3η ΆΣΚΗΣΗ ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Ενρίκα Ηλιάνα Ματζόρι ΑΜ:03120143

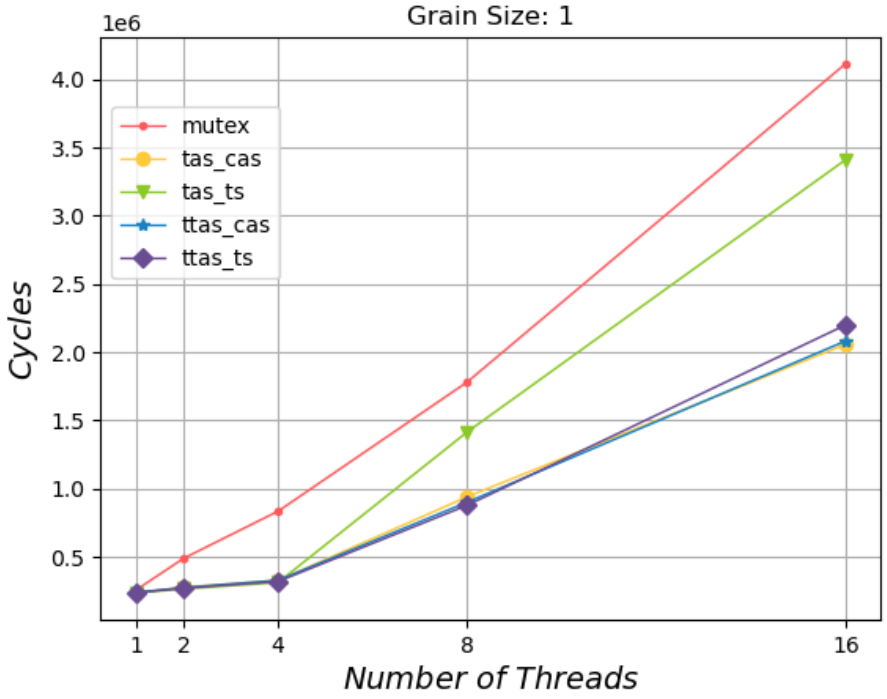
Κώδικας:
Συμπληρώνοντας τις ζητούμενες συναρτήσεις στο αρχείο lock.h υλοποιούμε τις μεθόδους συγχρονισμού που θα χρησιμοποιηθούν στα ακόλουθα πειράματα. Ο συμπληρωμένος κώδικας παρατίθεται παρακάτω:

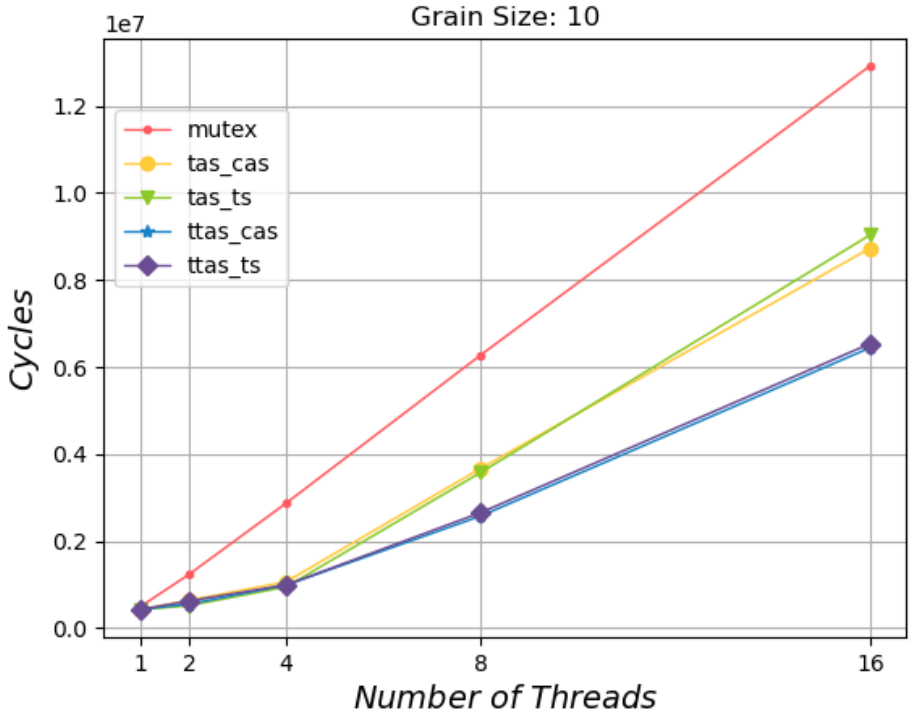
```c
#ifndef LOCK_H_
#define LOCK_H_

typedef volatile int spinlock_t;

#define UNLOCKED  0
#define LOCKED    1

//initialization
static inline void spin_lock_init(spinlock_t *spin_var)
{
    *spin_var = UNLOCKED;
}


static inline void spin_lock_tas_cas(spinlock_t *spin_var)
{
    while (__sync_val_compare_and_swap(spin_var, UNLOCKED, LOCKED) != UNLOCKED);
}

static inline void spin_lock_ttas_cas(spinlock_t *spin_var)
{
    //We implement busy-waiting loop until the lock is acquired
    while (1) {
        if (*spin_var == UNLOCKED) {
            if (__sync_val_compare_and_swap(spin_var, UNLOCKED, LOCKED) == UNLOCKED) {
                break;
            }
        }

    }
}

static inline void spin_lock_tas_ts(spinlock_t *spin_var)
{
    while(__sync_lock_test_and_set(spin_var, LOCKED) == LOCKED);
}

static inline void spin_lock_ttas_ts(spinlock_t *spin_var)
{
    //We implement busy-waiting loop until the lock is acquired
    while (1) {
        if (*spin_var == UNLOCKED) {
            if (__sync_lock_test_and_set(spin_var, LOCKED) == UNLOCKED) {
                break;
            }
        }
    }
}
//unlock the spinlock
static inline void spin_unlock(spinlock_t *spin_var)
{
    __sync_lock_release(spin_var);
}

#endif
```

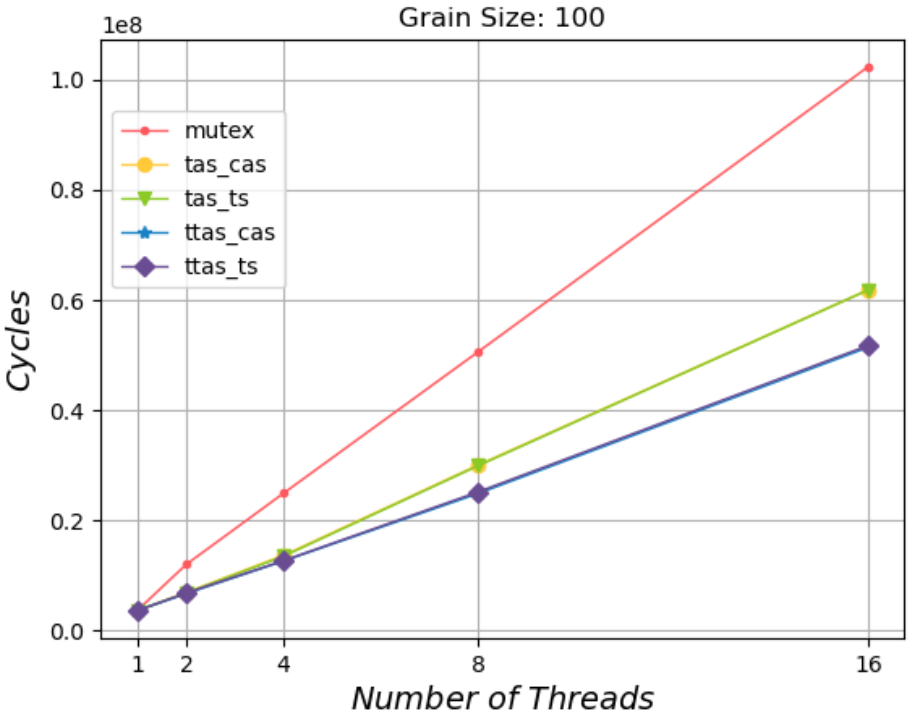
4.1 Σύγκριση υλοποιήσεων

4.1.1.

Παραθέτουμε τα τρία ζητούμενα διαγράμματα που αφορούν την κλιμάκωση του συνολικού χρόνου εκτέλεσης της περιοχής ενδιαφέροντος για κάθε υλοποίηση σε σχέση με το πλήθος των νημάτων και για κάθεμία τιμή του grain size:







4.1.2.

Το πρώτο συμπέρασμα που προκύπτει εξαρχής από τα διαγράμματα είναι ότι, σε κάθε περίπτωση, για όλες τις τιμές grain size και για κάθε αριθμό threads, η χρήση των mutexes ως μηχανισμού κλειδώματος οδηγεί συνολικά σε περισσότερους κύκλους. Αυτό οφείλεται λογικά στο ότι τα mutexes απαιτούν το «κοίμισμα» των νημάτων όταν δεν υπάρχει η δυνατότητα να πάρουν το lock μέχρι αυτό να απελευθερωθεί, οπότε και να «ξυπνήσουν». Η διαδικασία αυτή απαιτεί context switching, πράγμα που όταν γίνεται πολύ συχνά, είναι κοστοβόρο για την επίδοση του συστήματος. Απεναντίας, στην περίπτωση της χρήσης των spinlocks (TAS, TTAS), δεν έχουμε context switching και γενικά κοίμισμα διεργασιών, αλλά αυτά δουλεύουν με τη μέθοδο του busy-waiting loop.

Σε ό,τι αφορά τον μηχανισμό TAS (Test-and-Set), οι επεξεργαστές, ατομικά, προσπαθούν να αποκτήσουν το κλείδωμα, κάνοντας έτσι εγγραφές στην μνήμη. Λόγω του ότι αυτό επιτυγχάνεται από τον κάθε επεξεργαστή ατομικά, η πλειονότητα των εγγραφών αυτών θα οδηγήσουν σε αστοχίες εγγραφής. Έτσι, αυτό έχει ως συνέπεια ο ένας επεξεργαστής να ακυρώνει συνέχεια το cache line του άλλου, παράγοντας έτσι μεγάλη κίνηση στον δίαυλο(υψηλό memory bus traffic).

Απεναντίας, η μέθοδος TTAS (Test-and-Test-and-Set) έχει μειωμένο memory bus traffic συγκριτικά με την TAS αφού, πρωτού δοκιμάσει να πάρει, δηλαδή να κλειδώσει, το lock ανταγωνιζόμενος όλους τους άλλους επεξεργαστές δρώντας πάντα ατομικά, ελέγχει συνεχώς την τιμή του, και εκτελεί το atomic operation του όταν δει ότι το κλείδωμα είναι διαθέσιμο. Έτσι τα atomic operations που πραγματοποιούνται είναι λιγότερα και η κίνηση στον δίαυλο είναι μειωμένη.

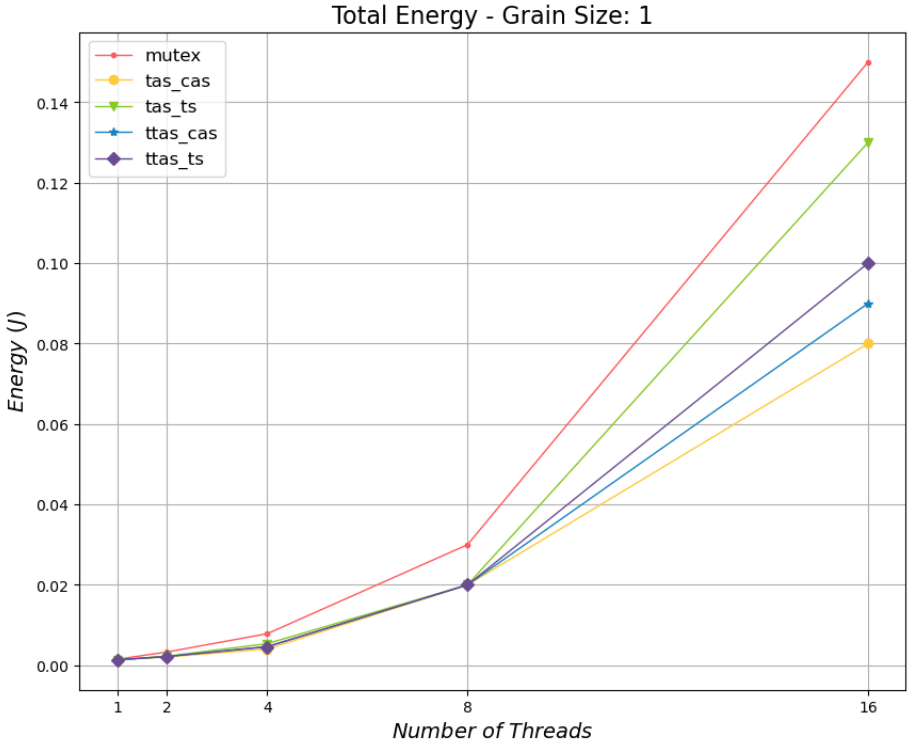
Τα παραπάνω αποδεικνύει και όσα φαινονται στα παραπάνω διαγράμματα, δηλαδή ότι η χρήση των μηχανισμών κλειδώματος TTAS δύνανται να οδηγήσει σε καλύτερη κλιμάκωση σε σύγκριση με τη μέθοδο TAS. Επίσης παρατηρούμε ότι η χρήση είτε του atomic operation test-and-set είτε του compare-and-swap δεν επέφερε κάποια ουσιώδη διαφορά στην επίδοση του συστήματος.

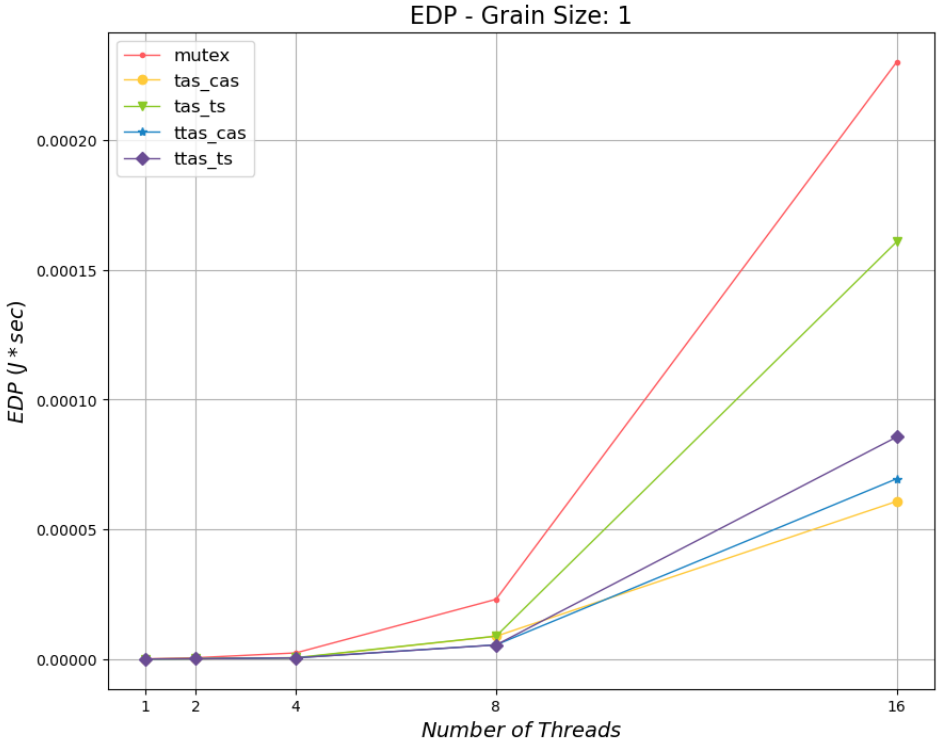
Παρατηρούμε ότι όσο αυξάνεται το grain size, δηλαδή το «μέγεθος» του φόρτου εργασίας που ανατίθεται σε κάθε thread πρέπει να ολοκληρωθεί πρωτού απαιτηθεί να επικοινωνήσει ή γενικά να συγχρονιστεί με άλλα, ο συνολικός αρθμός κύκλων αυξάνεται. Αυτό οφείλεται πιθανά στο ότι, για μεγάλο grain size, δεν έχει γίνει ομαλή κατανομή του φόρτου εργασίας σε κάθε thread, οπότε μερικά τελειώνουν όσα τους έχουν ανατεθεί πριν από τα άλλα, με αποτέλεσμα να μην κάνουν κάτι και να τα περιμένουν. Επισημαίνεται ότι, γενικά, με την αύξηση του grain size, μειώνεται το

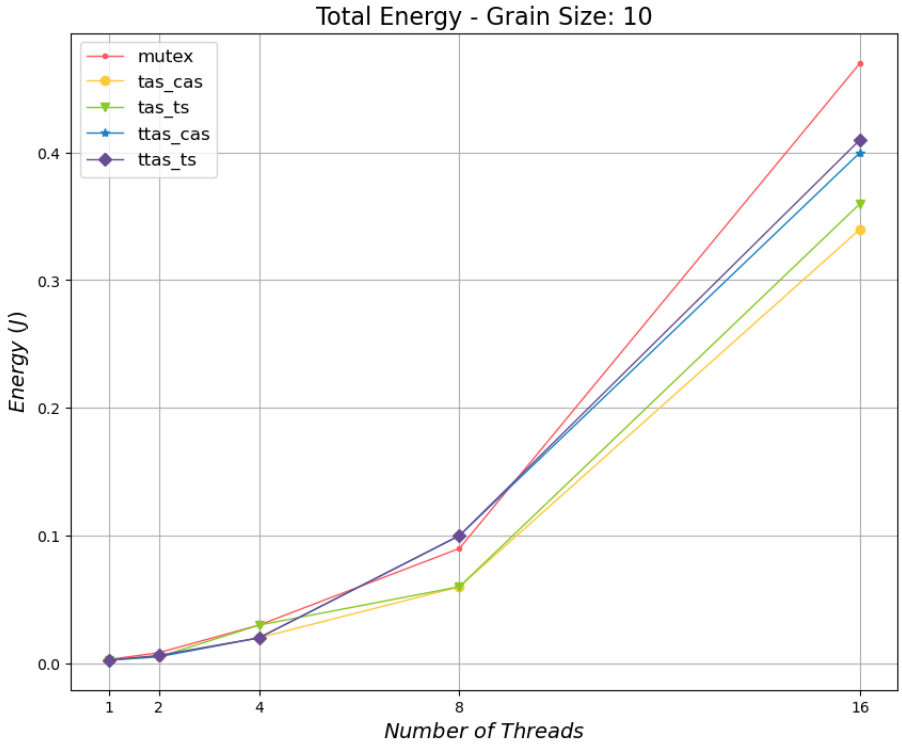
synchronization overhead, αφού τα threads απαιτείται να συγχρονίζονται πιο σπάνια. Ωστόσο, αυτό λόγω του μεγάλου χρόνου στον οποίο τα threads που τελείωσαν νωρίτερα παραμένουν «άεργα», δηλαδή του idle time, τελικά δεν επιτυγχάνεται κάτι καλύτερο στην τελική απόδοση αλλά αντίθετα, αυτή γίνεται δυσχεραίνει.

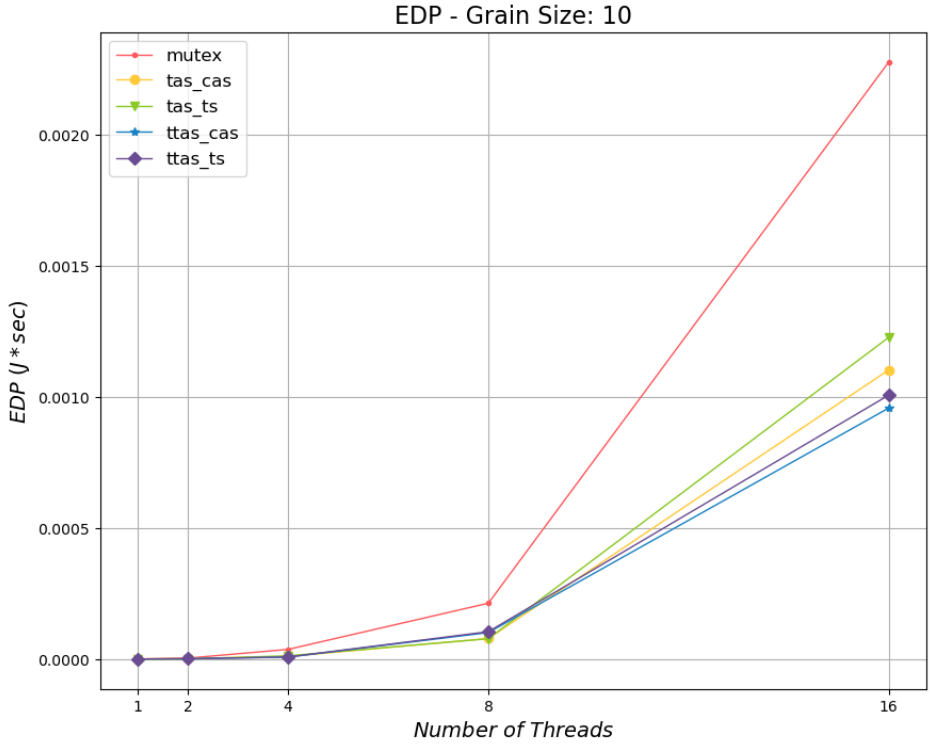
4.1.3.

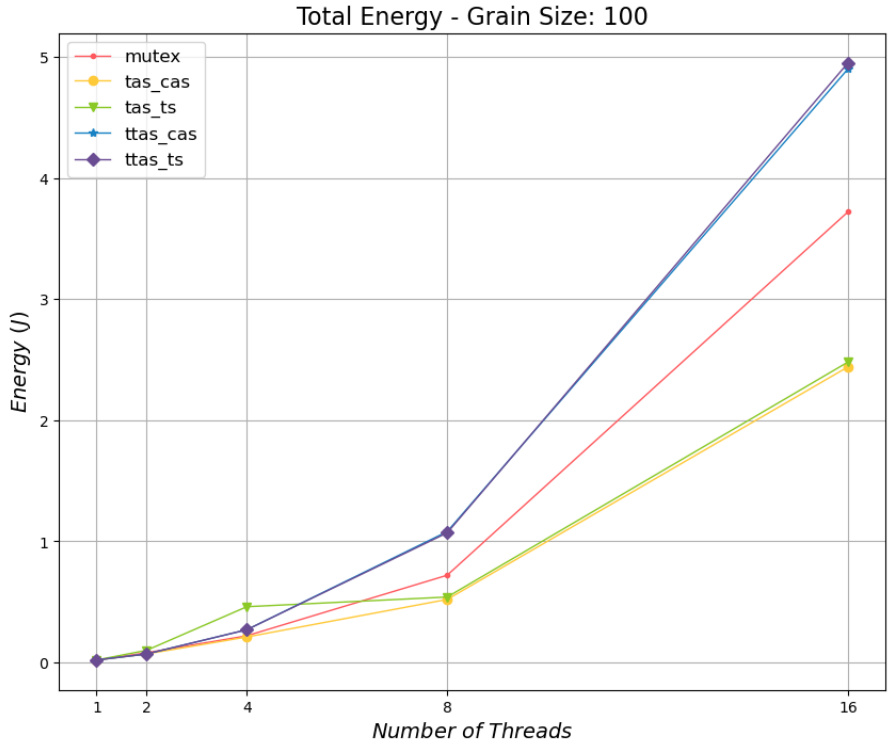
Παρατίθενται τα ζητούμενα διαγράμματα όπου πλέον αντί να μετράμε τον συνολικό αριθμό κύκλων, αναπαριστούμε για κάθε περίπτωση την συνολική κατανάλωση Ενέργειας σε Joule καθώς και την μεταβολή του EDP:

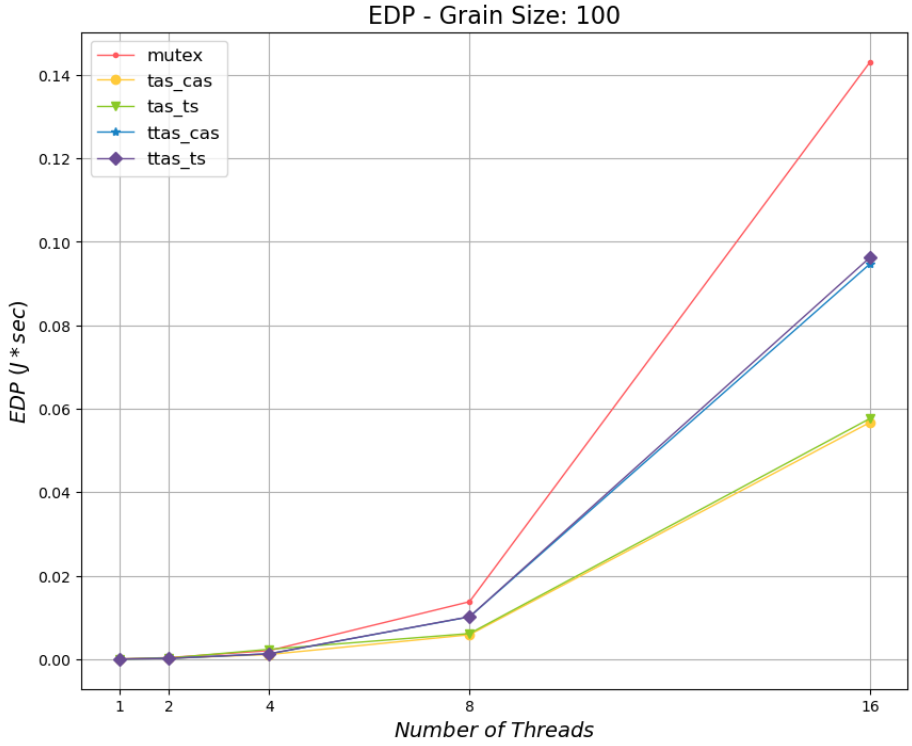





Total Energy - Grain Size: 10
mutex
tas_cas
tas_ts
ttas_cas
ttas_ts
Energy (J)
0.0
0.1
0.2
0.3
0.4
1
2
4
8
16
Number of Threads


EDP - Grain Size: 10
mutex
tas_cas
tas_ts
ttas_cas
ttas_ts
EDP (J*sec)
0.0000
0.0005
0.0010
0.0015
0.0020
1
2
4
8
16
Number of Threads



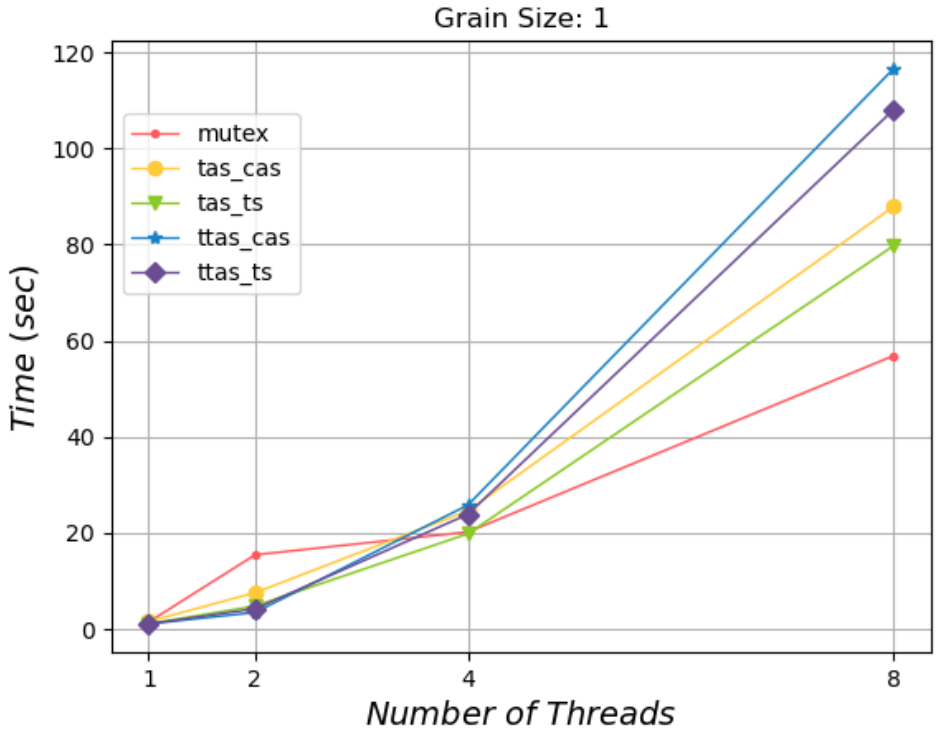


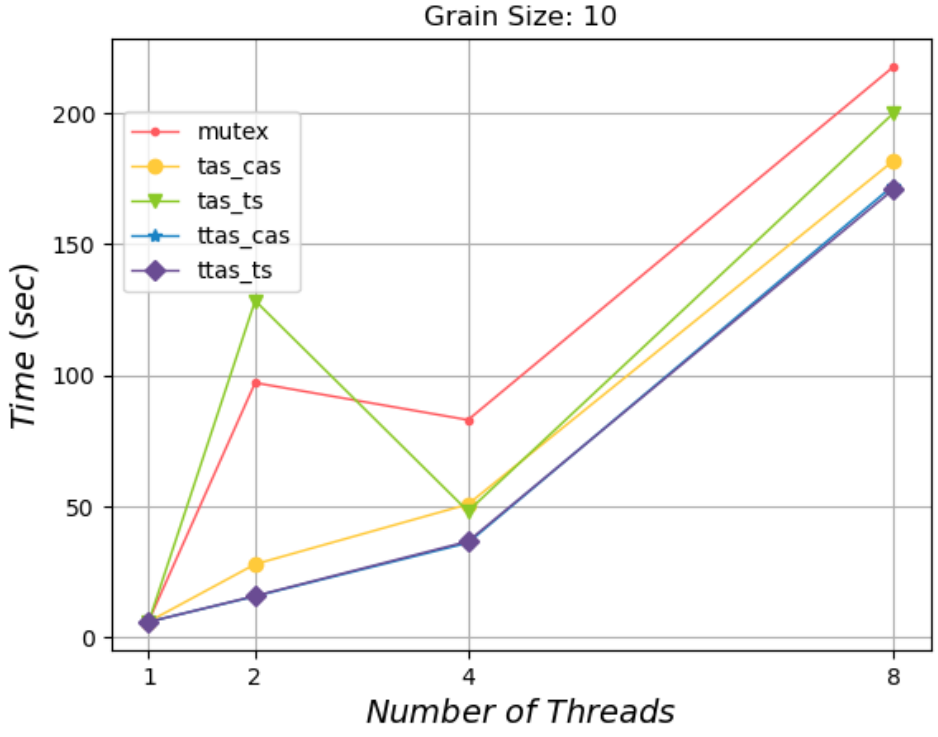
Ξεκινάμε παρατηρώντας τι συμβαίνει για μικρές τιμές του grain size.
Βλέπουμε ότι για grain size=1,10 , τα mutexes είναι ενεργειακά πιο κοστοβόρα από τους άλλους μηχανισμούς που χρησιμοποιούν spinlocks. Αυτό οφείλεται, όπως αναφέρθηκε προηγουμένως, στο context switching που απαιτείται στα mutexes και είναι πιο συχνό για μικρά grain sizes, καθώς υπάρχει μεγαλύτερη ανάγκη για συγχρονισμό. Σε ό,τι αφορά τα spinlocks, για grain size=1 παρατηρούμε ότι το πιο κοστοβόρο είναι το TAS ενώ για grain size=10, η ενεργειακή κατανάλωση κυμαίνεται σε παρόμοια επίπεδα για κάθε περίπτωση.
Για μεγάλη τιμή του grain size τα πράγματα είναι διαφορετικά. Πλέον πιο κοστοβόρα σε ενέργεια δεν είναι πλέον τα mutexes, αλλά η χρήση του μηχανισμού κλειδώματος TTAS. Αυτό οφείλεται στο ότι, όπως ειπώθηκε προηγουμένως, για μεγάλο grain size, οι ανάγκες συγχρονισμού είναι

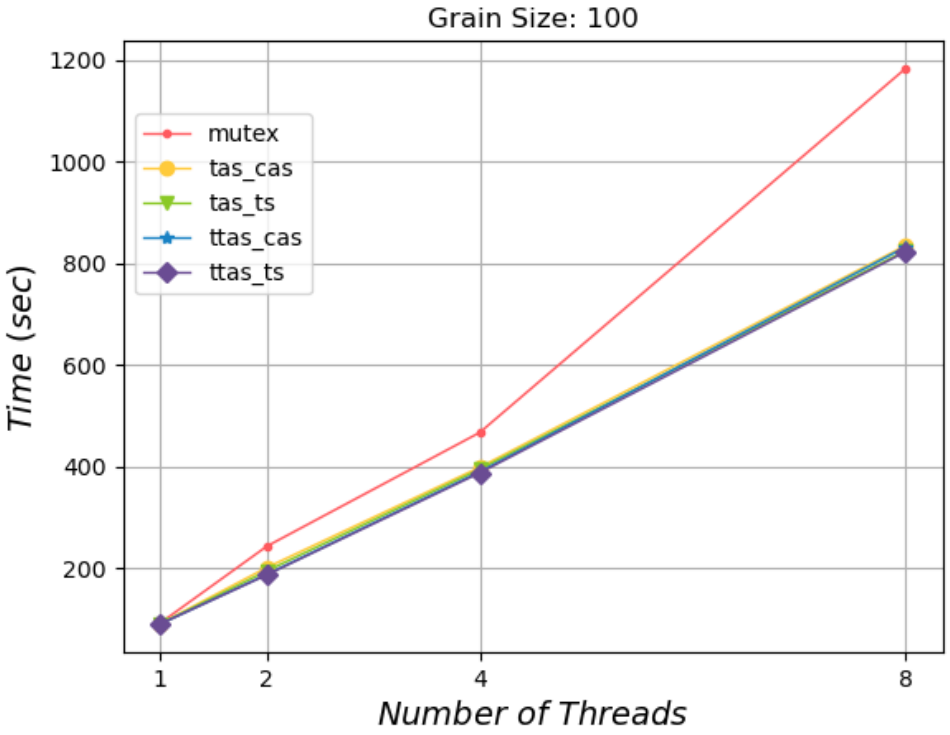
μικρότερες, οπότε η μέθοδος του busy-waiting-loop(συνεχή περιστροφή των επεξεργαστών) που χρησιμοποιείται από τα spinlocks, οδηγεί σε σπατάλη πόρων του συστήματος, που με το κοίμισμα των διεργασιών, στην περίπτωση των mutexes, αυτή αποφεύγεται. Η μέθοδος TTAS επίσης εμφανίζεται ενεργειακά πιο κοστοβόρα και από την μέθοδο TAS, πράγμα μου μπορεί να οφείλεται στα συνεχή διαβάσματα της κατάστασης του lock που κάνει ο πρώτος, καταναλώνοντας ενέργεια.

4.1.4.

Εκτελούμε τις προσομοιώσεις σε VM με 8 πυρήνες και με αριθμό επαναλήψεων ίσων με 120.000.000. Προκύπτουν λοιπόν τα εξής διαγράμματα:



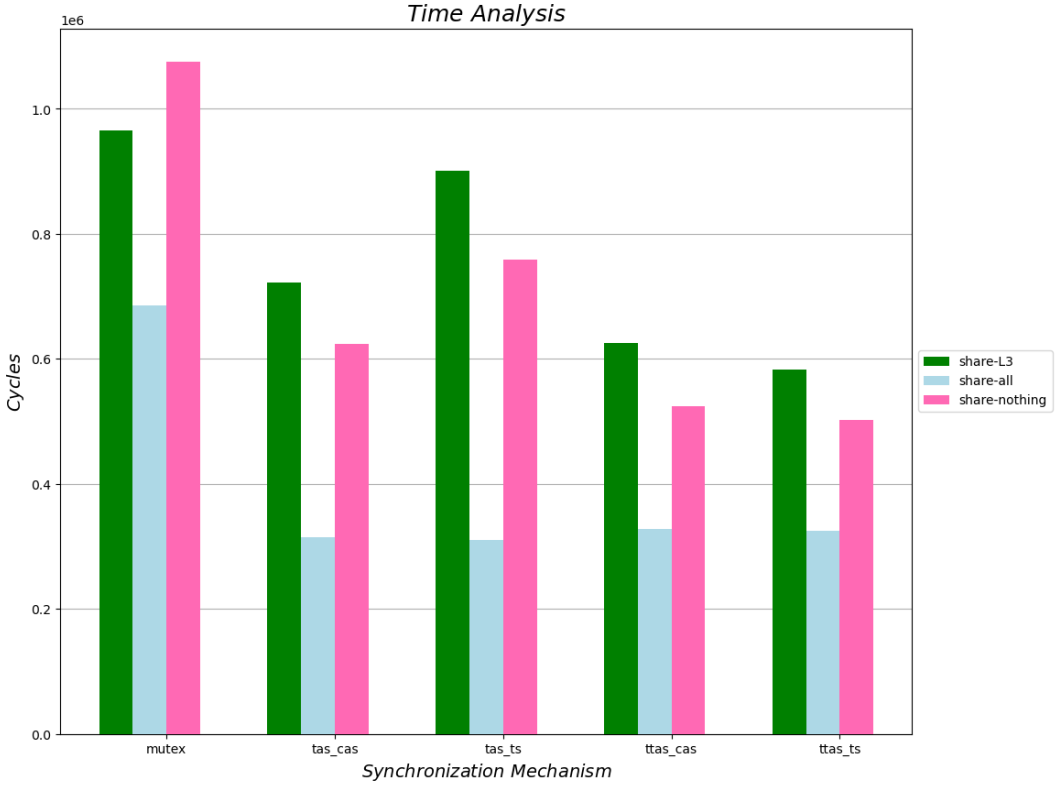


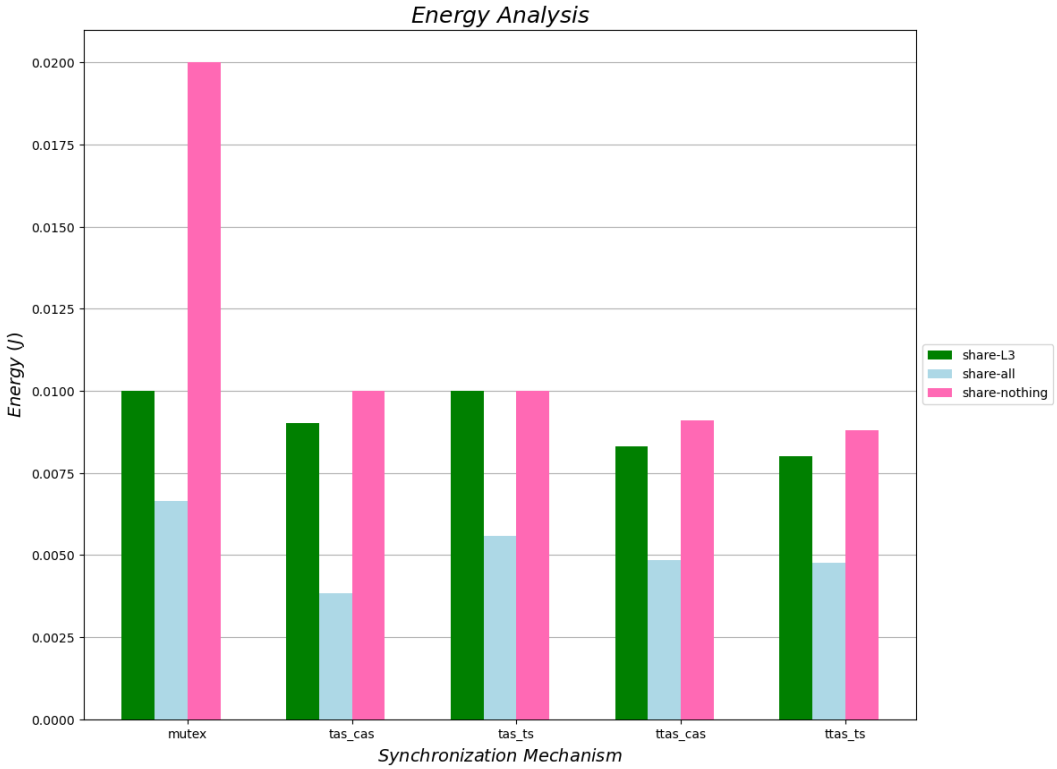


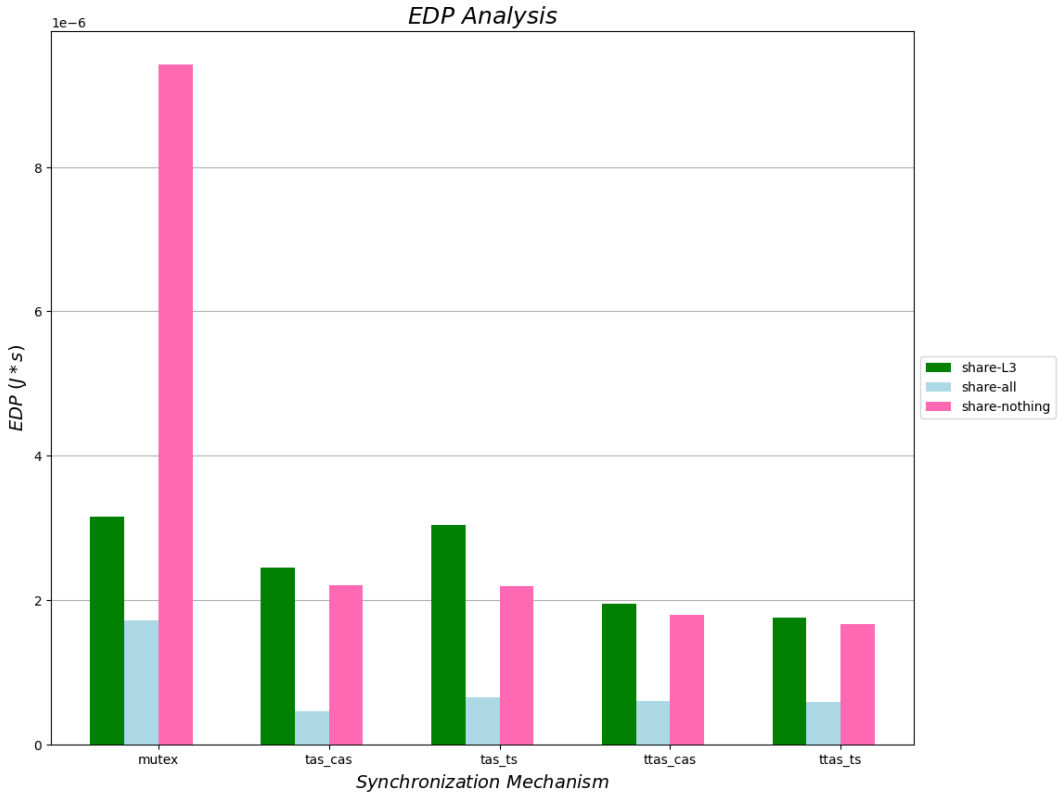
Παρατηρούμε ότι για grain size=1 και πλήθος threads ≥ 4, μη αναμενόμενα βλέπουμε ότι με τη χρήση mutexes ο συνολικός χρόνος εκτέλεσης είναι μικρότερος. Αυτό έρχεται σε μεγάλη αντίθεση με τα αποτελέσματα που λάβαμε από το πρώτο ερώτημα. Αυτό ίσως να οφείλεται στο γεγονός ότι το κρίσιμο τμήμα κώδικα είναι πολύ μικρό, και άρα το κλείδωμα είναι διαθέσιμο για το επόμενο thread πολύ γρήγορα, οπότε σπάνια θα χρειαστεί να γίνει context switch, δηλαδή ένα thread να αναγκαστεί να κοιμηθεί επειδή το κρίσιμο τμήμα κώδικα είναι κλειδωμένο. Για τις άλλες δύο τιμές του grain size παρατηρούμε ανάλογα αποτελέσματα και συμπεράσματα με όσα παρουσιάστηκαν στο πρώτο ερώτημα.

4.2 Τοπολογία νημάτων

Παρακάτω παρατίθενται τα ζητούμενα διαγράμματα με τις μετρήσεις για τις διάφορες τοπολογίες:







Σε κάθε περίπτωση, καλύτερη κλιμακωσιμότητα πετυχαίνουμε με την τοπολογία share-all. Αυτό οφείλεται στο γεγονός ότι η πρόσβαση στην κοινή L2 cache,στην περίπτωση κάποιου cache miss, είναι πολύ πιο γρήγορη απ' ότι εκείνη σε μία πιο απομακρυσμένη L3 cache(στην περίπτωση της share-L3) ή ακόμα περισσότερο, απ' ότι στην κύρια μνήμη (RAM) (στην περίπτωση της share-nothing). Γενικότερα, σε μια κρυφή μνήμη όσο πιο απομακρυσμένη είναι αυτή από τον πυρήνα, τόσο πιο μεγάλη και αργή είναι και τόσο περισσότερος χρόνος απαιτείται για την πρόσβαση σε αυτή. Σε ό,τι αφορά την κατανάλωση ενέργειας, τόσο απ' το διάγραμμα της ενέργειας όσο και του δείκτη EDP παρατηρούμε ότι μικρότερη κατανάλωση για κάθε τύπο κλειδώματος συναντάμε πάλι στην τοπολογία share-all. Αυτό οφείλεται στο γεγονός ότι η πρόσβαση στα δεδομένα είναι γρήγορη και αποδοτική, λόγω της κοινής L2 cache. Στους άλλους δύο τύπους τοπολογιών, στην περίπτωση των μεθόδων κλειδώματος με spinlocks, τα επίπεδα κατανάλωσης ενέργειας είναι παρεμφερή. Στην περίπτωση του mutex, στην τοπολογία share-nothing συναντάμε αρκετά μεγαλύτερη κατανάλωση ενέργειας σε σχέση με αυτή στις υπόλοιπες τοπολογίες.