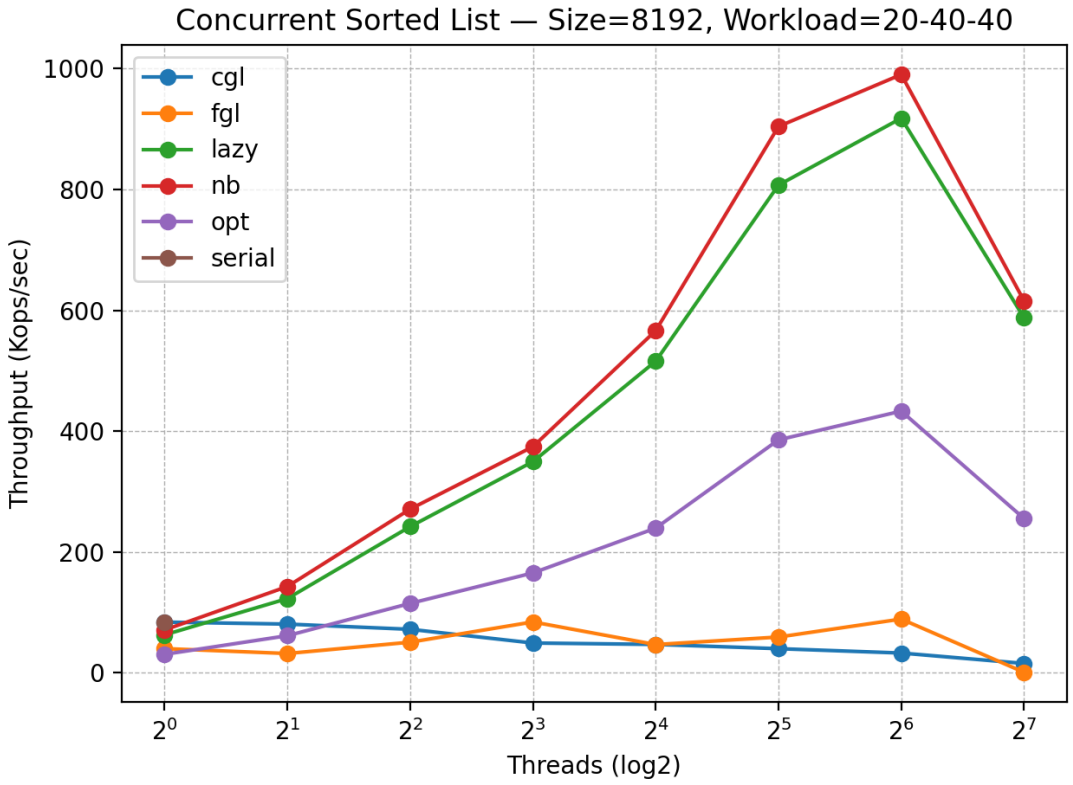


Για μικρό μέγεθος λίστας, η non-blocking (lock-free) υλοποίηση επιτυγχάνει το υψηλότερο throughput, παρουσιάζοντας σαφή υπεροχή σε μεγάλο εύρος αριθμού νημάτων. Το throughput αυξάνεται έντονα έως τα 64 νήματα, γεγονός που υποδηλώνει ότι η αποφυγή locks επιτρέπει μεγαλύτερο βαθμό ταυτόχρονης προόδου ακόμη και υπό υψηλό contention.

Η lazy synchronization ακολουθεί σε απόδοση, με καλή κλιμάκωση έως τα 32–64 νήματα. Παρότι χρησιμοποιεί locks στις ενημερώσεις, ο διαχωρισμός λογικής και φυσικής διαγραφής μειώνει τη διάρκεια κατοχής locks και περιορίζει τις συγκρούσεις.

Η optimistic synchronization παρουσιάζει αισθητά χαμηλότερο throughput. Η συχνότητα των αποτυχημένων validations αυξάνεται λόγω των πολλών updates, με αποτέλεσμα σημαντικό ποσοστό του χρόνου εκτέλεσης να αναλώνεται σε επαναλήψεις (retries).

Οι υλοποιήσεις fine-grain και coarse-grain locking εμφανίζουν τη χαμηλότερη απόδοση. Στην coarse-grain, όλες οι ενημερώσεις σειριοποιούνται, ενώ στη fine-grain το κόστος από τα πολλαπλά locks σε κάθε ενημέρωση επιβαρύνει έντονα το throughput.



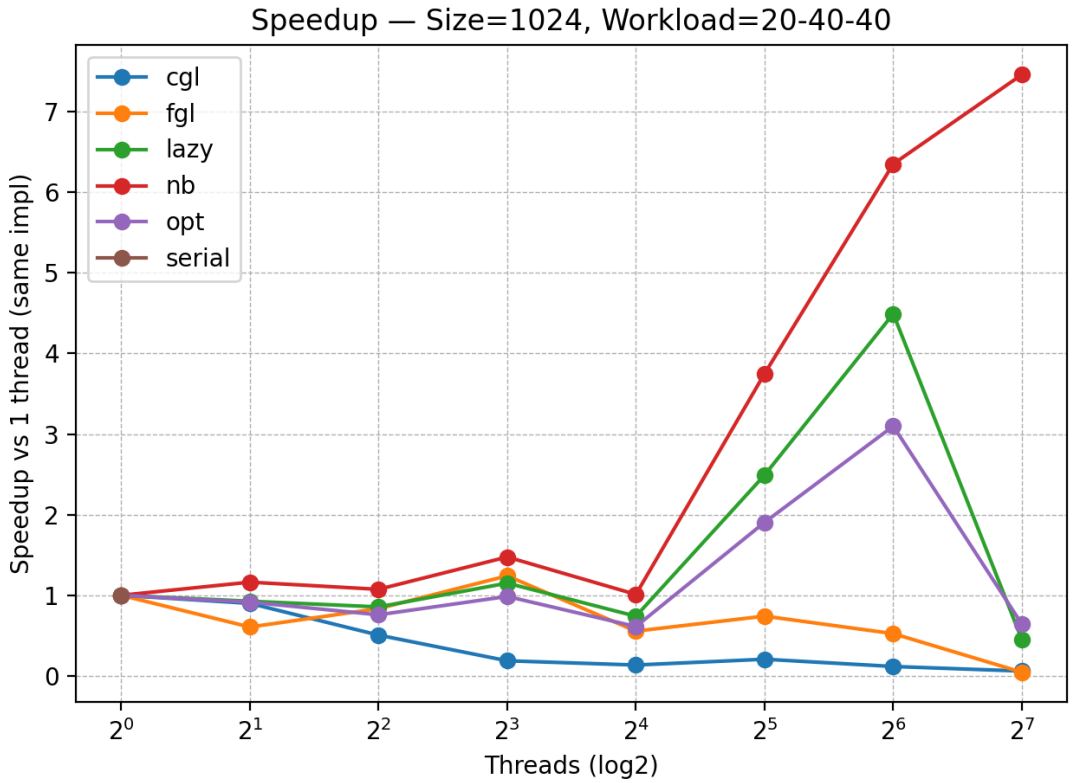
Για μεγαλύτερο μέγεθος λίστας, το throughput μειώνεται σε όλες τις υλοποιήσεις, καθώς οι ενημερώσεις απαιτούν μεγαλύτερες διασχίσεις και αυξάνεται το memory contention.

Η non-blocking υλοποίηση παραμένει η καλύτερη σε απόλυτους όρους, φτάνοντας σε μέγιστο throughput γύρω στα 32–64 νήματα, πριν εμφανίσει πτώση στα 128 νήματα. Η πτώση αυτή αποδίδεται στον συνδυασμό αυξημένου CAS contention και hyperthreading/oversubscription.

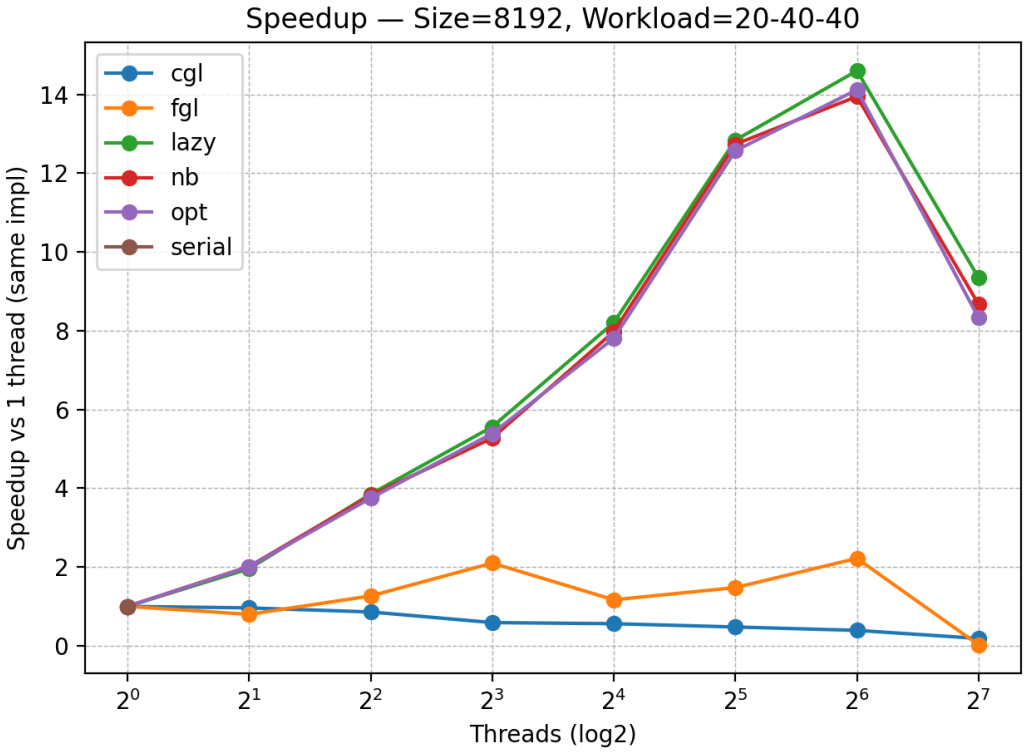
Η lazy synchronization διατηρεί σταθερά καλή απόδοση, αν και υστερεί ελαφρώς έναντι της non-blocking σε αυτό το έντονα update-heavy σενάριο. Η optimistic synchronization επηρεάζεται ακόμη περισσότερο από το αυξημένο μήκος της λίστας, καθώς κάθε αποτυχημένο validation συνεπάγεται επαναδιάσχιση μεγαλύτερου τμήματος της δομής.

Speedup ως προς τον αριθμό νημάτων

Η ανάλυση του speedup αποκαλύπτει σημαντικές διαφορές στη δυνατότητα κλιμάκωσης



Για S = 1024, η non-blocking υλοποίηση επιτυγχάνει τη μεγαλύτερη επιτάχυνση, με speedup που ξεπερνά κατά πολύ τις υπόλοιπες υλοποιήσεις. Η lazy synchronization παρουσιάζει επίσης αξιόλογο speedup, αν και σαφώς χαμηλότερο.



Για S = 8192, τόσο η non-blocking όσο και η lazy εμφανίζουν μέγιστο speedup περίπου στα 32–64 νήματα, με αισθητή πτώση στα 128 νήματα.

Οι coarse-grain και fine-grain locking υλοποιήσεις παρουσιάζουν speedup μικρότερο της μονάδας σε αρκετά σημεία, υποδεικνύοντας ότι η προσθήκη νημάτων όχι μόνο δεν επιταχύνει, αλλά επιβαρύνει την εκτέλεση λόγω έντονου contention.

Επίδραση του μεγέθους της λίστας

Η σύγκριση μεταξύ S=1024 και S=8192 δείχνει ότι το μεγαλύτερο μέγεθος λίστας:

- Μειώνει το throughput σε όλες τις υλοποιήσεις.
- Εντείνει ιδιαίτερα το κόστος των retries σε optimistic και non-blocking προσεγγίσεις.
- Καθιστά πιο εμφανές το πλεονέκτημα της lazy synchronization, η οποία περιορίζει τη διάρκεια κρίσιμων τμημάτων.

Παρότι το absolute throughput μειώνεται, η σχετική κατάταξη των υλοποιήσεων παραμένει παρόμοια, με τις non-blocking και lazy να υπερέχουν σε σχέση με τις locking-based λύσεις.

Συμπεράσματα για το workload 20-40-40

Από την ανάλυση του update-heavy workload προκύπτουν τα εξής:

- Η non-blocking (lock-free) υλοποίηση επιτυγχάνει την υψηλότερη απόδοση, ιδιαίτερα σε μικρό και μεσαίο μέγεθος λίστας, επιβεβαιώνοντας το πλεονέκτημα της απουσίας locks σε περιβάλλον έντονων ενημερώσεων.
- Η lazy synchronization αποτελεί τον καλύτερο συμβιβασμό μεταξύ απόδοσης και σταθερότητας, ειδικά όταν αυξάνεται το μέγεθος της λίστας.
- Η optimistic synchronization υποφέρει σημαντικά λόγω συχνών αποτυχημένων validations και δεν ενδείκνυται για update-dominated workloads.
- Οι coarse-grain και fine-grain locking υλοποιήσεις παρουσιάζουν πολύ περιορισμένη κλιμάκωση και αποτελούν τις λιγότερο αποδοτικές επιλογές στο συγκεκριμένο σενάριο.

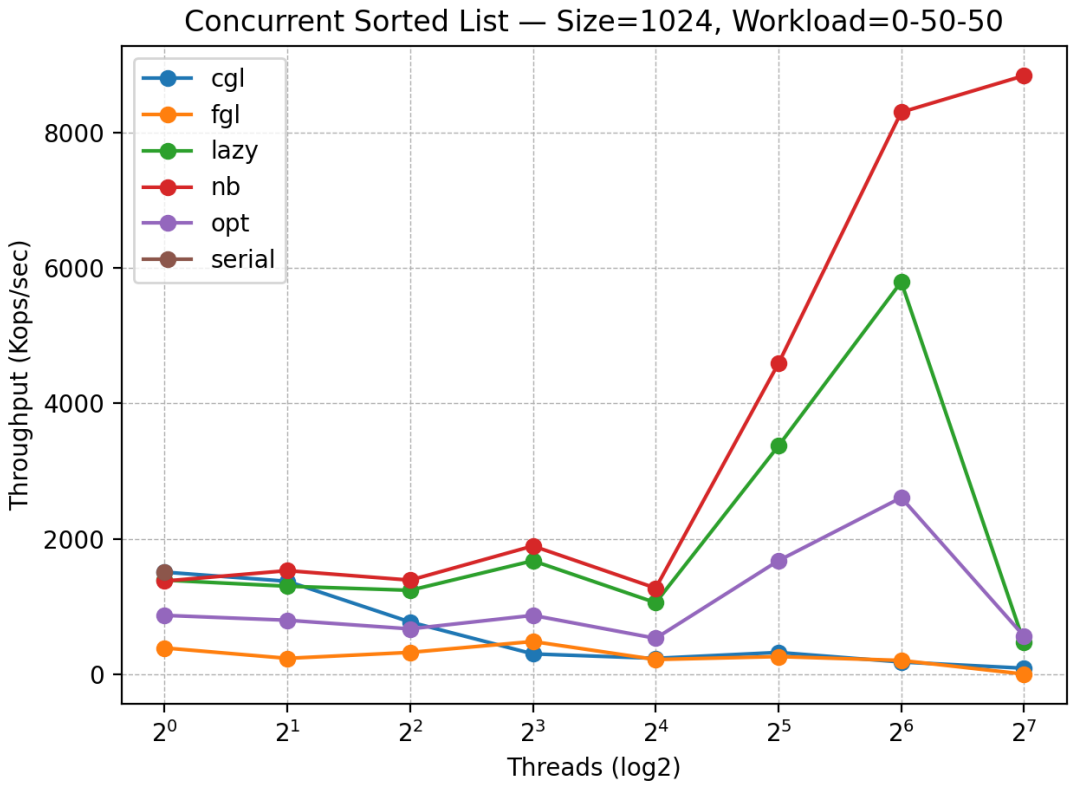
Η πτώση της απόδοσης στα 128 νήματα οφείλεται κυρίως σε αρχιτεκτονικούς περιορισμούς (hyperthreading και oversubscription), σε συνδυασμό με αυξημένο contention σε atomic ή locking μηχανισμούς.

Δ. Workload 0-50-50 (μόνο ενημερώσεις)

Στο workload 0-50-50 όλες οι λειτουργίες είναι ενημερώσεις (add() και remove()), χωρίς καθόλου αναζητήσεις. Πρόκειται για το πιο απαιτητικό σενάριο από πλευράς συγχρονισμού, καθώς κάθε λειτουργία τροποποιεί τη δομή της λίστας και συνεπώς δημιουργείται έντονο contention τόσο σε locks όσο και σε atomic operations.

Το workload αυτό αναδεικνύει καθαρά τις διαφορές μεταξύ blocking, optimistic και lock-free προσεγγίσεων.

<u>Throughput ως προς τον αριθμό νημάτων</u>

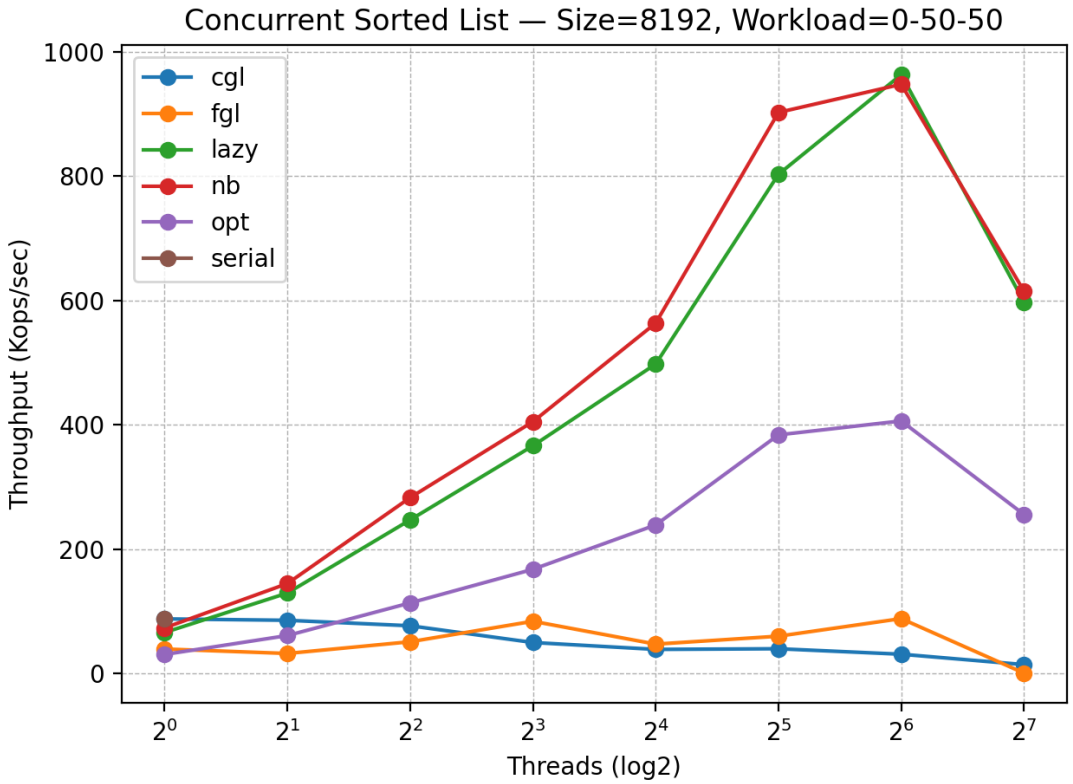


Για μικρό μέγεθος λίστας, η non-blocking (lock-free) υλοποίηση επιτυγχάνει το υψηλότερο throughput σε όλο το εύρος των νημάτων. Το throughput αυξάνεται σημαντικά έως τα 64 νήματα, όπου και παρατηρείται η μέγιστη απόδοση. Η απουσία locks επιτρέπει στα νήματα να προοδεύουν χωρίς να μπλοκάρουν μεταξύ τους, ακόμη και όταν όλες οι λειτουργίες είναι ενημερώσεις.

Η lazy synchronization ακολουθεί σε απόδοση, παρουσιάζοντας καλή κλιμάκωση έως τα 32–64 νήματα. Ο διαχωρισμός της λογικής από τη φυσική διαγραφή μειώνει το contention στα κρίσιμα τμήματα, αν και το κόστος των locks παραμένει αισθητό σε σύγκριση με την lock-free προσέγγιση.

Η optimistic synchronization εμφανίζει μέτριο throughput. Τα συχνά updates οδηγούν σε αποτυχημένα validations και retries, τα οποία περιορίζουν την απόδοση, αν και παραμένει σαφώς ανώτερη από τις locking-based υλοποιήσεις.

Οι fine-grain και coarse-grain locking υλοποιήσεις παρουσιάζουν τη χαμηλότερη απόδοση. Στην coarse-grain, όλες οι ενημερώσεις σειριοποιούνται, ενώ στη fine-grain το κόστος από τα πολλαπλά locks σε κάθε ενημέρωση επιβαρύνει έντονα το throughput.



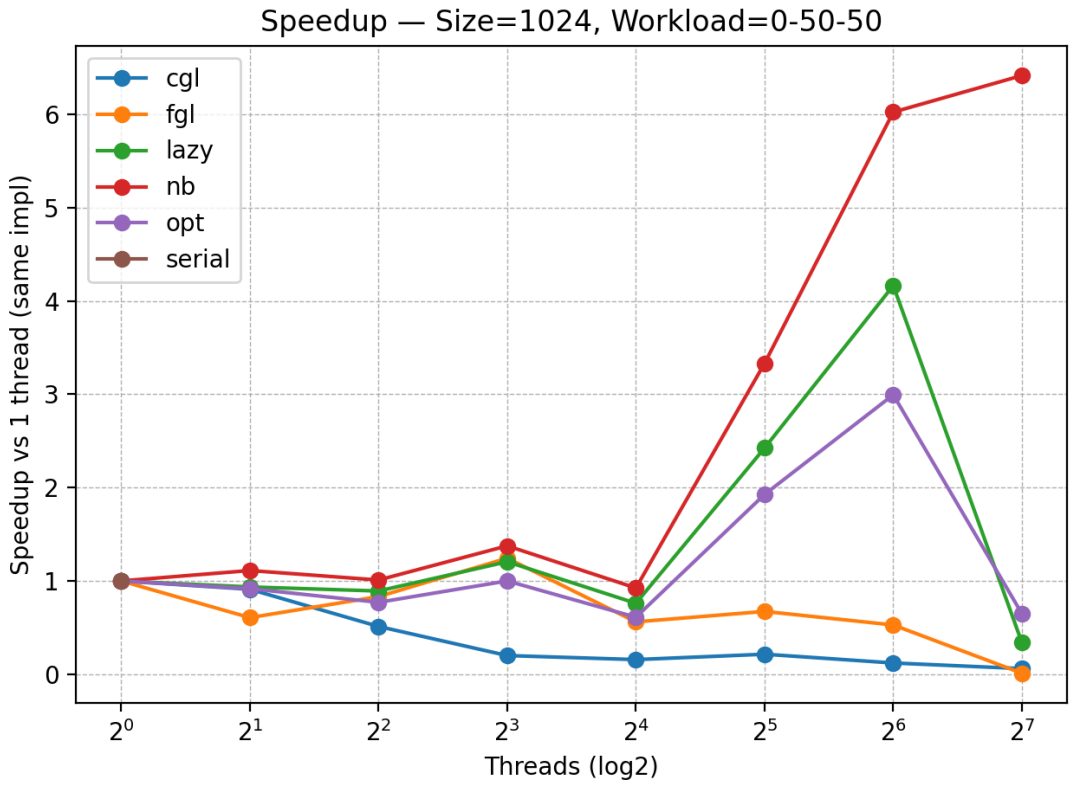
Με μεγαλύτερη λίστα, το throughput μειώνεται για όλες τις υλοποιήσεις, λόγω αυξημένου κόστους διάσχισης και μεγαλύτερης πιθανότητας συγκρούσεων.

Η non-blocking υλοποίηση παραμένει η ταχύτερη, φτάνοντας σε μέγιστο throughput γύρω στα 32–64 νήματα, πριν εμφανίσει πτώση στα 128 νήματα. Η πτώση αυτή αποδίδεται στον συνδυασμό έντονου CAS contention και oversubscription.

Η lazy synchronization παρουσιάζει παρόμοια ποιοτική συμπεριφορά, αλλά με χαμηλότερο απόλυτο throughput σε σχέση με τη non-blocking. Η optimistic synchronization επηρεάζεται ιδιαίτερα από το αυξημένο μέγεθος της λίστας, καθώς κάθε αποτυχημένο validation συνεπάγεται επαναδιάσχιση μεγαλύτερου τμήματος της δομής.

Speedup ως προς τον αριθμό νημάτων

Η ανάλυση του speedup επιβεβαιώνει τα συμπεράσματα από το throughput:

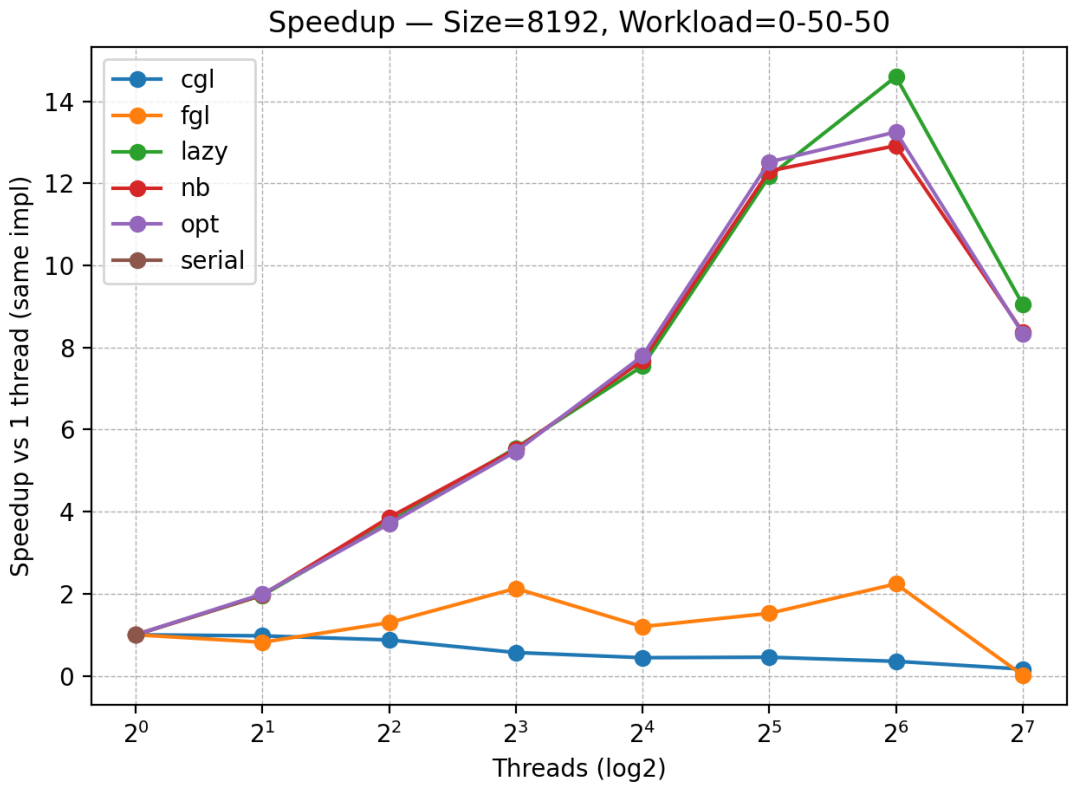


Για S = 1024, η non-blocking υλοποίηση παρουσιάζει τη μεγαλύτερη επιτάχυνση, ξεπερνώντας όλες τις άλλες υλοποιήσεις σε όλο το εύρος των νημάτων.

Η lazy synchronization επιτυγχάνει αξιόλογο speedup, αν και χαμηλότερο από τη non-blocking, λόγω του κόστους των locks στις ενημερώσεις.

Η optimistic synchronization εμφανίζει περιορισμένο speedup, καθώς τα retries ακυρώνουν μέρος του οφέλους από την παράλληλη εκτέλεση.

Οι coarse-grain και fine-grain locking υλοποιήσεις παρουσιάζουν speedup μικρότερο της μονάδας σε αρκετές περιπτώσεις, υποδεικνύοντας ότι η αύξηση του αριθμού νημάτων επιβαρύνει την εκτέλεση.



Για S = 8192, όλες οι καμπύλες speedup εμφανίζουν κορεσμό ή πτώση στα 128 νήματα, γεγονός που αποδίδεται στη χρήση hyperthreading και oversubscription σε συνδυασμό με έντονο contention.

<u>Επίδραση του μεγέθους της λίστας</u>

Η αύξηση του μεγέθους της λίστας από 1024 σε 8192 στοιχεία:

- Μειώνει σημαντικά το throughput σε όλες τις υλοποιήσεις.
- Εντείνει το κόστος των retries τόσο στις optimistic όσο και στις non-blocking υλοποιήσεις.
- Καθιστά πιο εμφανές το πλεονέκτημα της lazy synchronization έναντι των locking-based λύσεων, λόγω της μείωσης του χρόνου κατοχής locks.

Παρά τη μείωση των απόλυτων τιμών, η σχετική κατάταξη των υλοποιήσεων παραμένει σταθερή, με τις non-blocking και lazy να υπερέχουν.

Συμπεράσματα για το workload 0-50-50

Από την ανάλυση του αμιγώς update-heavy workload προκύπτουν τα εξής:

- Η non-blocking (lock-free) υλοποίηση αποτελεί την πιο αποδοτική επιλογή, επιτυγχάνοντας το υψηλότερο throughput και speedup.
- Η lazy synchronization προσφέρει τον καλύτερο συμβιβασμό μεταξύ απόδοσης και σταθερότητας, ειδικά σε μεγαλύτερα μεγέθη λίστας.
- Η optimistic synchronization δεν ενδείκνυται για workloads με αποκλειστικά ενημερώσεις, λόγω συχνών αποτυχημένων validations.
- Οι coarse-grain και fine-grain locking υλοποιήσεις αποτυγχάνουν να κλιμακώσουν και εμφανίζουν έντονη υποβάθμιση της απόδοσης.

4. Γενικό Συμπέρασμα

Από τη μελέτη των διαφορετικών workloads προκύπτει ότι η απόδοση και η κλιμάκωση μιας ταυτόχρονης ταξινομημένης λίστας εξαρτώνται άμεσα από το ποσοστό ενημερώσεων και τον μηχανισμό συγχρονισμού. Οι υλοποιήσεις με coarse-grain και fine-grain locking παρουσιάζουν περιορισμένη κλιμάκωση λόγω blocking και αυξημένου lock overhead, ανεξάρτητα από το workload. Η optimistic synchronization αποδίδει καλά μόνο σε read-heavy σενάρια, αλλά υποβαθμίζεται σημαντικά όταν αυξάνονται οι ενημερώσεις λόγω συχνών retries.

Η lazy synchronization εμφανίζει τη πιο σταθερή και ισορροπημένη συμπεριφορά σε όλα τα workloads, ενώ η non-blocking (lock-free) υλοποίηση επιτυγχάνει την υψηλότερη απόδοση σε update-heavy σενάρια, με κόστος αυξημένο CAS contention σε υψηλό αριθμό νημάτων. Τέλος, η εμφάνιση κορεσμού ή πτώσης της απόδοσης στα 64–128 νήματα αποδίδεται κυρίως σε αρχιτεκτονικούς περιορισμούς (hyperthreading και oversubscription) και όχι στους ίδιους τους αλγορίθμους.

Σ.Η.Μ.Μ.Υ. Ε.Μ.Π.
Δεκέμβριος 2025