ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΣΥΣΤΗΜΑΤΑ ΠΑΡΑΛΛΗΛΗΣ ΕΠΕΞΕΡΓΑΣΙΑΣ

ΑΝΑΦΟΡΑ 2ης ΑΣΚΗΣΗΣ

Στοιχεία Ομάδας

- Αναγνωριστικό: parlab05
- Μέλος 1ο: Πέππας Μιχαήλ – Αθανάσιος, Α.Μ: 03121026
- Μέλος 2ο: Σαουνάτσος Ανδρέας, Α.Μ: 03121197
- Ημερομηνία Παράδοσης Αναφοράς: 25.11.2025

- **Αμοιβαίος Αποκλεισμός – Κλειδώματα**

1. Εισαγωγή και Αρχεία

Στην παρούσα άσκηση μελετάμε διάφορους μηχανισμούς locks σε πολυνηματικές εφαρμογές, μέσω του αλγορίθμου ταξινόμησης K-means (στη shared έκδοσή του). Στόχος μας, είναι να κατανοήσουμε πώς διαφορετικές στρατηγικές συγχρονισμού (απουσία κλειδώματος, κλασικά mutex και spinlocks, locks τύπου TAS/TTAS, ιεραρχικά κλειδώματα τύπου array και CLH, καθώς και η εντολή critical του OpenMP) επηρεάζουν τον χρόνο εκτέλεσης και τη δυνατότητα κλιμάκωσης της εφαρμογής, όταν αυτή εκτελείται σε ένα πολυπύρηνο σύστημα με μέχρι και 64 λογικά νήματα (το sandman έχει 4x8=32 φυσικούς πυρήνες και 32x2=64 λογικούς). Το κυρίως σώμα του προγράμματος παραμένει το ίδιο και μεταβάλλεται μόνο ο μηχανισμός του lock, ώστε να μπορούμε να απομονώσουμε και να συγκρίνουμε το καθαρό κόστος συγχρονισμού κάθε κλειδώματος.

Ο δοσμένος κώδικας βασίζεται στον K-means από την προηγούμενη άσκηση και αποτελείται από τα κοινά αρχεία file_io.c και util.c, το header file kmeans.h και το πρόγραμμα main.c, το οποίο αναλαμβάνει να διαβάσει τις παραμέτρους και να καλέσει την υλοποίηση του αλγορίθμου. Το αρχείο seq_kmeans.c περιέχει τη σειριακή έκδοση του K-means, ενώ το omp_naive_kmeans.c παρέχει μια απλή παράλληλη προσέγγιση με OpenMP. Για τη μελέτη των κλειδωμάτων χρησιμοποιούνται δύο επιπλέον αρχεία: το omp_critical_kmeans.c, όπου ο συγχρονισμός υλοποιείται με #pragma omp critical, και το omp_lock_kmeans.c, το οποίο ορίζει την παράλληλη εκδοχή που βασίζεται σε μια διεπαφή με κλειδώματα, ως παράμετρο. Η υλοποίηση των επιμέρους κλειδωμάτων γίνεται στον φάκελο locks/, μέσω των αρχείων nosync_lock.c, pthread_mutex_lock.c, pthread_spin_lock.c, tas_lock.c, ttas_lock.c, array_lock.c και clh_lock.c, τα οποία μοιράζονται το κοινό header lock.h (ορισμός του τύπου lock_t και των συναρτήσεων lock_init, lock_acquire, lock_release, lock_free) και το αρχείο alloc.h για τις βοηθητικές δεσμεύσεις μνήμης. Έτσι, με κατάλληλο linking στο Makefile προκύπτουν τα διαφορετικά εκτελέσιμα (kmeans_omp_nosync_lock, kmeans_omp_pthread_mutex_lock, kmeans_omp_pthread_spin_lock, kmeans_omp_tas_lock, kmeans_omp_ttas_lock, kmeans_omp_array_lock, kmeans_omp_clh_lock), τα οποία μοιράζονται τον ίδιο κώδικα του K-means αλλά χρησιμοποιούν διαφορετικό μηχανισμό lock.

Το Makefile του φακέλου έχει προσαρμοστεί ώστε να συνθέτει όλα τα παραπάνω εκτελέσιμα, αξιοποιώντας τις κοινές πηγές file_io.c και util.c και προσθέτοντας κάθε φορά την αντίστοιχη υλοποίηση κλειδώματος από τον υποφάκελο locks/. Για τα κλειδώματα που βασίζονται σε POSIX mutex ή spinlocks ενεργοποιείται επιπλέον η παράμετρος -pthread, ενώ όλα τα παράλληλα εκτελέσιμα μεταγλωττίζονται με τις σημαίες του OpenMP (-fopenmp). Παράλληλα, το πρόγραμμα make_on_queue.sh έχει τροποποιηθεί, ώστε να μετακινείται στον σωστό φάκελο της άσκησης (cd /home/parallel/parlab05/a3) και να εκτελεί εκεί την εντολή make. Με αυτόν τον τρόπο εξασφαλίζουμε ότι όλες οι εκδόσεις του K-means (σειριακή, naive, critical και με κλειδώματα) μεταγλωττίζονται απευθείας στο περιβάλλον του sandman, πριν από τη λήψη των μετρήσεων.

Τέλος, το πρόγραμμα run_on_queue.sh είναι υπεύθυνο για την αυτοματοποιημένη εκτέλεση των πειραμάτων στο μηχάνημα του sandman. Στην τελική του μορφή το script τρέχει όλα τα εκτελέσιμα κλειδωμάτων και για όλους τους αριθμούς νημάτων που ζητούνται (1, 2, 4, 8, 16, 32, 64). Για κάθε συνδυασμό κλειδώματος και αριθμό νημάτων, θέτει κατάλληλα τη μεταβλητή OMP_NUM_THREADS και ορίζει την πολιτική δέσμευσης πυρήνων μέσω της GOMP_CPU_AFFINITY, ώστε τα νήματα να κατανέμονται στους πρώτους φυσικούς πυρήνες του συστήματος (εκτέλεση πάντα «με affinity», όπως απαιτείται). Τα αποτελέσματα κάθε εκτέλεσης αποθηκεύονται σε ξεχωριστό κατάλογο της μορφής benchmarks/<lock_name>/S32_N16_C32_L10_T<threads>/, όπου δημιουργούνται αρχεία meta.txt με τα μεταδεδομένα της εκτέλεσης (εκτελέσιμο, τύπος κλειδώματος, παράμετροι K-means, αριθμός νημάτων, τιμές affinity) και output.txt με την πλήρη έξοδο του προγράμματος, συμπεριλαμβανομένης της γραμμής με τον χρόνο εκτέλεσης και τον αριθμό επαναλήψεων. Έτσι, οι επόμενες ενότητες μπορούν να επεξεργαστούν τα δεδομένα των benchmarks με τρόπο αντίστοιχο της προηγούμενης άσκησης και να παράγουν συγκρίσιμα διαγράμματα για όλες τις υλοποιήσεις κλειδωμάτων.

Για λόγους πληρότητας, το τροποποιημένο run_on_queue.sh (μόνο αυτό διαφοροποιήθηκε σημαντικά) παρουσιάζεται ακολούθως:

2. Λήψη Μετρήσεων και Διαγράμματα

Στη συνέχεια παρουσιάζουμε τους πίνακες με τα αποτελέσματα όλων των μετρήσεων που προέκυψαν από την εκτέλεση του K-means για τις παραμέτρους εισόδου SIZE=32, COORDS=16, CLUSTERS=32 και LOOPS=10 και για αριθμό νημάτων T ∈ {1, 2, 4, 8, 16, 32, 64}. Παραθέτουμε έναν πίνακα για κάθε μηχανισμό (συμπεριλαμβανομένου του critical). Με αυτόν τον τρόπο μπορούμε να συγκρίνουμε άμεσα το κόστος συγχρονισμού κάθε μηχανισμού και πώς αυτό κλιμακώνεται καθώς αυξάνεται ο βαθμός παραλληλισμού:

nosync_lock:

Threads	Total Time
1	1.3238
2	0.7005
4	0.4084
8	0.2984
16	0.7346
32	1.3143
64	0.9590

pthread_mutex_lock:

Threads	Total Time
1	1.3241
2	0.8372
4	0.7302
8	0.8456
16	2.6544
32	3.7170
64	3.8797

pthread_spin_lock:

Threads	Total Time
1	1.3174
2	0.7447
4	0.5589
8	0.8860
16	3.3755
32	7.7281
64	3.7769

tas_lock:

Threads	Total Time
1	1.3291
2	0.7396
4	0.5305
8	1.1465
16	4.5320
32	10.6944
64	9.3531

ttas_lock:

Threads	Total Time
1	1.3151
2	0.7513
4	0.5621
8	0.8613
16	3.2426
32	7.3978
64	4.3841

array_lock:

Threads	Total Time
1	1.3585
2	0.8212
4	0.5121
8	0.4942
16	1.5424
32	2.2050
64	2.1843

clh_lock:

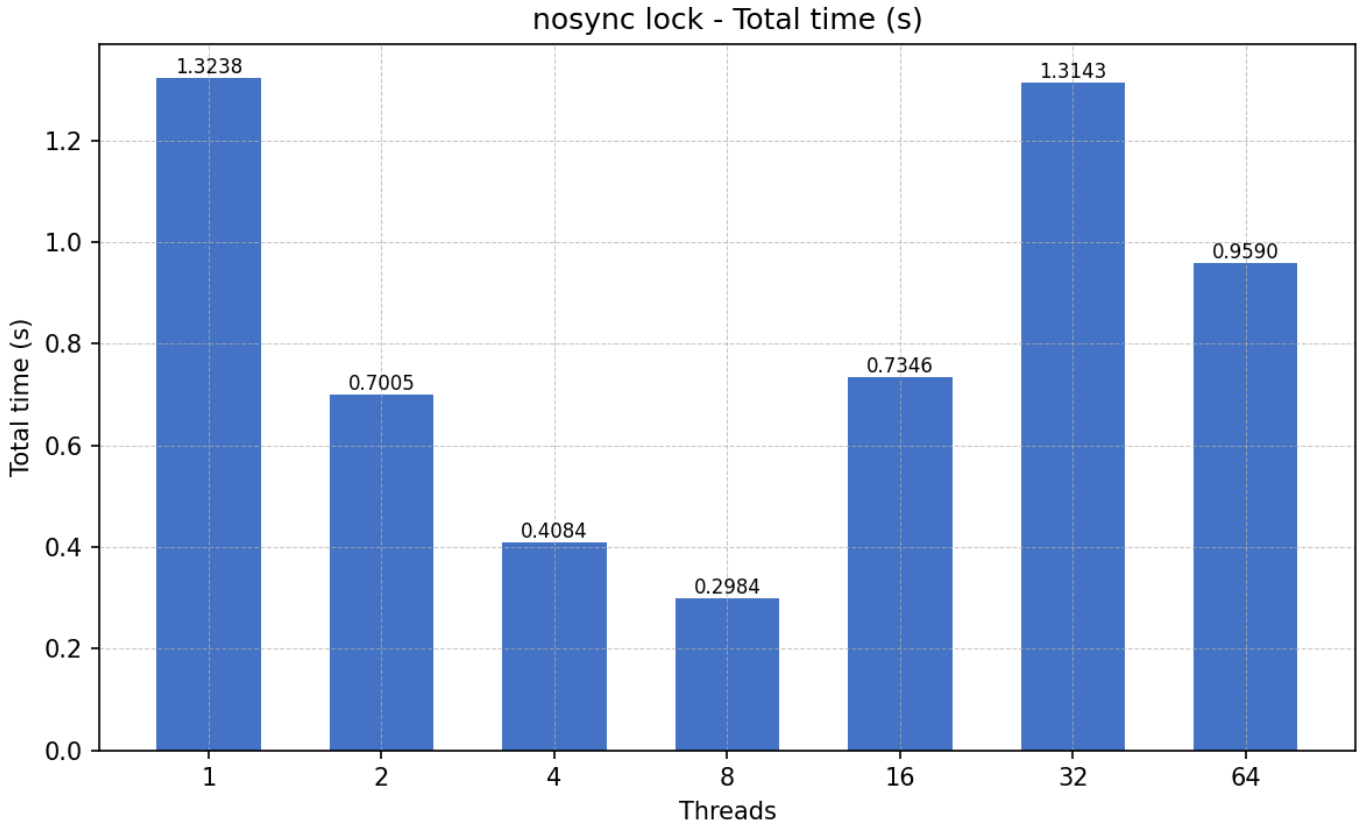
Threads	Total Time
1	1.3182
2	0.7995
4	0.4884
8	0.4319
16	1.2522
32	1.6519
64	1.4726

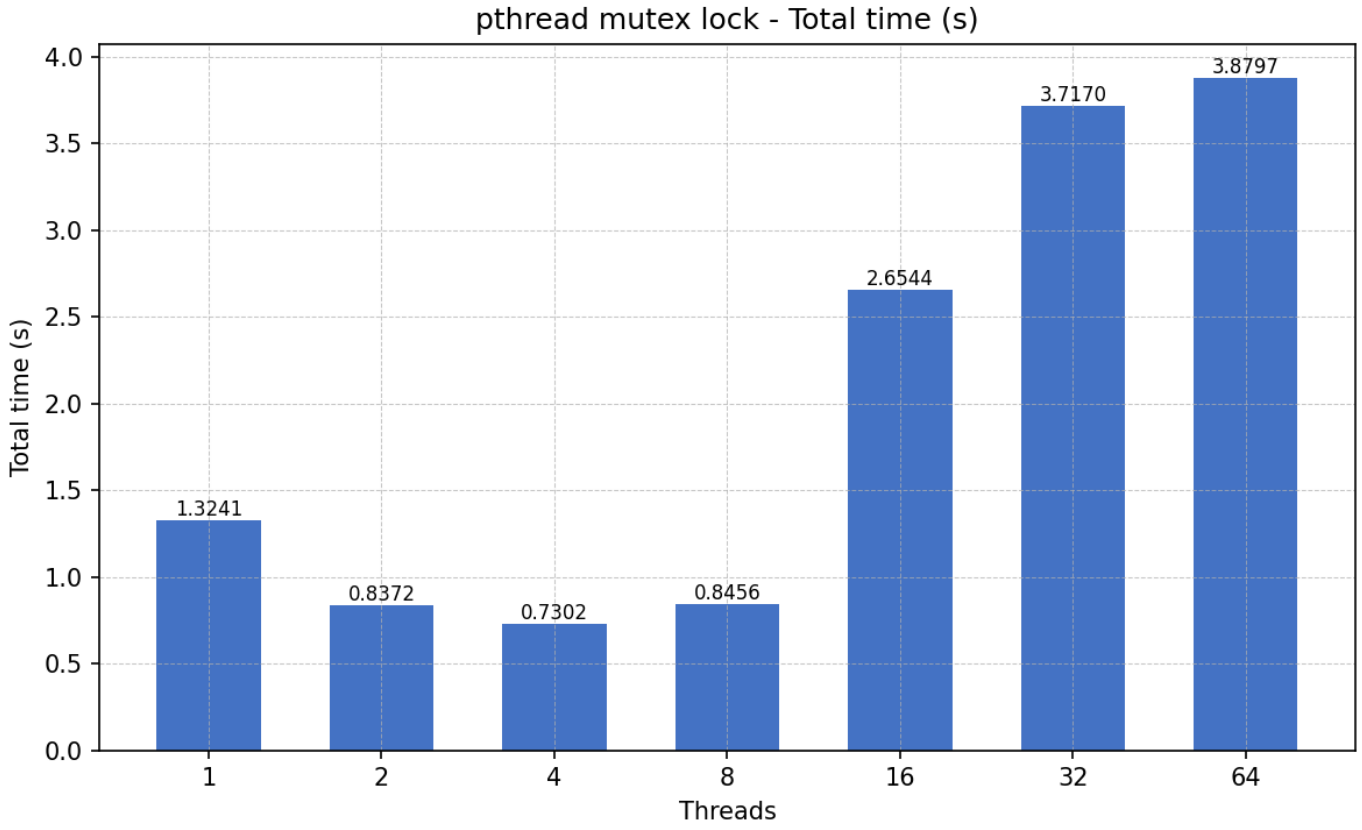
critical:

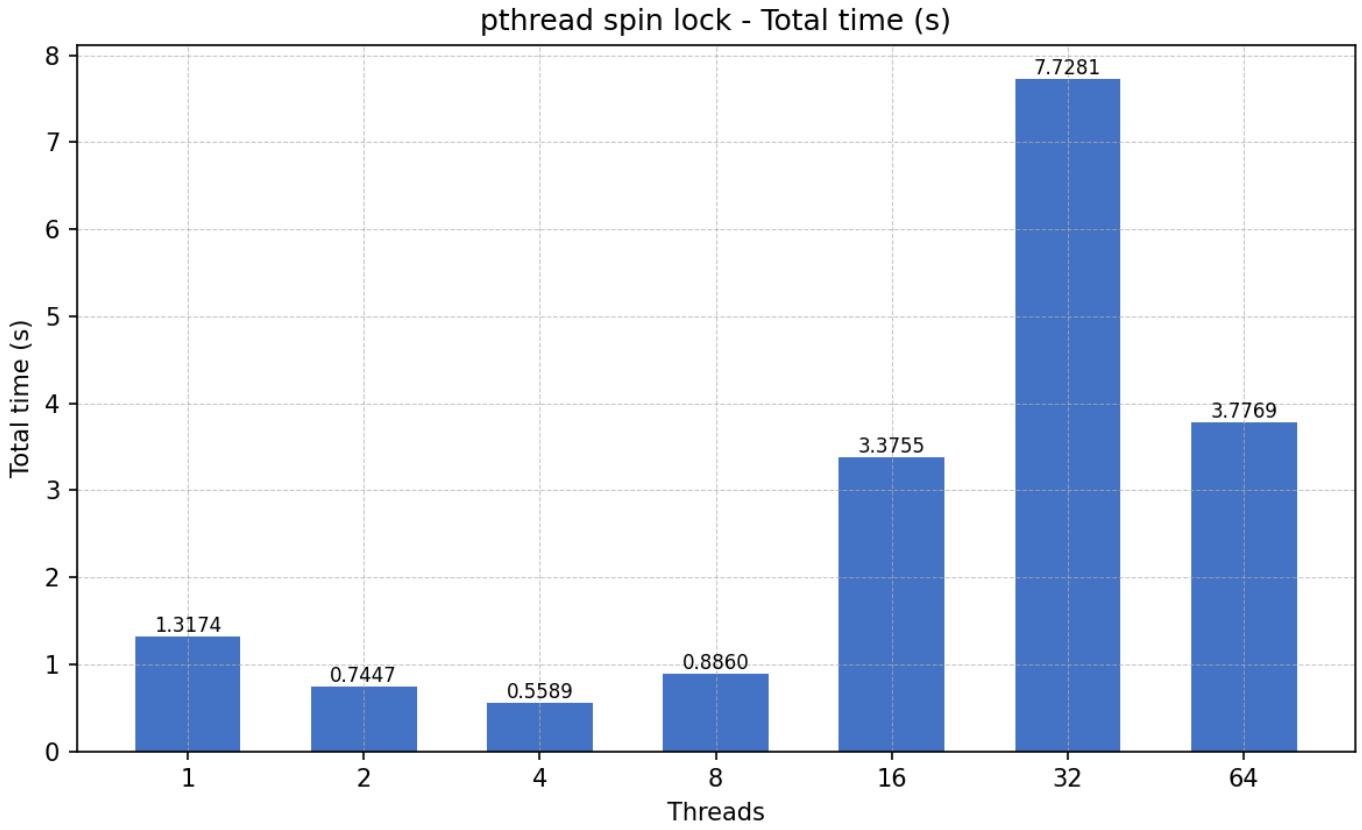
Threads	Total Time
1	1.3187
2	0.7827
4	0.4916
8	0.7655
16	3.4095
32	7.9911
64	6.1087

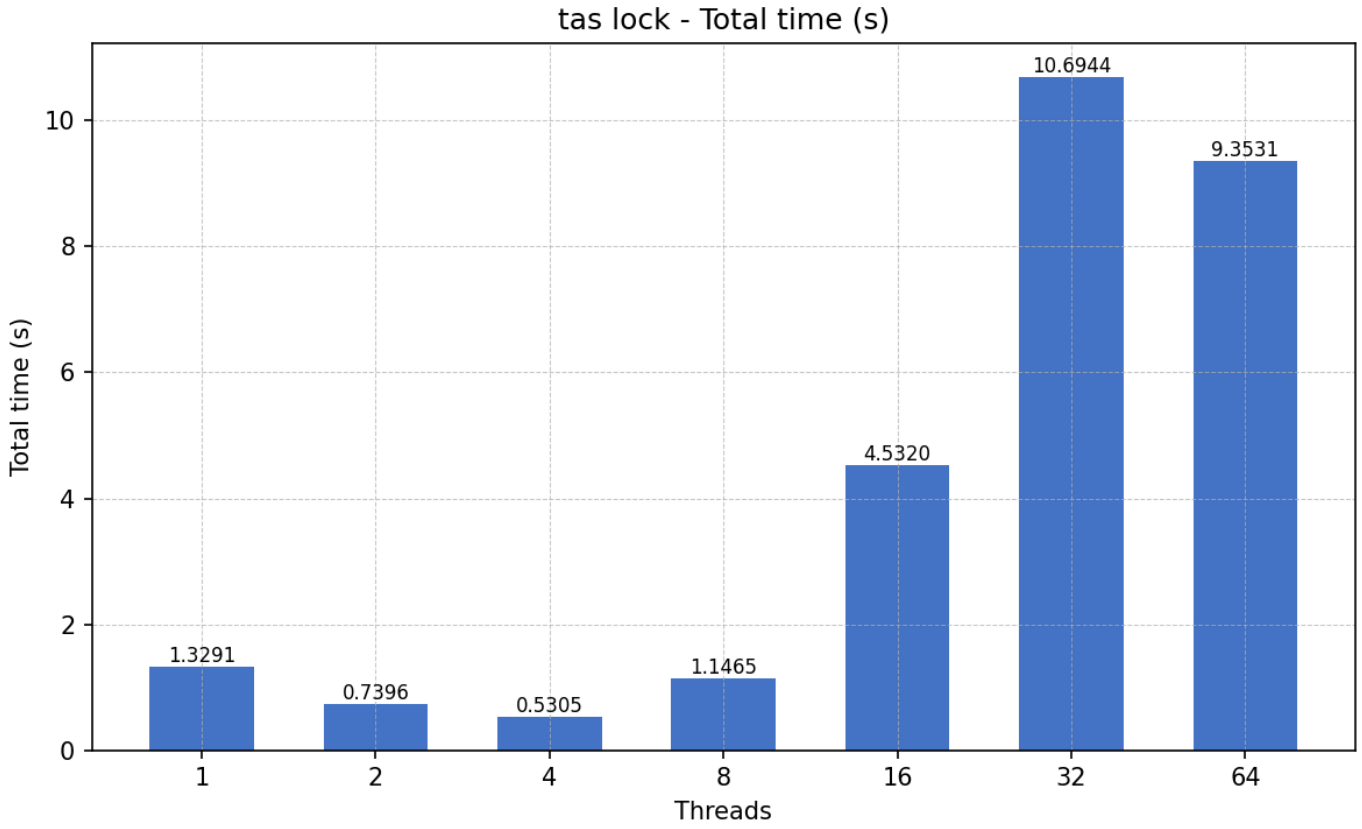
Για να είναι πιο ευδιάκριτες οι διαφορές μεταξύ των locks, παρουσιάζουμε τα αποτελέσματα και σε διαγράμματα. Στα διαγράμματα που ακολουθούν απεικονίζουμε τον χρόνο εκτέλεσης σε συνάρτηση με τον αριθμό νημάτων, για κάθε υλοποίηση κλειδώματος, καθώς και ένα συγκεντρωτικό διάγραμμα για όλους τους τύπους μαζί. Τα διαγράμματα αυτά μας επιτρέπουν να αξιολογήσουμε ποια

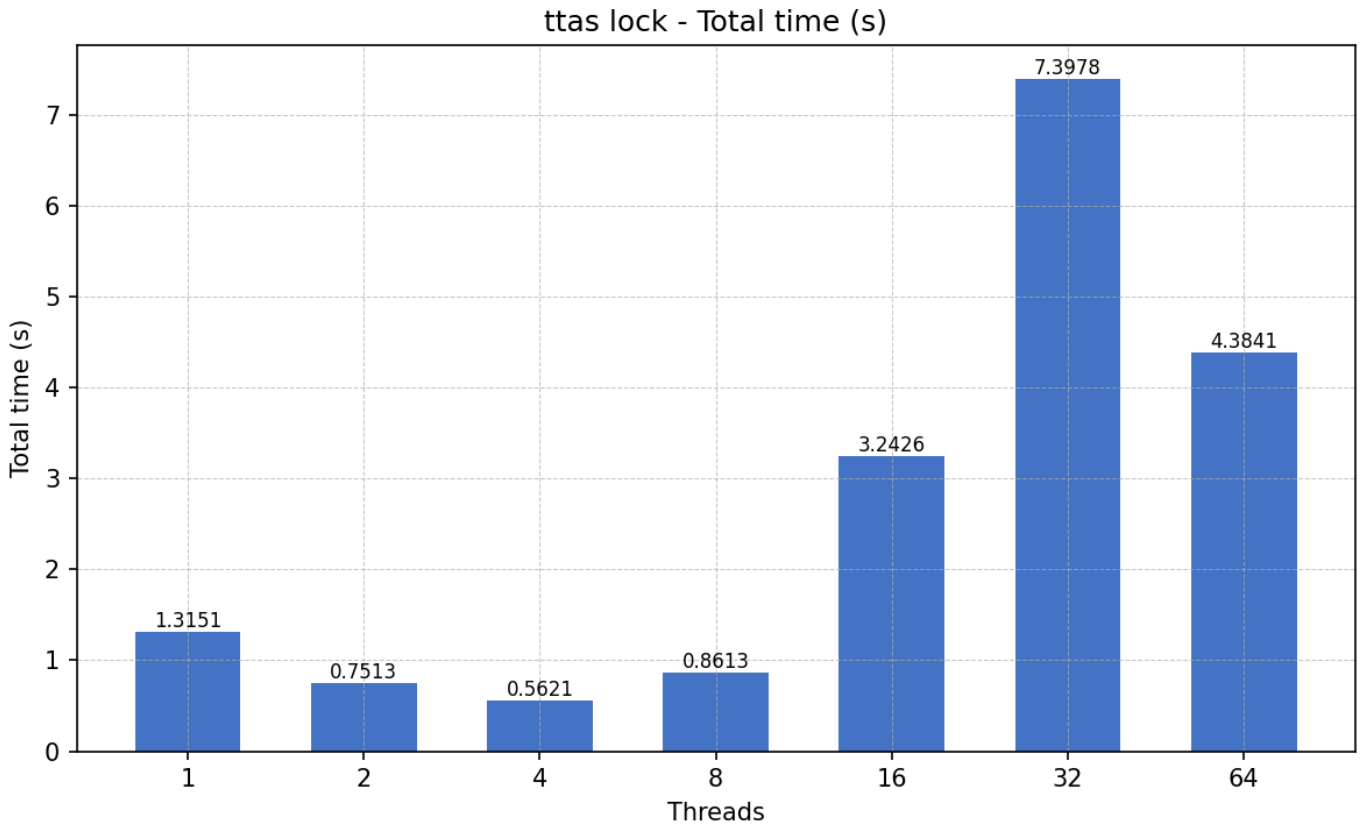
κλειδώματα παραμένουν αποδοτικά υπό αυξημένη συμφόρηση, ποια εμφανίζουν κορεσμό λόγω κόστους συγχρονισμού, καθώς και πώς συγκρίνεται η χρήση critical sections του OpenMP με τις υπόλοιπες υλοποιήσεις κλειδώματος:

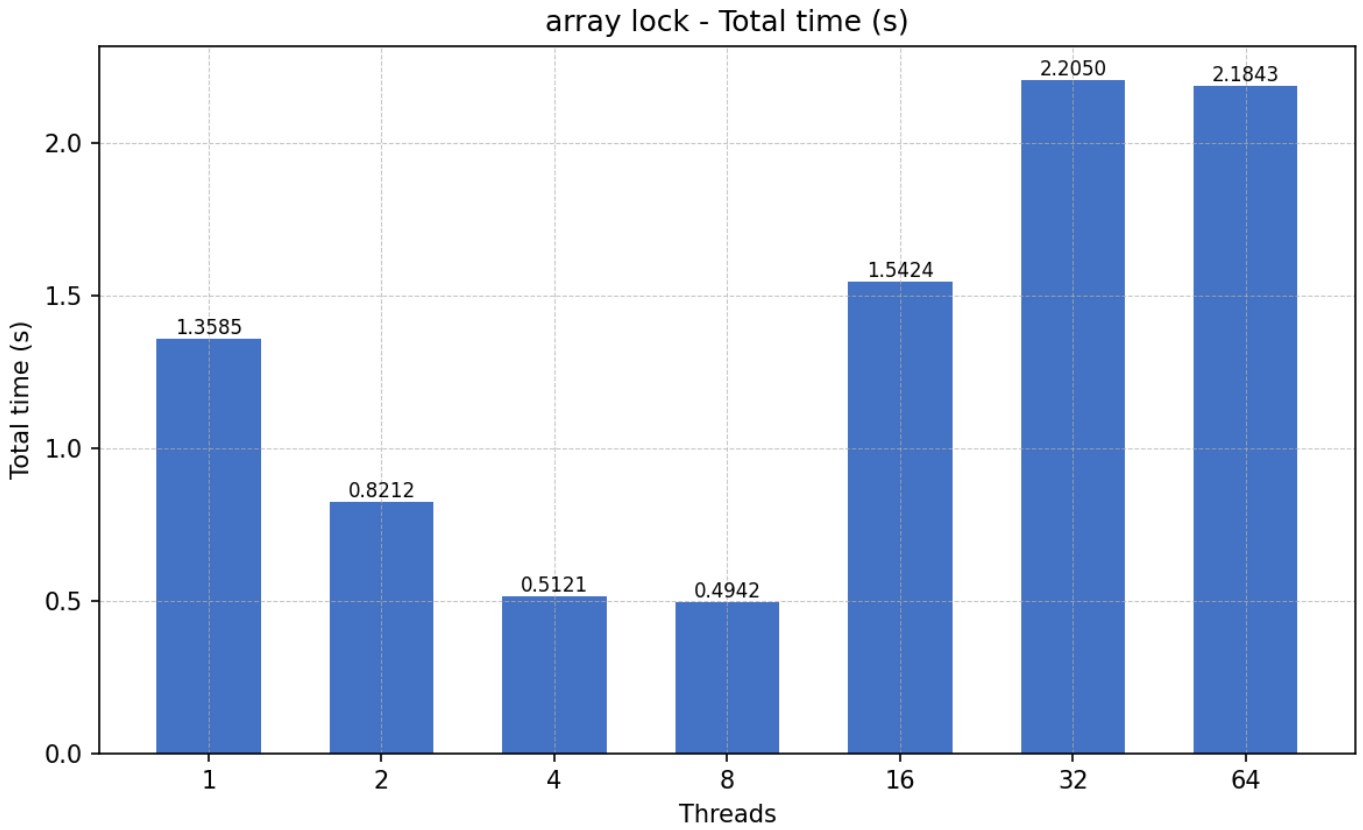


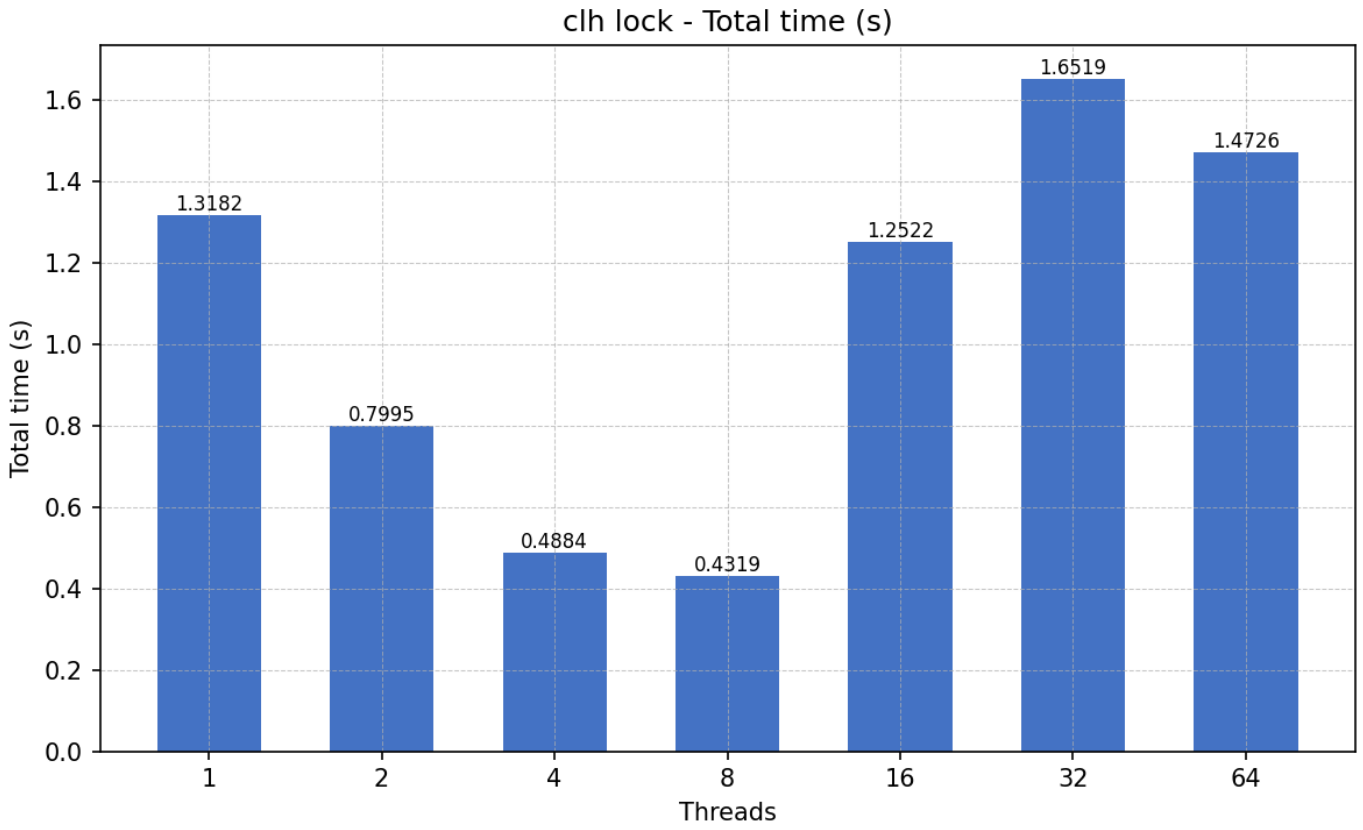


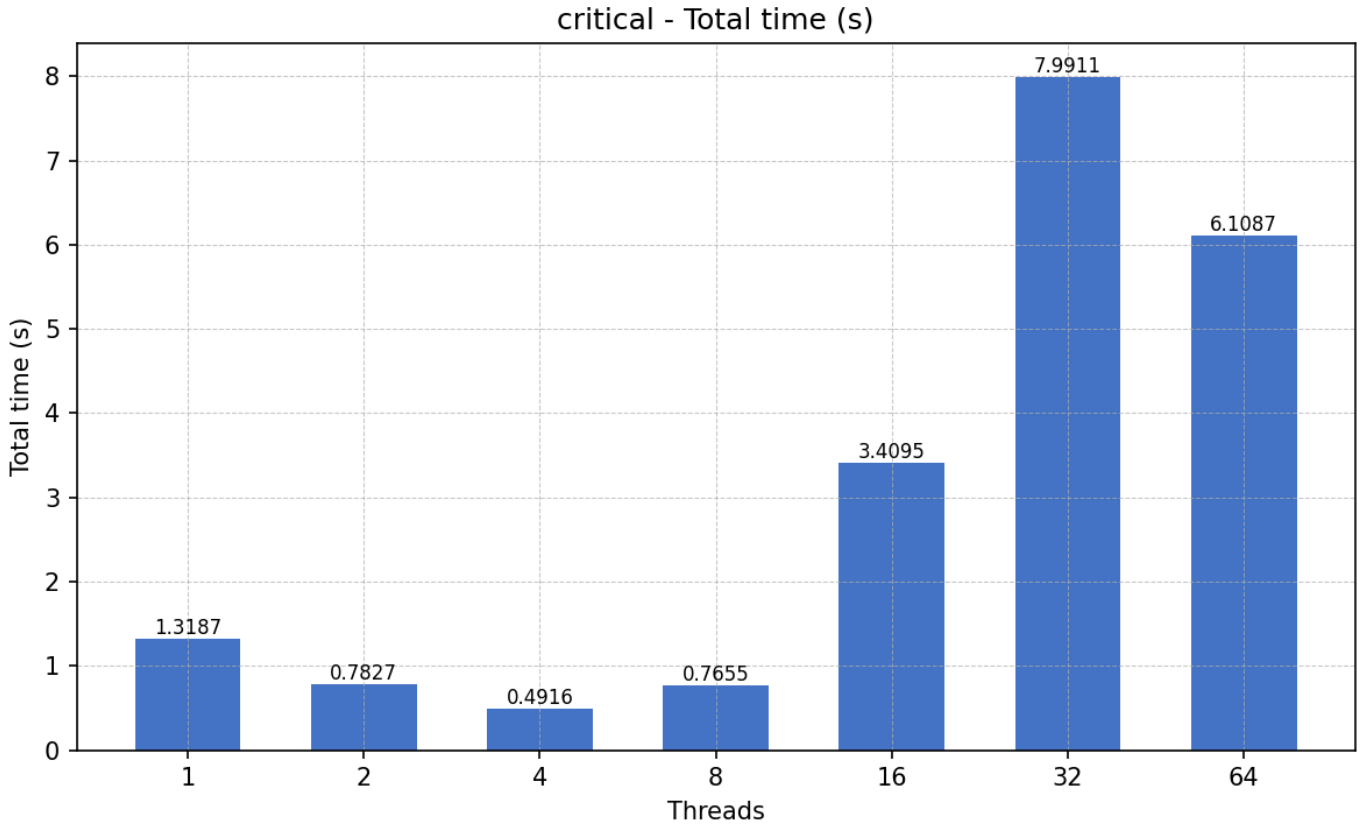

pthread spin lock - Total time (s)
Total time (s)
Threads
1.3174
0.7447
0.5589
0.8860
3.3755
7.7281
3.7769

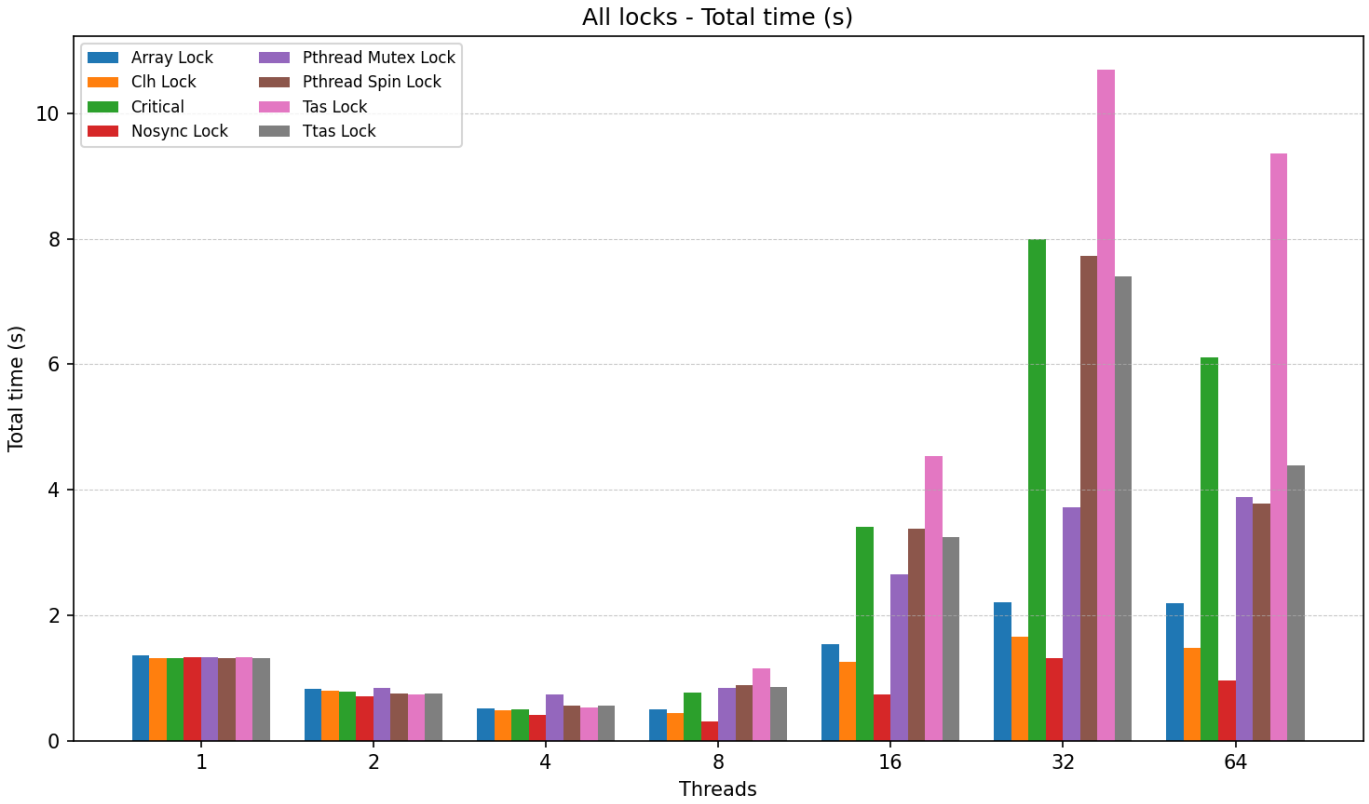

tas lock - Total time (s)
Total time (s)
Threads
1.3291
0.7396
0.5305
1.1465
4.5320
10.6944
9.3531


ttas lock - Total time (s)
Total time (s)
Threads
1.3151
0.7513
0.5621
0.8613
3.2426
7.3978
4.3841
0
1
2
3
4
5
6
7
1
2
4
8
16
32
64


array lock - Total time (s)
Total time (s)
Threads
1.3585
0.8212
0.5121
0.4942
1.5424
2.2050
2.1843
0.0
0.5
1.0
1.5
2.0
1
2
4
8
16
32
64


clh lock - Total time (s)
Total time (s)
Threads
1.3182
0.7995
0.4884
0.4319
1.2522
1.6519
1.4726


critical - Total time (s)
Total time (s)
Threads
1.3187
0.7827
0.4916
0.7655
3.4095
7.9911
6.1087



3. Σύγκριση Αποτελεσμάτων

Ξεκινώντας από τη σύγκριση μεταξύ των κλειδωμάτων, για T=1 όλοι οι μηχανισμοί (nosync, mutex, spin, TAS/TTAS, array, CLH, omp_critical) έχουν πολύ κοντινούς χρόνους, αφού δεν υπάρχει ουσιαστικό contention και το κόστος του lock είναι αμελητέο. Καθώς αυξάνουμε τα νήματα, ο «ιδανικός» δείκτης nosync_lock και τα queue-based locks (array_lock, clh_lock) βελτιώνουν τον χρόνο μέχρι περίπου τα 8 νήματα και από εκεί και πέρα χειροτερεύουν ήπια. Αντίθετα, τα pthread_mutex_lock, pthread_spin_lock, TAS/TTAS και το omp_critical_kmeans αρχίζουν να πέφτουν γρήγορα: για 8–16 νήματα ο χρόνος σταθεροποιείται ή αυξάνεται και για περισσότερα αυξάνεται ραγδαία, καθώς το κρίσιμο τμήμα σειριοποιείται και το lock γίνεται το βασικό bottleneck.

Η συμπεριφορά αυτή συνδέεται και με τη φύση του ίδιου του προβλήματος: στο configuration {32,16,32,10} ο K-means είναι έντονα memory-bound, με περιορισμένο υπολογιστικό φορτίο ανά στοιχείο (compute intensity). Όταν προσθέτουμε ένα «βαρύ» lock πάνω σε ένα μικρό, κυρίως memory-bound workload, το κόστος συγχρονισμού κυριαρχεί, ειδικά όταν πολλά νήματα προσπαθούν να μπουν στο ίδιο critical section. Τα queue locks περιορίζουν καλύτερα το contention

(λιγότερη τυχαία κίνηση σε κοινές cache lines) και γι’ αυτό αντέχουν περισσότερο σε υψηλό παραλληλισμό από ό,τι τα απλά spinlocks ή το critical section.

Σε σχέση με την προηγούμενη άσκηση, τα ευρήματα είναι απολύτως συνεπή. Η shared-clusters υλοποίηση με #pragma omp atomic είχε λίγο κέρδος μέχρι περίπου τα 8 νήματα και μετά υπόκειται σε κορεσμό, ακριβώς επειδή πολλές atomic ενημερώσεις σε λίγες κοινόχρηστες δομές σειριοποιούσαν μεγάλο μέρος της δουλειάς. Στην τωρινή άσκηση, το omp_critical_kmeans και τα απλά spin/TAS locks εμφανίζουν την ίδια συμπεριφορά: λειτουργικά σωστά, αλλά με περιορισμένη κλιμάκωση λόγω έντονου synchronization και memory contention.

Αντίθετα, η λύση της προηγούμενης άσκησης με copied clusters και explicit reduction πέτυχε πολύ καλύτερη κλιμάκωση γιατί μείωσε δραστικά το fine-grained synchronization: κάθε νήμα δούλευε σε ιδιωτικές δομές και συγχρονιζόταν μόνο στη φάση του reduction. Τα queue locks της τωρινής άσκησης κινούνται προς αυτή την κατεύθυνση (περιορίζουν το contention και βελτιώνουν την shared υλοποίηση), αλλά δεν μπορούν να φτάσουν το επίπεδο της αλγοριθμικής βελτίωσης του copied clusters + reduction, ιδίως σε ένα μικρό και τόσο memory-bound πρόβλημα.

Όπως τονίζεται και στις διαφάνειες του μαθήματος, η επίδοση κάθε μηχανισμού κλειδώματος εξαρτάται κρίσιμα από τον αριθμό νημάτων, το μέγεθος του κρίσιμου/μη κρίσιμου τμήματος και το επίπεδο συμφόρησης. Δεν υπάρχει ένα «καθολικά καλύτερο» κλείδωμα, αλλά διαφορετικοί μηχανισμοί που αποδίδουν καλύτερα σε διαφορετικά σενάρια.

4. Τα Κλειδώματα

Κάθε κλείδωμα υλοποιεί τον αμοιβαίο αποκλεισμό με διαφορετικό τρόπο και αυτό έχει άμεσο αντίκτυπο στην επίδοση, ειδικά σε ένα έντονα memory-bound πρόβλημα, όπως το K-means της άσκησης.

Το nosync_lock δεν είναι πραγματικό κλείδωμα. Ουσιαστικά δεν κάνει καμία προστασία του κρίσιμου τμήματος και χρησιμοποιείται μόνο ως θεωρητικό baseline. Γι’ αυτό εμφανίζει τους καλύτερους χρόνους: δεν υπάρχει overhead συγχρονισμού, άρα όλα τα νήματα γράφουν χωρίς συντονισμό πάνω στις κοινόχρηστες δομές. Στην πράξη όμως η υλοποίηση αυτή δεν είναι ορθά συγχρονισμένη και μπορεί να οδηγήσει σε λανθασμένα αποτελέσματα. Γι’ αυτό χρησιμοποιείται μόνο για να απομονώσουμε το «καθαρό» κόστος του locking.

Το pthread_mutex_lock υλοποιεί ένα blocking lock σε επίπεδο POSIX. Όταν υπάρχει μικρό contention, η απόδοσή του είναι σχετικά καλή, αλλά μόλις πολλαπλά νήματα αρχίσουν να μπλοκάρουν πάνω στο ίδιο mutex, ο μηχανισμός αναγκάζεται να κάνει system calls και πιθανές μεταβάσεις σε kernel space, με context switches κ.λπ. Σε ένα σενάριο με πολλές, μικρής διάρκειας κρίσιμες περιοχές (όπως οι ενημερώσεις στους μετρητές του K-means) αυτό το overhead γίνεται δυσανάλογα μεγάλο και εξηγεί γιατί το mutex δεν κλιμακώνεται καλά για T≥8. Το pthread_spin_lock, από την άλλη, είναι spin lock: τα νήματα δεν αποκοιμούνται αλλά κάνουν busy-wait σε μια μεταβλητή. Αυτό αποφεύγει τις ακριβές μεταβάσεις στο kernel όταν το lock κρατιέται για πολύ λίγο χρόνο, αλλά υπό έντονο contention το busy-wait καταναλώνει CPU cycles και δημιουργεί μεγάλη πίεση στο memory system, αφού όλα τα νήματα διαβάζουν/γράφουν την ίδια cache line – γι’ αυτό βλέπουμε χειροτέρευση χρόνου για πολλά νήματα.

Τα tas_lock και ttas_lock είναι απλές υλοποιήσεις spin lock σε επίπεδο user-space. Το TAS (test-and-set) κάνει συνεχές atomic γράψιμο σε μια κοινή μεταβλητή μέχρι να πάρει το lock, με αποτέλεσμα να προκαλεί μεγάλο traffic στο bus και να «πετάει» την cache line από πυρήνα σε πυρήνα. Το TTAS (test-and-test-and-set) βελτιώνει την κατάσταση: τα νήματα πρώτα κάνουν απλό read (χωρίς write) και μόνο όταν φαίνεται ότι το lock είναι ελεύθερο επιχειρούν atomic test-and-set. Αυτό μειώνει κάπως τον αριθμό των writes, αλλά όλα τα νήματα εξακολουθούν να γυρίζουν γύρω από την ίδια cache line. Συνεπώς, σε χαμηλό contention τα TAS/TTAS είναι σχετικά ελαφριά, ενώ σε υψηλό contention έχουν πολύ κακή κλιμάκωση λόγω τρικυμίας στο πρωτόκολλο συνοχής της μνήμης.

Τα array_lock και clh_lock είναι queue-based locks και εδώ φαίνεται η διαφορά τους στην επίδοση. Στο array lock κάθε νήμα παίρνει μια θέση σε έναν «κυκλικό» πίνακα και περιμένει σε δικό του κελί, έτσι ώστε μόνο ένας μικρός αριθμός cache lines να αλλάζει χέρια κάθε φορά. Στο CLH lock κάθε νήμα δημιουργεί έναν κόμβο σε μια λογική ουρά και σπινάρει πάνω σε μια μεταβλητή που ανήκει στον «προκάτοχό» του. Και στις δύο περιπτώσεις, το spinning γίνεται πάνω σε τοπικά ή σχεδόν τοπικά δεδομένα, με λιγότερη ανταλλαγή cache lines ανά απόκτηση lock και με εγγενή δικαιοσύνη (FIFO σειρά). Αυτό εξηγεί γιατί τα queue locks είχαν πολύ καλύτερη συμπεριφορά για T=8–32: μειώνουν δραστικά το lock contention που βλέπαμε στα TAS/TTAS και spin locks, και δεν «πνίγουν» τον αλγόριθμο με άσκοπο coherence traffic.

Τέλος, το omp_critical_kmeans χρησιμοποιεί την εντολή #pragma omp critical, η οποία αντιστοιχεί σε ένα implicit global lock γύρω από το κρίσιμο τμήμα. Η

λειτουργία του είναι εννοιολογικά παρόμοια με έναν mutex: μόνο ένα νήμα κάθε φορά περνάει μέσα στο critical section, όλα τα υπόλοιπα περιμένουν. Ο απλός αυτός μηχανισμός είναι εύχρηστος προγραμματιστικά, αλλά, όπως και το pthread_mutex_lock, γίνεται γρήγορα bottleneck όταν πολλά νήματα προσπαθούν να ενημερώσουν την ίδια κοινόχρηστη δομή. Έτσι, οι χρόνοι του omp_critical_kmeans αντανακλούν την ίδια εικόνα με το naive shared-clusters της προηγούμενης άσκησης: σωστή αλλά βαριά μορφή συγχρονισμού, που περιορίζει την παραλληλία και δεν κλιμακώνεται καλά σε υψηλό contention, ειδικά σε ένα μικρό και έντονα memory-bound πρόβλημα.

Τα κλειδώματα τύπου TAS/TTAS και spin lock ανήκουν στην κατηγορία των απλών spinlocks, ενώ τα array και CLH συγκαταλέγονται στα scalable queue locks, τα οποία – όπως φαίνεται και από τις μετρήσεις μας – μειώνουν το coherence traffic και κλιμακώνονται καλύτερα υπό έντονο contention.

Σ.Η.Μ.Μ.Υ. Ε.Μ.Π.
Νοέμβριος 2025