ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

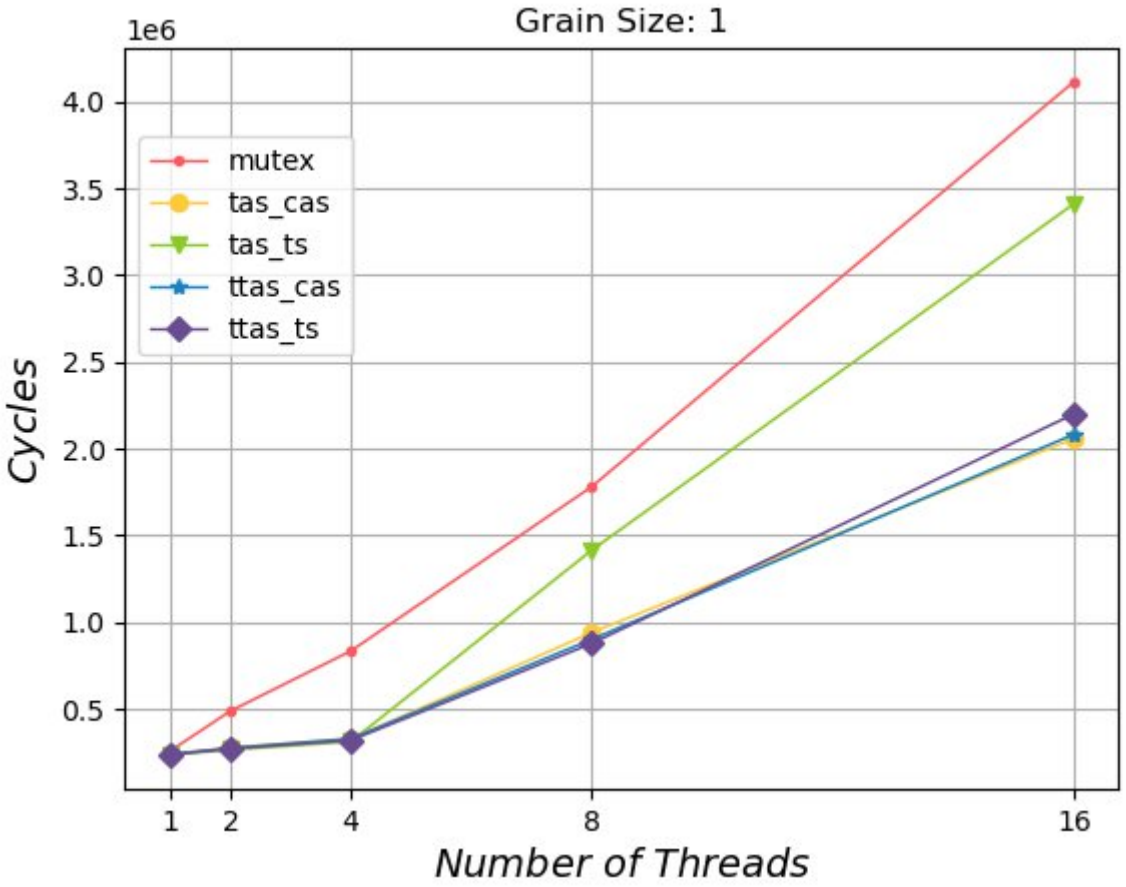
3η ΑΣΚΗΣΗ

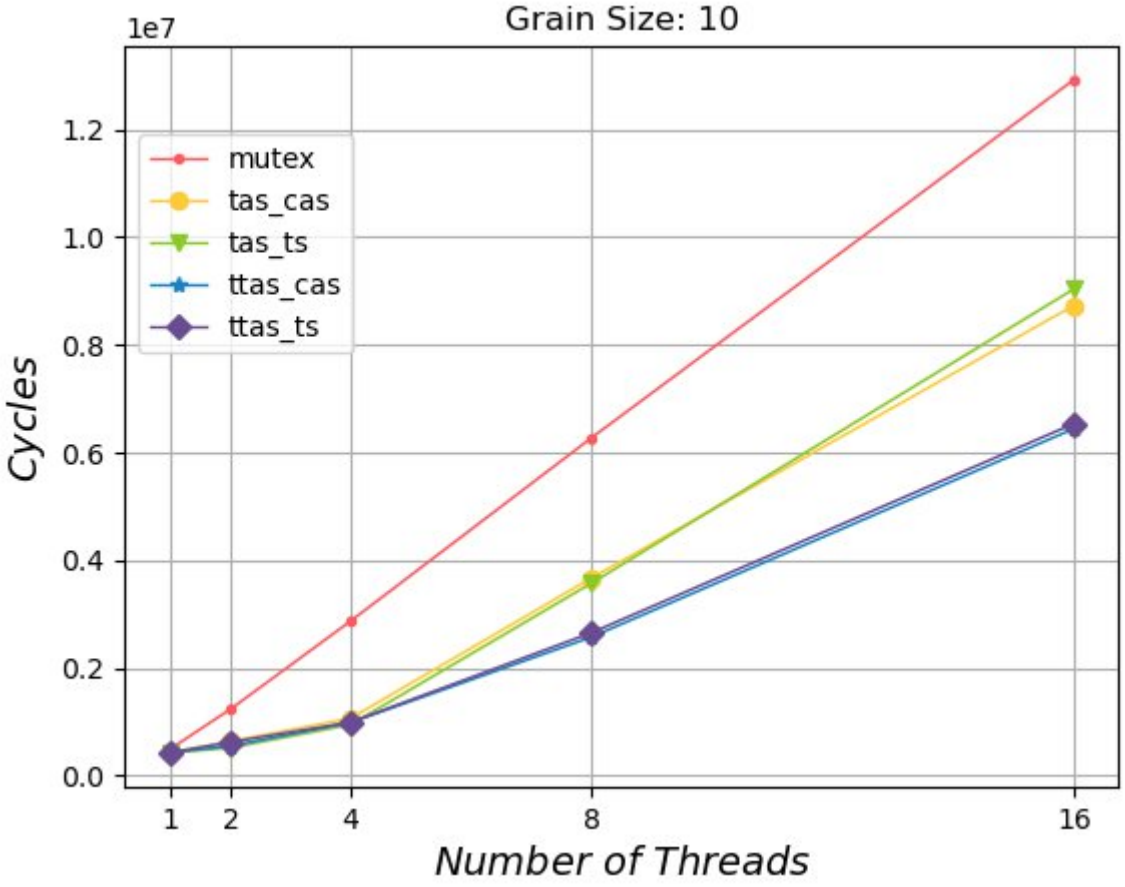
Εμμανουήλ Παντελάκης, 20853

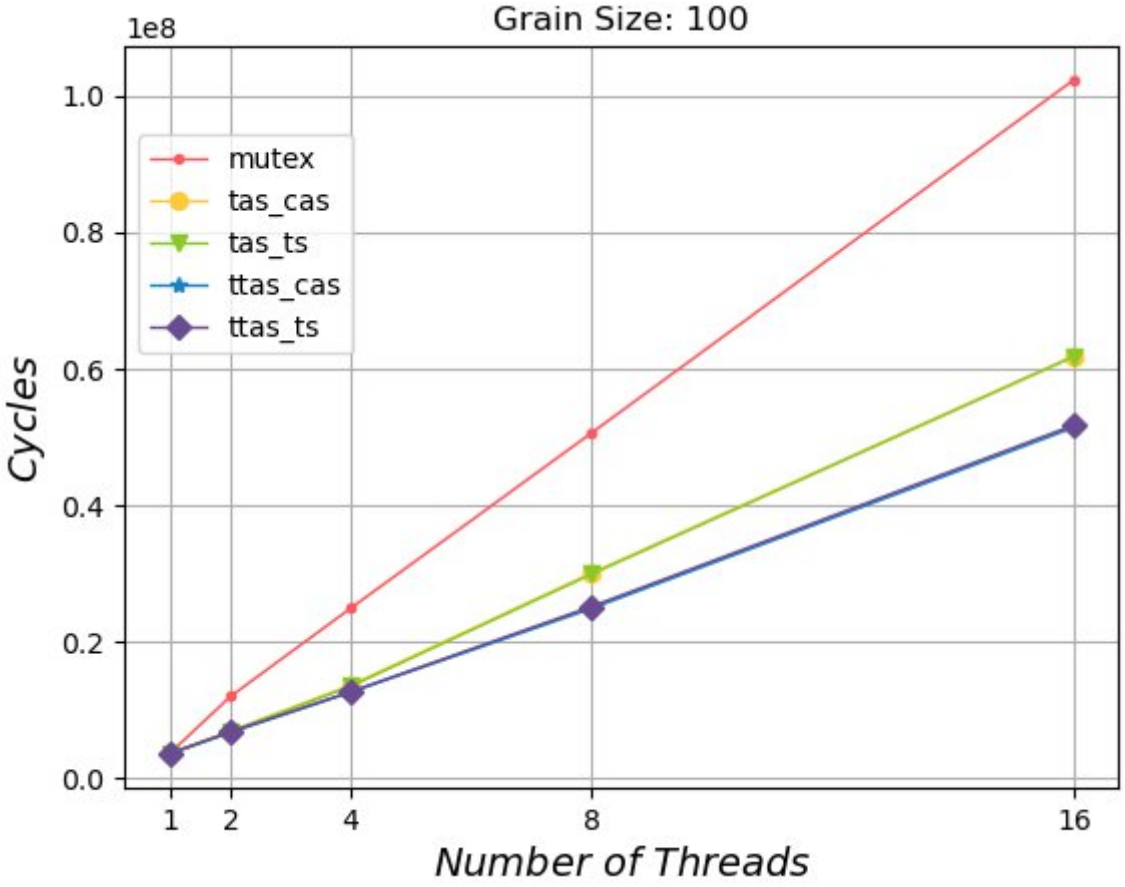
4.1 Σύγκριση Υλοποιήσεων

Ερώτημα 4.1.1

Παρατίθενται τα ζητούμενα διαγράμματα:




Grain Size: 10
1e7
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Cycles
mutex
tas_cas
tas_ts
ttas_cas
ttas_ts
1
2
4
8
16
Number of Threads


Grain Size: 100
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Cycles
mutex
tas_cas
tas_ts
ttas_cas
ttas_ts
1
2
4
8
16
Number of Threads

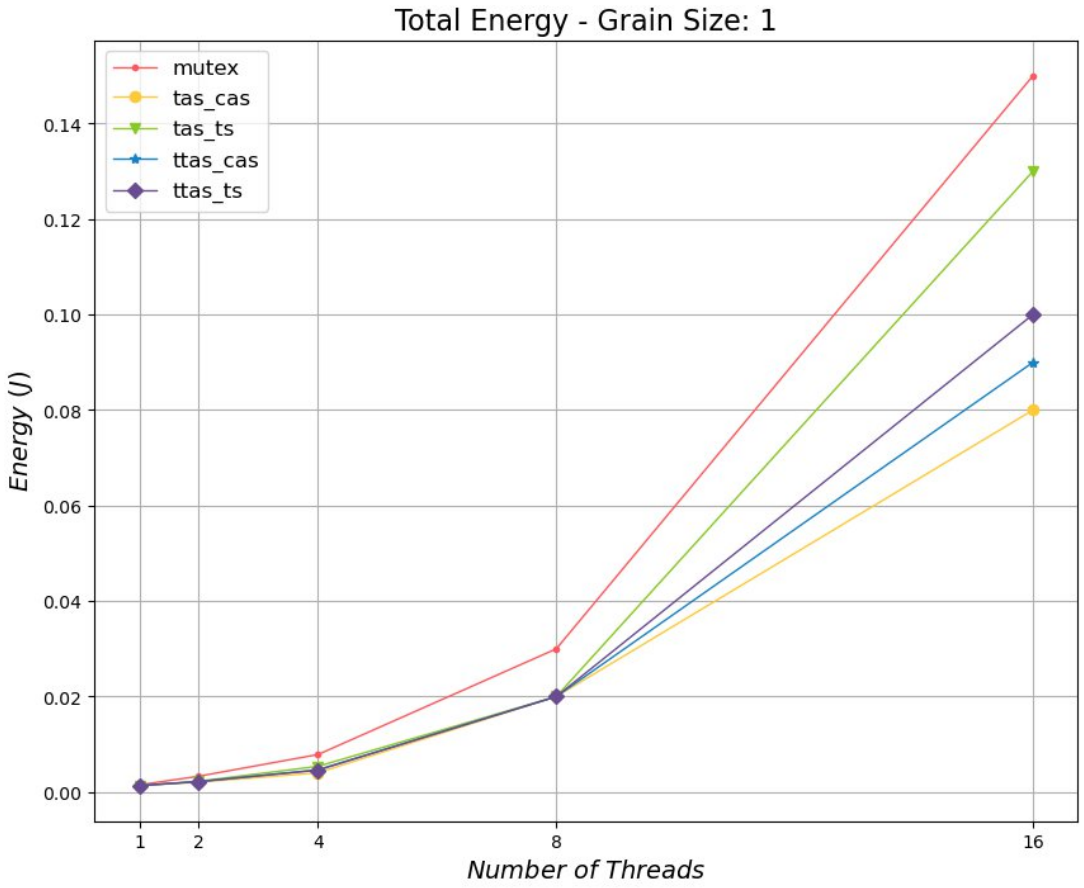
Ερώτημα 4.1.2

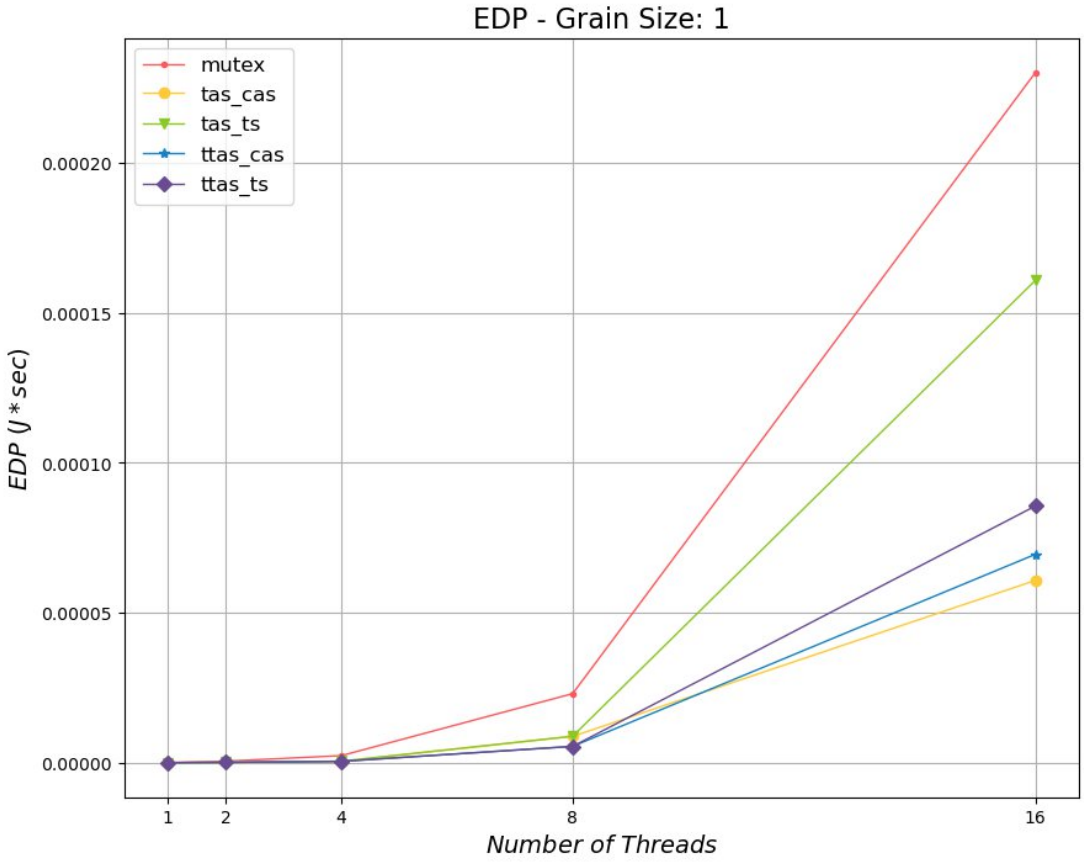
Παρατηρήσεις – Συμπεράσματα:΄

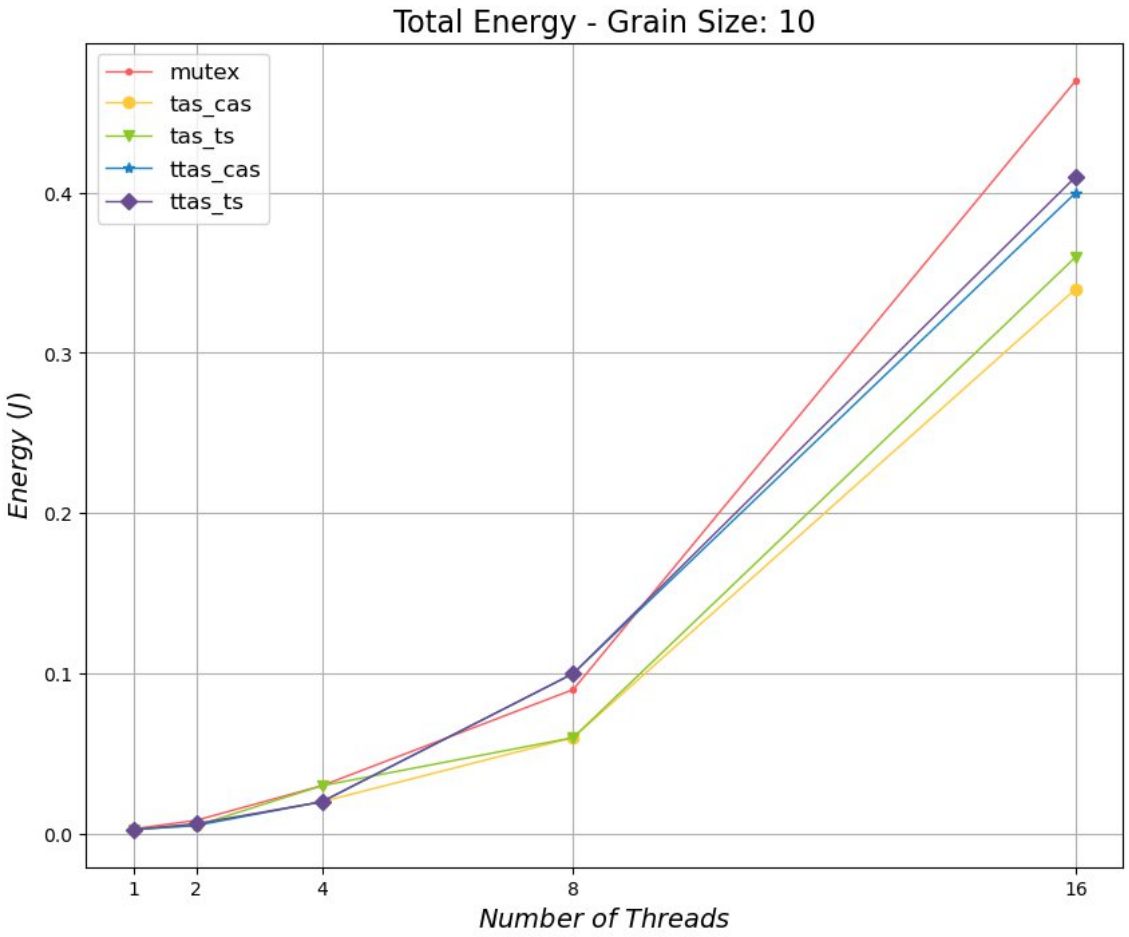
- Η χρήση mutexes ως μηχανισμό κλειδώματος οδήγησε σε περισσότερους κύκλους σε σχέση με την χρήση μηχανισμών κλειδώματος spinlocks (TAS, TTAS), γεγονός που οφείλεται στο context switching που απαιτεί η χρήση mutexes σε αντίθεση με τα spinlocks όπου διατηρούν ενεργό τον επεξεργαστή σε κατάσταση “περιστροφής”.
- Όταν χρησιμοποιείται ο μηχανισμός TAS, οι επεξεργαστές “περιστρέφονται” γύρω από το κλείδωμα και προσπαθούν να το αποκτήσουν δημιουργώντας με αυτό τον τρόπο εγγραφές στη μνήμη. Οι περισσότερες από τις εγγραφές αυτές θα οδηγήσουν σε αστοχίες εγγραφής επειδή κάθε επεξεργαστής προσπαθεί να αποκτήσει τη μεταβλητή κλειδώματος σε αποκλειστική κατάσταση (ατομικά). Αυτό έχεις ως αποτέλεσμα ο ένας επεξεργαστής να ακυρώνει συνεχώς την cache line του άλλου, παράγοντας ιδιαίτερα αυξημένη περιττή κίνηση στο δίαυλο.
- Όταν χρησιμοποιείται ο μηχανισμός TAS, ο επεξεργαστής “περιστρέφεται” γύρω από το κλείδωμα, διαβάζει την μεταβλητή κλειδώματος και ελέγχει εάν αυτό είναι διαθέσιμο. Εάν διαπιστωθεί ότι είναι διαθέσιμο, ο κάθε επεξεργαστής ανταγωνίζεται όλους τους άλλους επεξεργαστές που βρίσκονταν σε κατάσταση περιστροφής για να δει ποιος μπορεί να κλειδώσει ατομικά πρώτος τη μεταβλητή κλειδώματος. Αυτό έχει ως αποτέλεσμα να συμβαίνουν λιγότερα atomic operations και έτσι η κίνηση στο δίαυλο να είναι μειωμένη.
- Από τα παραπάνω καταλαβαίνουμε ότι η χρήση των μηχανισμών κλειδώματος Test-and-Test-and-Set (TTAS) είναι πιθανότερο να οδηγήσει σε καλύτερη κλιμακωσιμότητα σε σχέση με την χρήση των μηχανισμών κλειδώματος Test-and-Set (TAS), όπως επιβεβαιώνεται και από τα γραφήματα του προηγούμενου ερωτήματος.
- Για τους μηχανισμούς TAS και TTAS η χρήση είτε του atomic operation test_and_set είτε του compare_and_swap δεν εμφανίζει κάποια ιδιαίτερη διαφορά στην επίδοση- κυρίως για grain size >1 όπου παρατηρείται σχεδόν ταύτιση των επιδόσεων.
- Σχετικά με το grain size παρατηρούμε ότι όσο αυξάνεται το grain size, η κλιμακωσιμότητα φαίνεται να αποκτά γραμμικότητα, γεγονός που οφείλεται στην μικρότερη ανάγκη για συγχρονισμό καθώς κάθε επεξεργαστής εκτελεί ένα ικανό αριθμό υπολογισμών μέχρι να απελευθερώσει το κλείδωμα και να χρειαστεί εκ νέου συγχρονισμός.

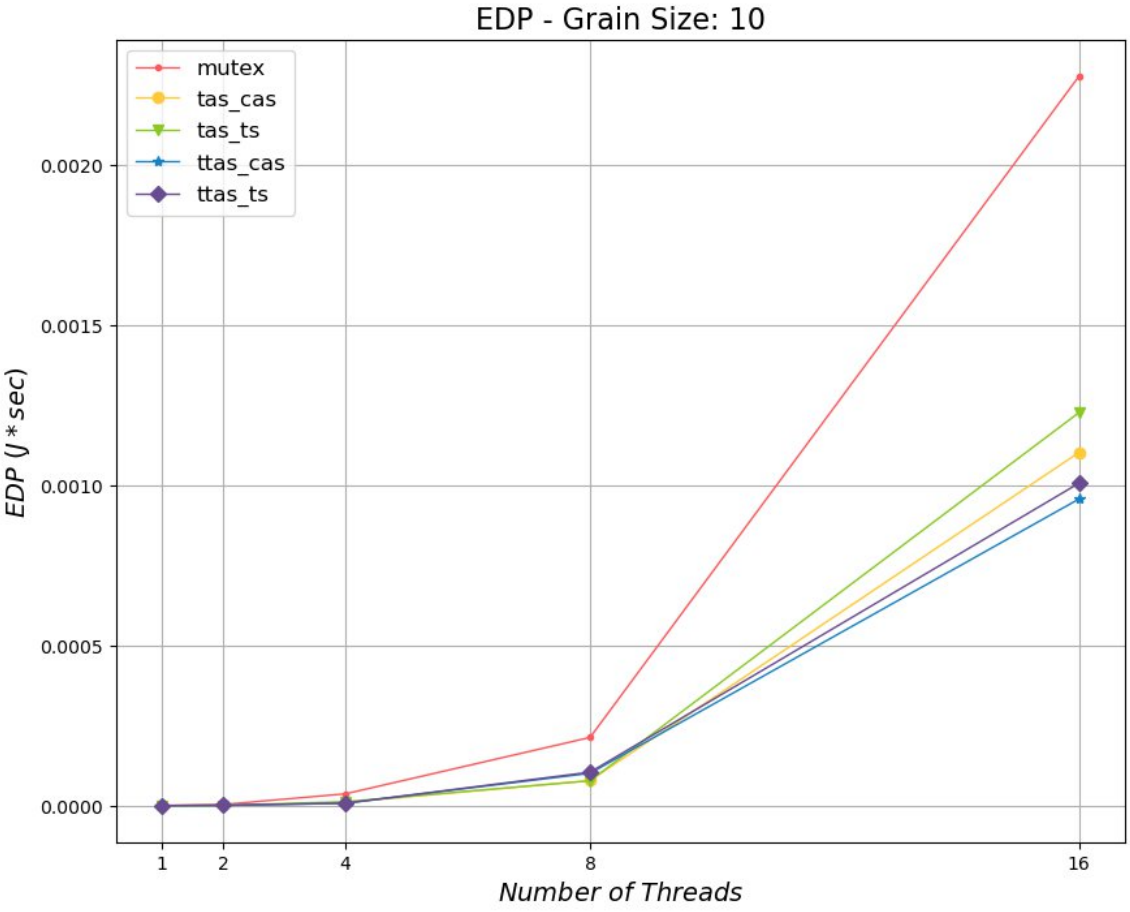
Ερώτημα 4.1.3

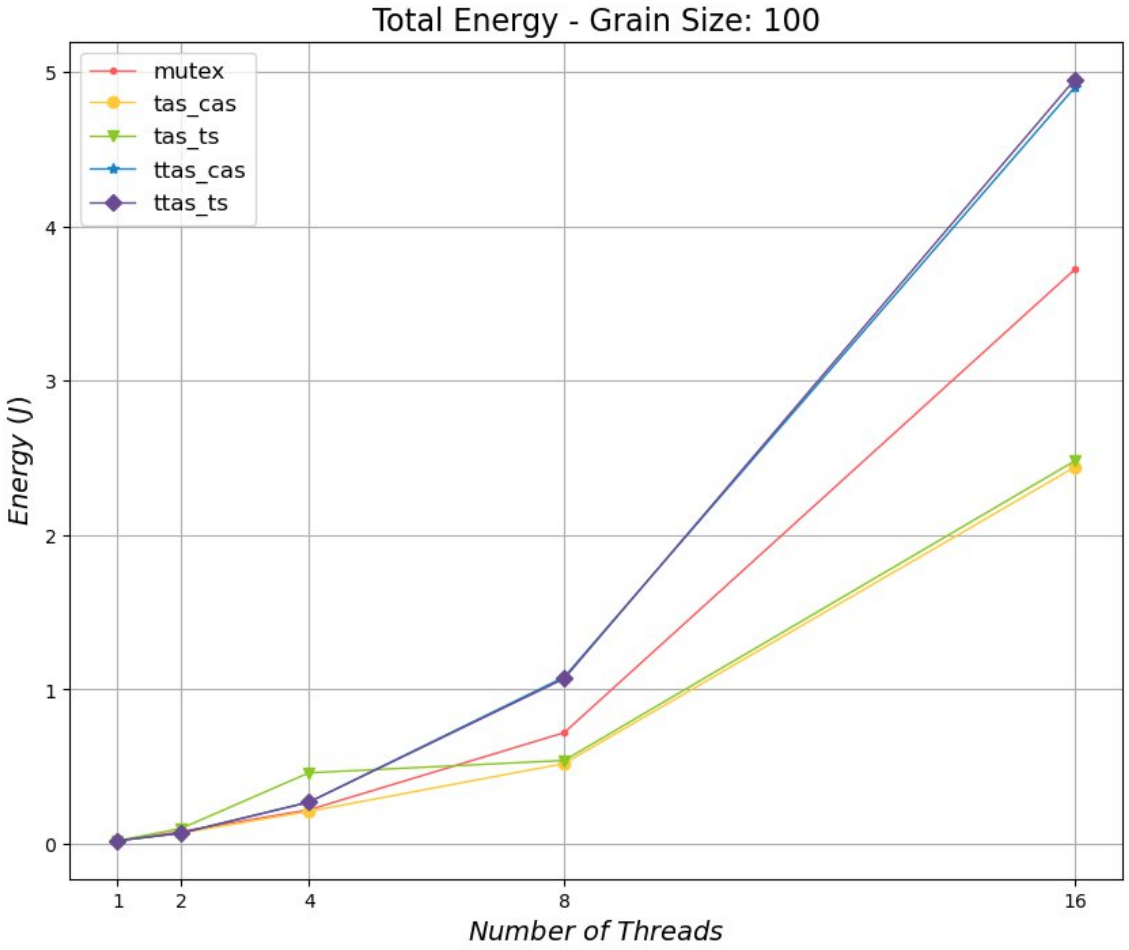
Παρατίθενται τα διαγράμματα κατανάλωσης ενέργειας για τις διαφορετικές υλοποιήσεις αξιοποιώντας ως δείκτες την συνολική κατανάλωση ισχύος (Total Energy) σε Joule καθώς και το EDP:

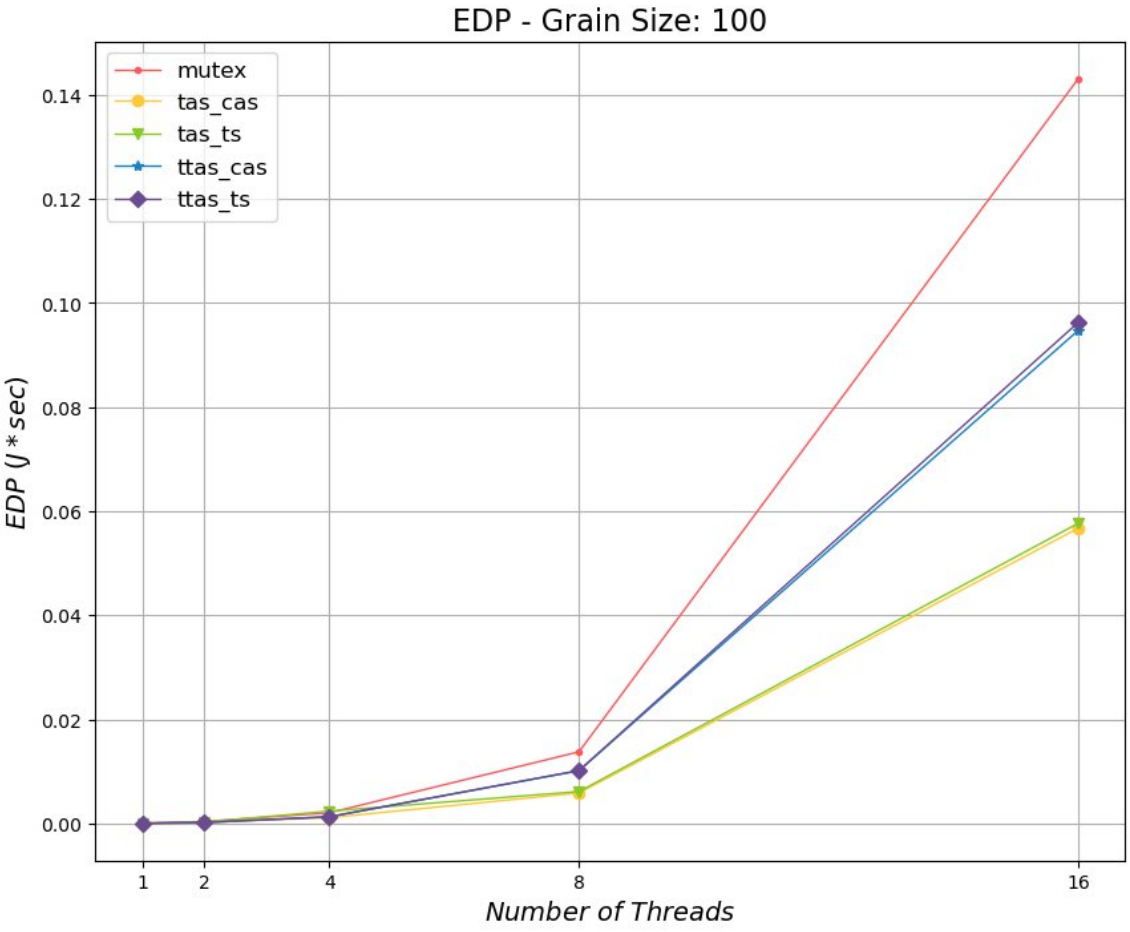





Total Energy - Grain Size: 10
mutex
tas_cas
tas_ts
ttas_cas
ttas_ts
Energy (J)
0.0
0.1
0.2
0.3
0.4
1
2
4
8
16
Number of Threads


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


Total Energy - Grain Size: 100
mutex
tas_cas
tas_ts
ttas_cas
ttas_ts
Energy (J)
0
1
2
3
4
5
1
2
4
8
16
Number of Threads

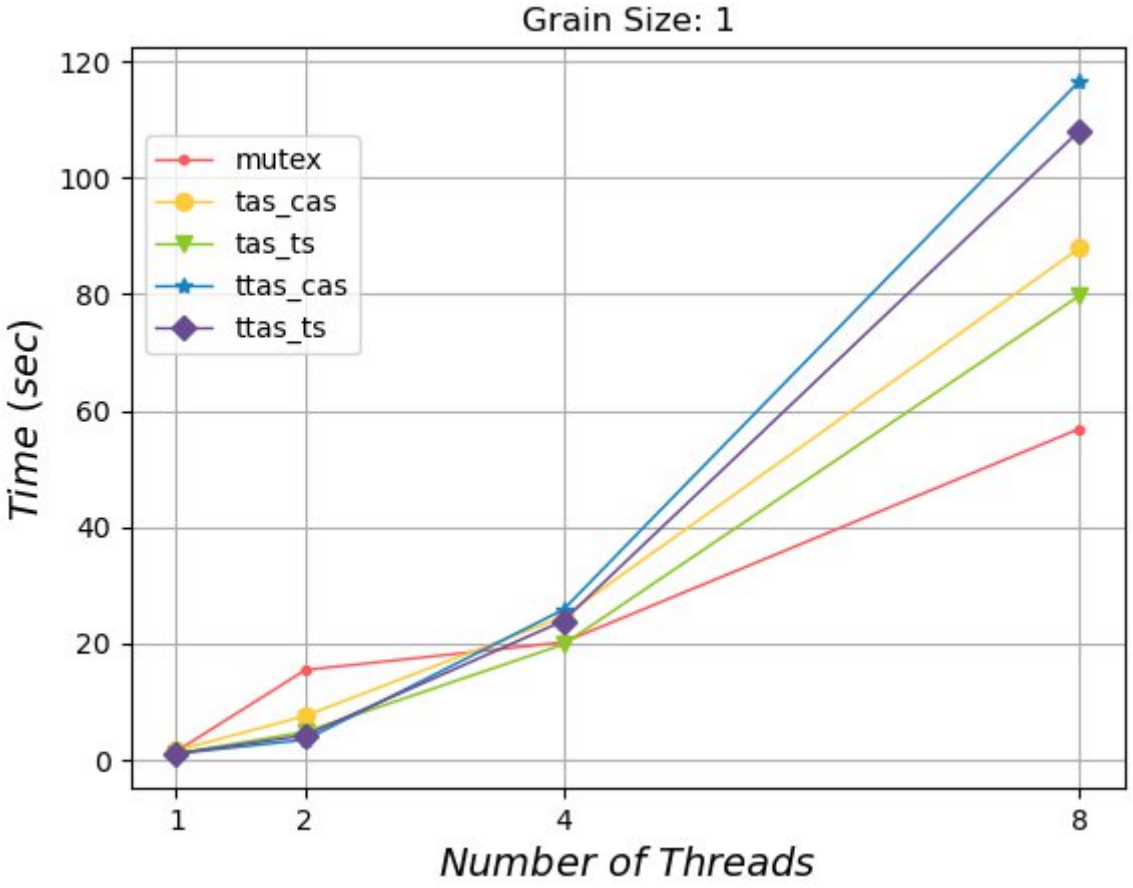

EDP - Grain Size: 100
mutex
tas_cas
tas_ts
ttas_cas
ttas_ts
EDP (J*sec)
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
1
2
4
8
16
Number of Threads

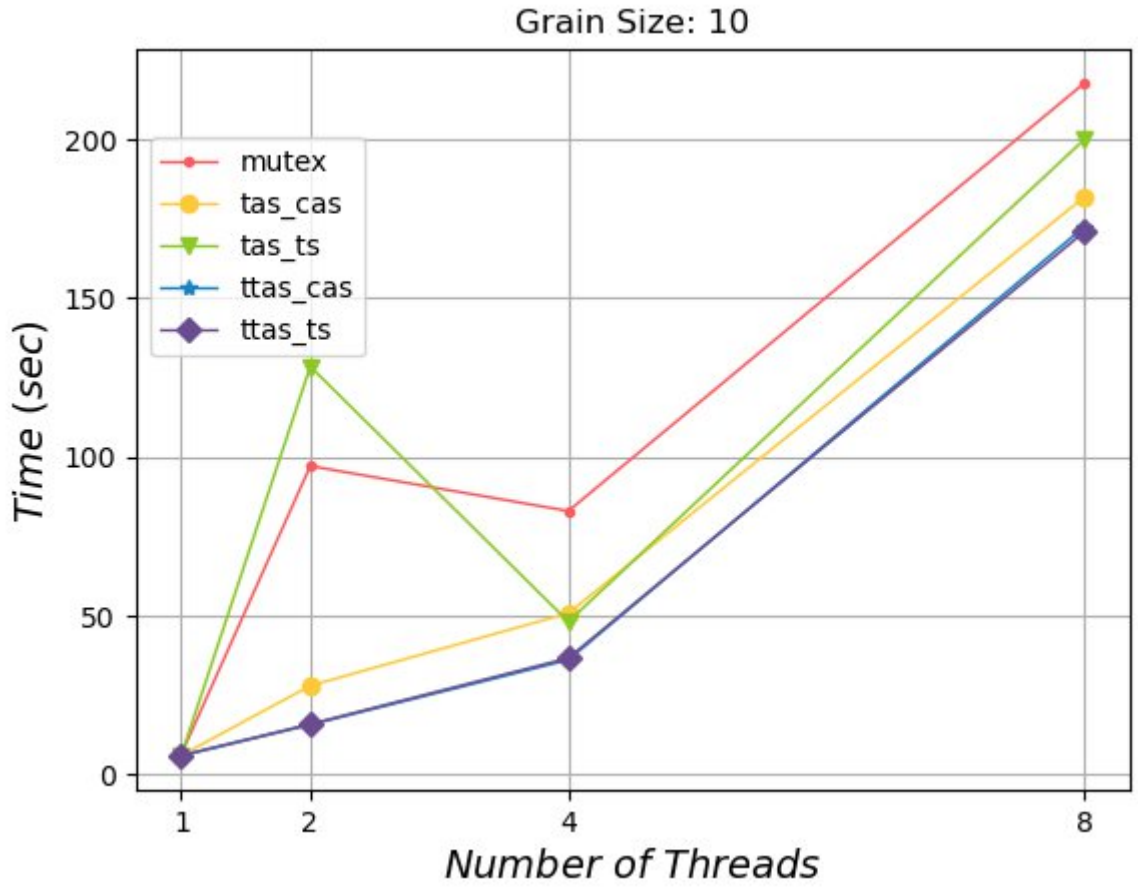
Παρατηρήσεις – Συμπεράσματα:

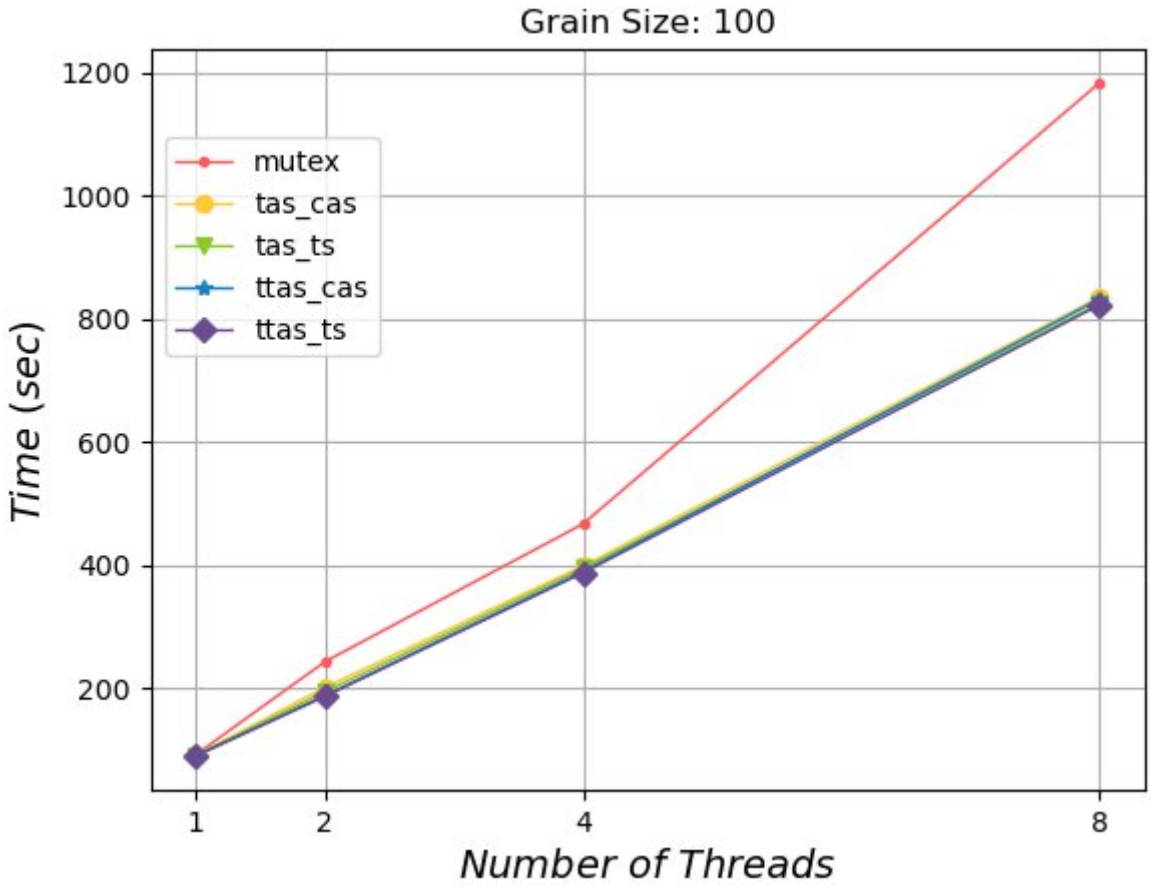
- Για μικρές τιμές grain size παρατηρούμε ότι τα mutexes είναι περισσότερο ενεργειακά κοστοβόρα σε σχέση με τα spinlocks, γεγονός που οφείλεται στο συνεχές context switching που απαιτείται. Όσο αφορά τα spinlocks, για grain size ίσο με 10 και οι δύο υλοποιήσεις παρουσιάζουν παρόμοια ενεργειακή κατανάλωση, ενώ για grain size ίσο με 1 φαίνεται η υλοποίηση TAS να είναι πιο κοστοβόρα.
- Για μεγάλη τιμή του grain size ο μηχανισμός κλειδώματος TTAS εμφανίζει την μεγαλύτερη ενεργειακή κατανάλωση και μάλιστα μεγαλύτερη από τα mutexes, γεγονός που οφείλεται στη συνεχή περιστροφή των επεξεργαστών η οποία λόγω του μεγαλύτερου αριθμού υπολογισμών διαρκεί περισσότερο. Επίσης παρατηρούμε ότι ο μηχανισμός TAS παρουσιάζει την μικρότερη ενεργειακή κατανάλωση, γεγονός που φαίνεται να οφείλεται στα συνεχή διαβάσματα του μηχανισμού TTAS τα οποία λόγω της συχνότητας και του πλήθος τους καθίστανται περισσότερο ενεργειακά κοστοβόρα.

Ερώτημα 4.1.4

Οι προηγούμενες προσομοιώσεις εκτελέστηκαν σε πραγματικό σύστημα με 8 logical cores. Ο αριθμός επαναλήψεων τέθηκε ίσος με 120000000. Παρατίθενται τα αντίστοιχα διαγράμματα:




Grain Size: 10
mutex
tas_cas
tas_ts
ttas_cas
ttas_ts
Time (sec)
0
50
100
150
200
1
2
4
8
Number of Threads


Grain Size: 100
mutex
tas_cas
tas_ts
ttas_cas
ttas_ts
Time (sec)
200
400
600
800
1000
1200
1
2
4
8
Number of Threads

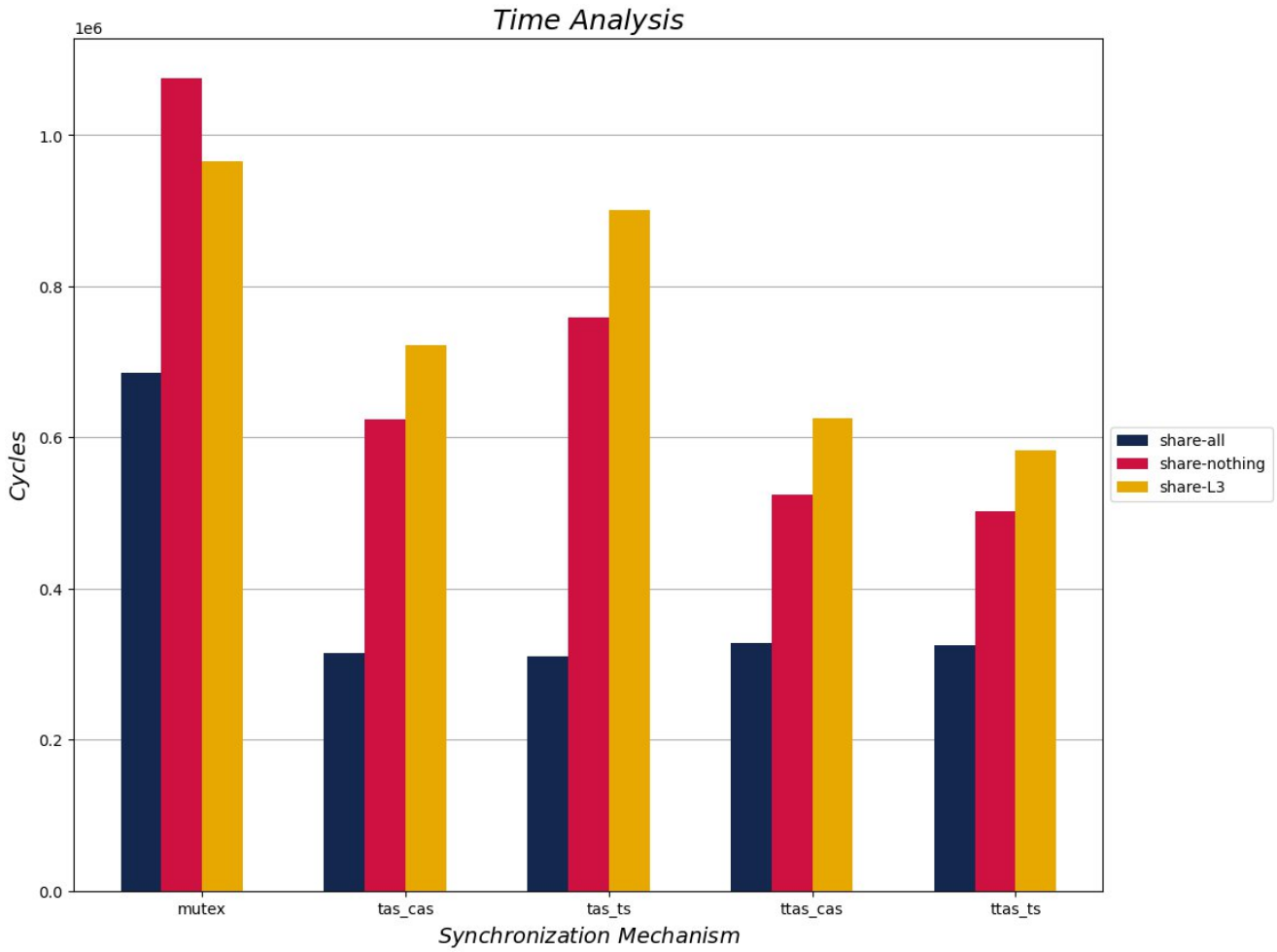
Παρατηρήσεις – Συμπεράσματα:

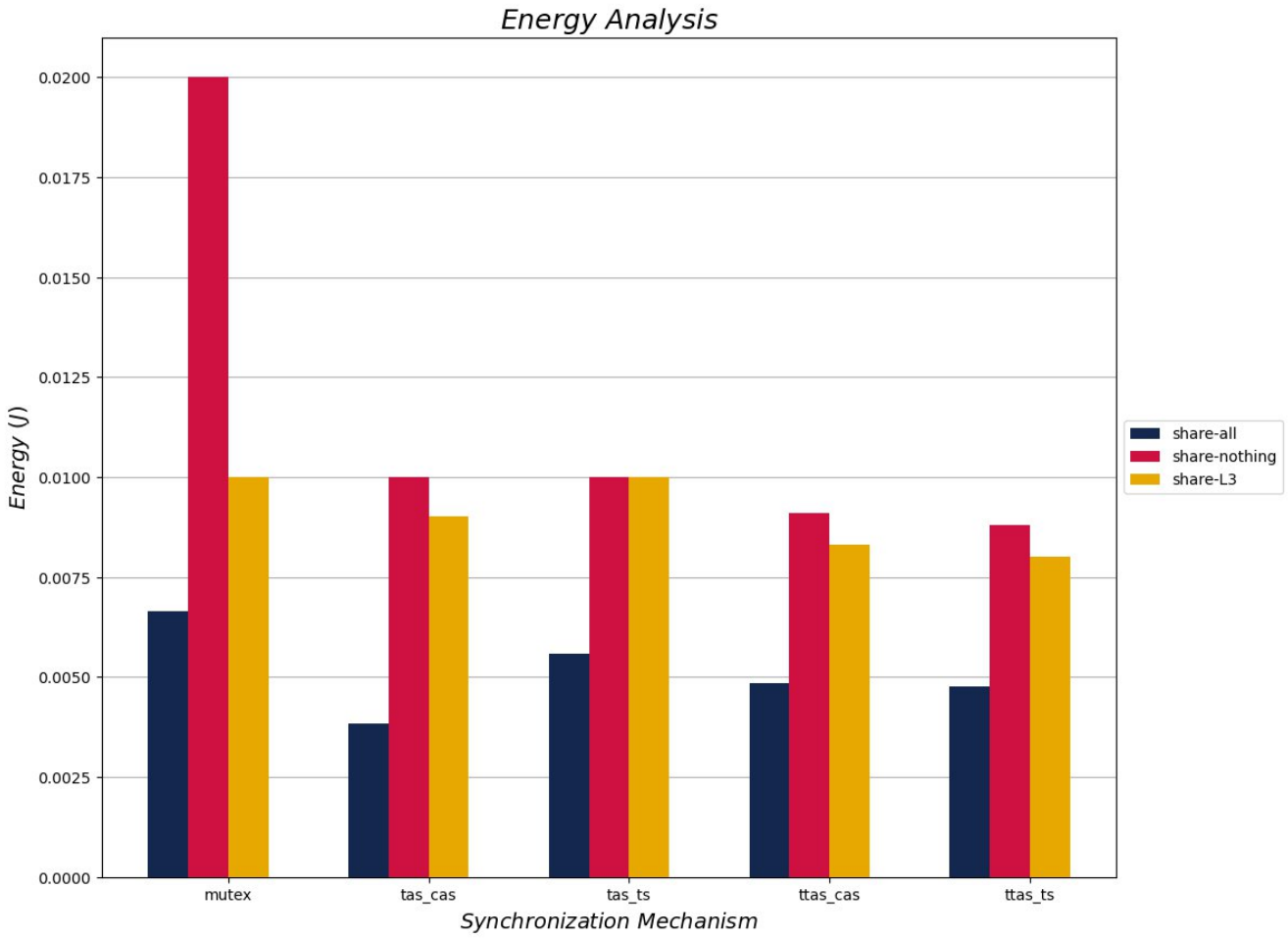
- Παρατηρούμε ότι για grain size ίσο με 1 και αριθμό πυρήνων ≥ 4, σε αντίθεση με τις προσομοιώσεις του 1ου ερωτήματος, ο μηχανισμός κλειδώματος mutex, παραδόξως, παρουσιάζει τον μικρότερο χρόνο εκτέλεσης. Η παρατήρηση αυτή μας οδηγεί στο συμπέρασμα ότι η διάρκεια της εκτέλεσης του κρίσιμου τμήματος είναι πολύ μικρή με αποτέλεσμα το κλείδωμα να εμφανίζεται πολλές φορές και πολύ γρήγορα διαθέσιμο αποφεύγοντας έτσι τις εναλλαγές κατάστασης.
- Για τις μεγαλύτερες τιμές του grain size φαίνεται να υπάρχει συμφωνία με τις προσομοιώσεις του 1ου ερωτήματος, καθώς ο μηχανισμός mutex παρουσιάζει τον μεγαλύτερο χρόνο εκτέλεσης, ενώ οι υπόλοιποι μηχανισμοί spinlocks παρουσιάζουν καλύτερη επίδοση με παρόμοιους (ή και σχεδόν ίδιους) χρόνους εκτέλεση με τον μηχανισμό TTAS να παρουσιάζει την καλύτερη επίδοση. Οι λόγοι που παρατηρούνται αυτές οι επιδόσεις είναι ίδιοι με αυτούς που παρουσιάστηκαν στο 1ο ερώτημα.

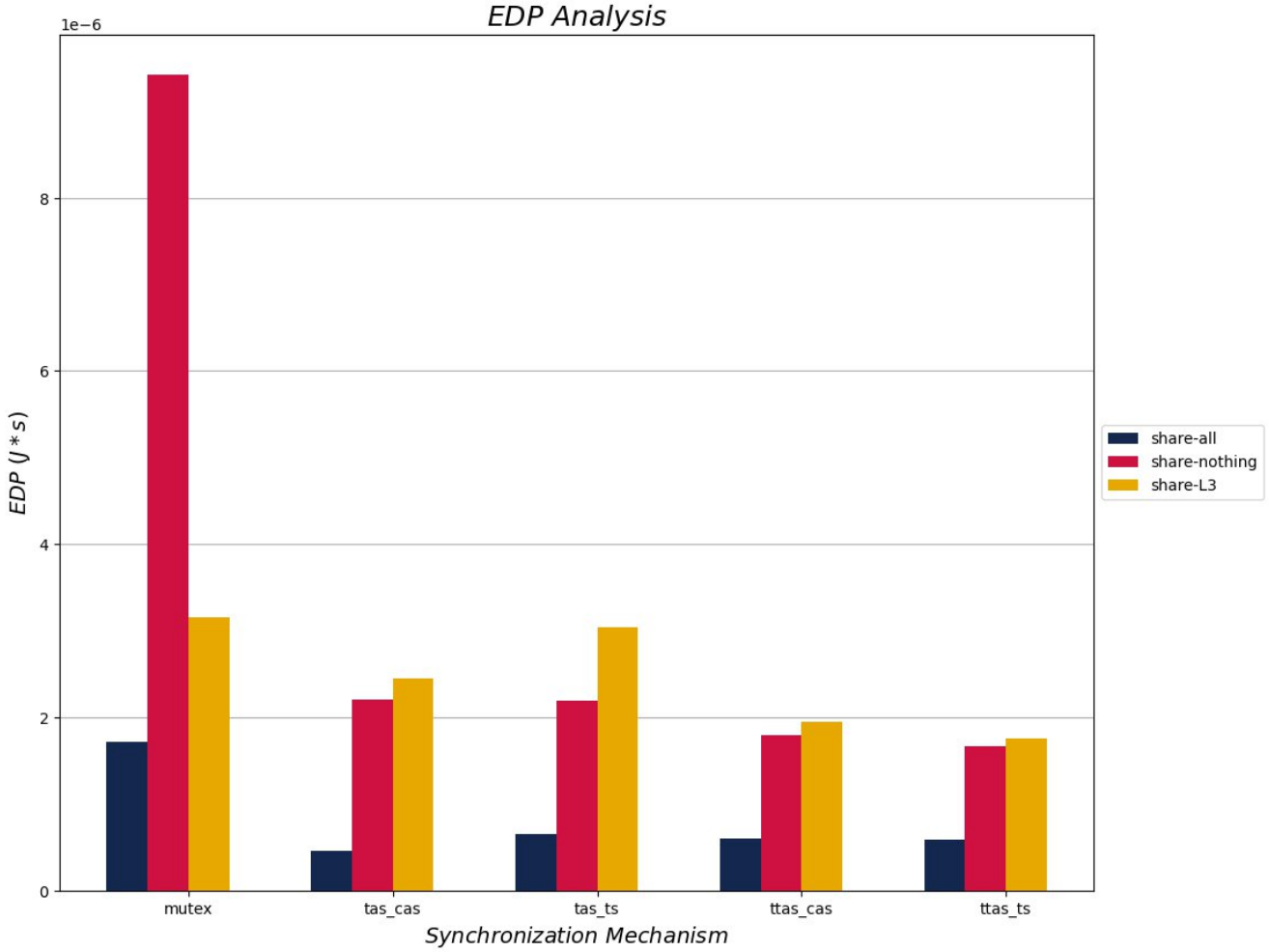
4.2 Τοπολογία νημάτων

Ερώτημα 4.2.1

Παρατίθενται τα ζητούμενα διαγράμματα:




Energy Analysis
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
Energy (J)
mutex
tas_cas
tas_ts
ttas_cas
ttas_ts
Synchronization Mechanism
share-all
share-nothing
share-L3


EDP Analysis
1e−6
8
6
4
2
0
EDP (J * s)
mutex
tas_cas
tas_ts
ttas_cas
ttas_ts
Synchronization Mechanism
share-all
share-nothing
share-L3

Παρατηρήσεις – Συμπεράσματα:

- Η καλύτερη κλιμακωσιμότητα για όλους τους μηχανισμούς κλειδώματος παρατηρείται στην τοπολογία share-all. Αυτό οφείλεται στο γεγονός ότι στην περίπτωση κάποιου cache miss από έναν πυρήνα, η πρόσβαση στην κοινή L2 cache είναι πολύ πιο γρήγορη σε σύγκριση με την πρόσβαση σε μια πιο απομακρυσμένη L3 cache (share-L3) ή ακόμα περισσότερο στη κύρια μνήμη (RAM) (share-nothing). Όσο πιο απομακρυσμένη από τον πυρήνα είναι μια κρυφή μνήμη τόσο πιο μεγάλη και πιο αργή είναι και τόσο περισσότερος χρόνος απαιτείται για την πρόσβαση σε αυτή.
- Ως προς την κατανάλωση ενέργειας, και οι δύο μετρικές (Total Energy και EDP) δείχνουν ότι η μικρότερη κατανάλωση παρατηρείται στην τοπολογία share-all για όλους τους διαφορετικούς μηχανισμούς κλειδώματος, γεγονός που οφείλεται στη γρήγορη και αποδοτική πρόσβαση στα δεδομένα λόγω της κοινής L2 cache. Οι τοπολογίες share-L3 και share-nothing στην περίπτωση των μηχανισμών κλειδώματος spinlocks παρουσιάζουν παρεμφερή κατανάλωση ενέργειας, ενώ στην περίπτωση του μηχανισμού κλειδώματος mutex η τοπολογία share-nothing παρουσιάζει ιδιαίτερα αυξημένη κατανάλωση συγκριτικά με τις υπόλοιπες τοπολογίες.

Παράρτημα κώδικα

Υλοποίηση Μηχανισμών Κλειδώματος

```
#ifndef LOCK_H_
#define LOCK_H_

typedef volatile int spinlock_t;

#define UNLOCKED  0
#define LOCKED    1

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)
{
    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)
{
    while (1) {
        if (*spin_var == UNLOCKED) {
            if (__sync_lock_test_and_set(spin_var, LOCKED) == UNLOCKED) break;
        }
    }
}


static inline void spin_unlock(spinlock_t *spin_var)
{
    __sync_lock_release(spin_var);
}

#endif
```