



**Σχολή Ηλεκτρολόγων Μηχανικών**

**&**

**Μηχανικών Υπολογιστών**

**Προηγμένα Θέματα Αρχιτεκτονικής Υπολογιστών**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Άσκηση Τέταρτη**

**Μανδηλαράς Νικηφόρος, Α.Μ: 03112012**

**Όγδοο Εξάμηνο**

**Παραδοτέα: 17/ 7/2016**



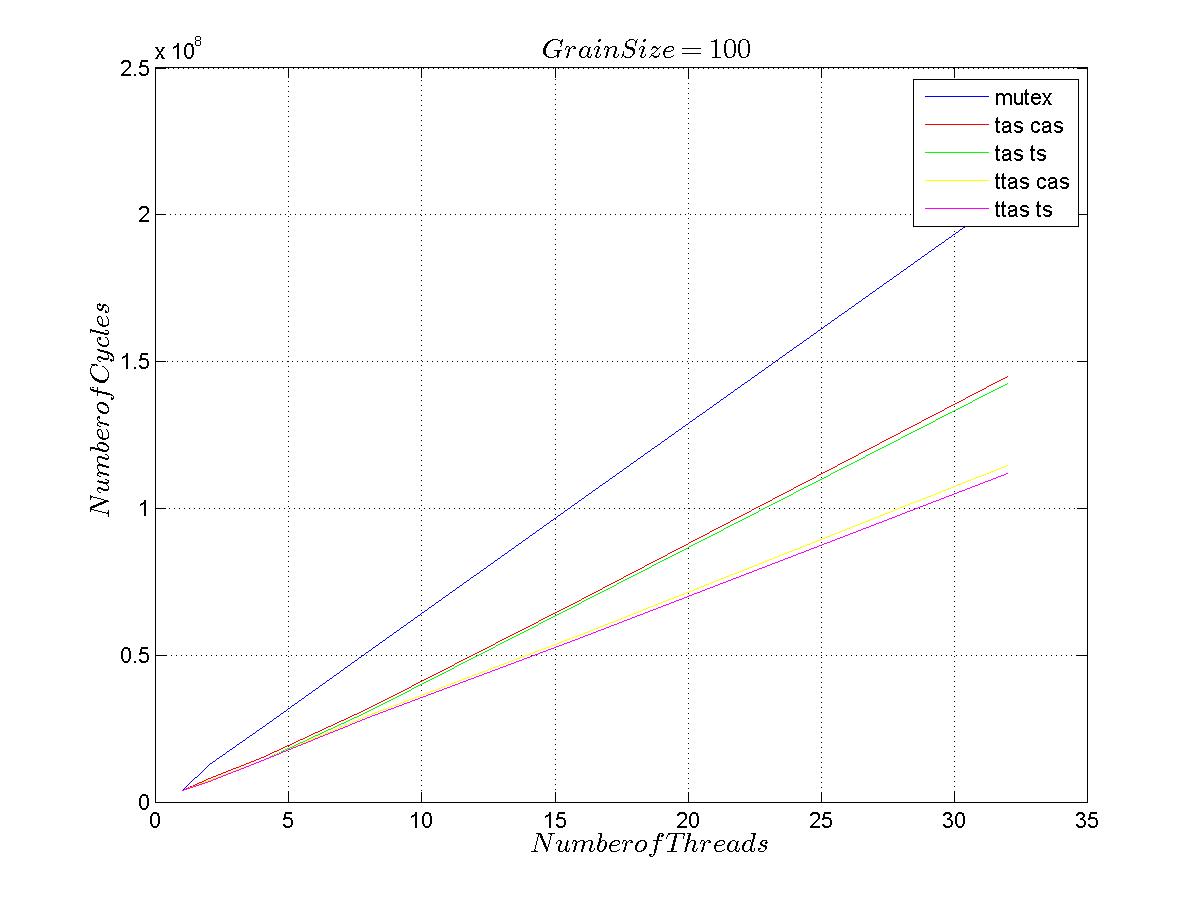
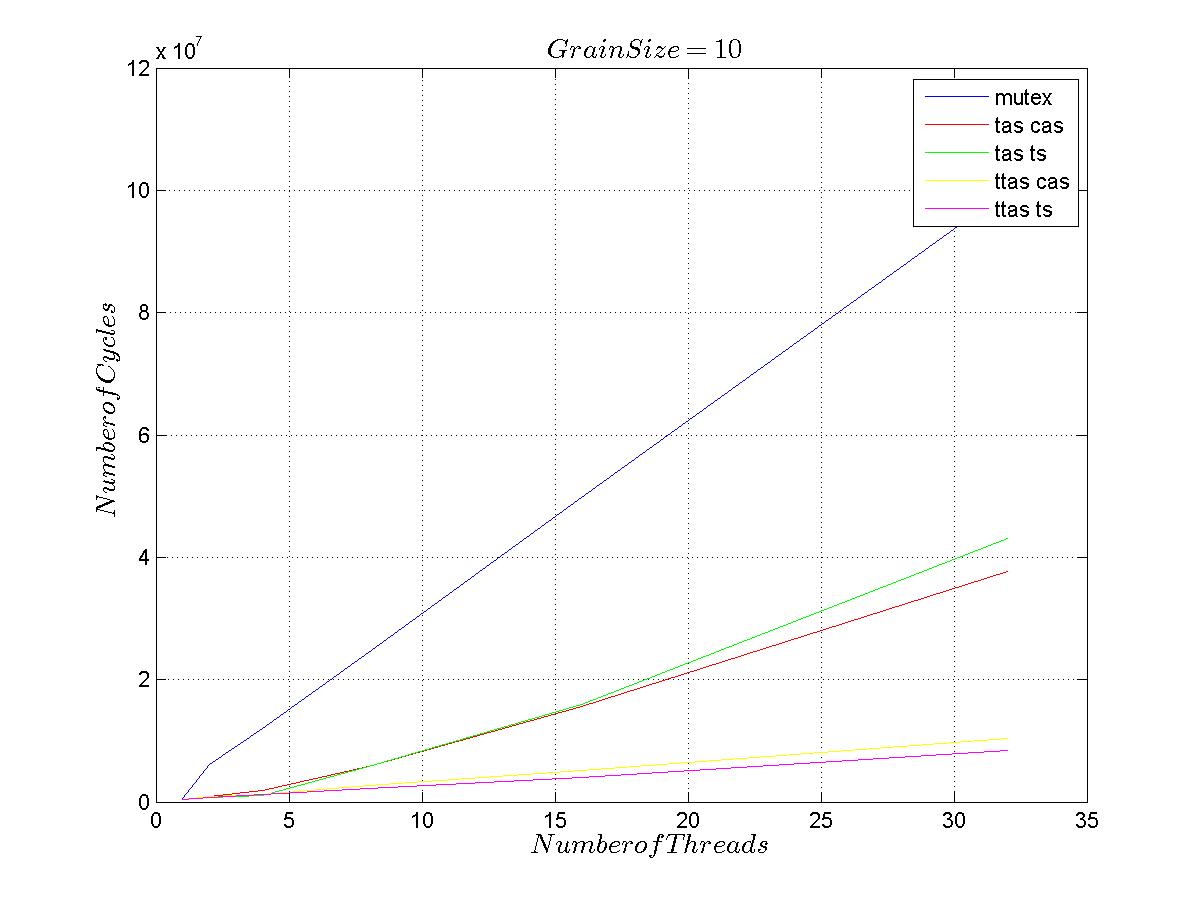
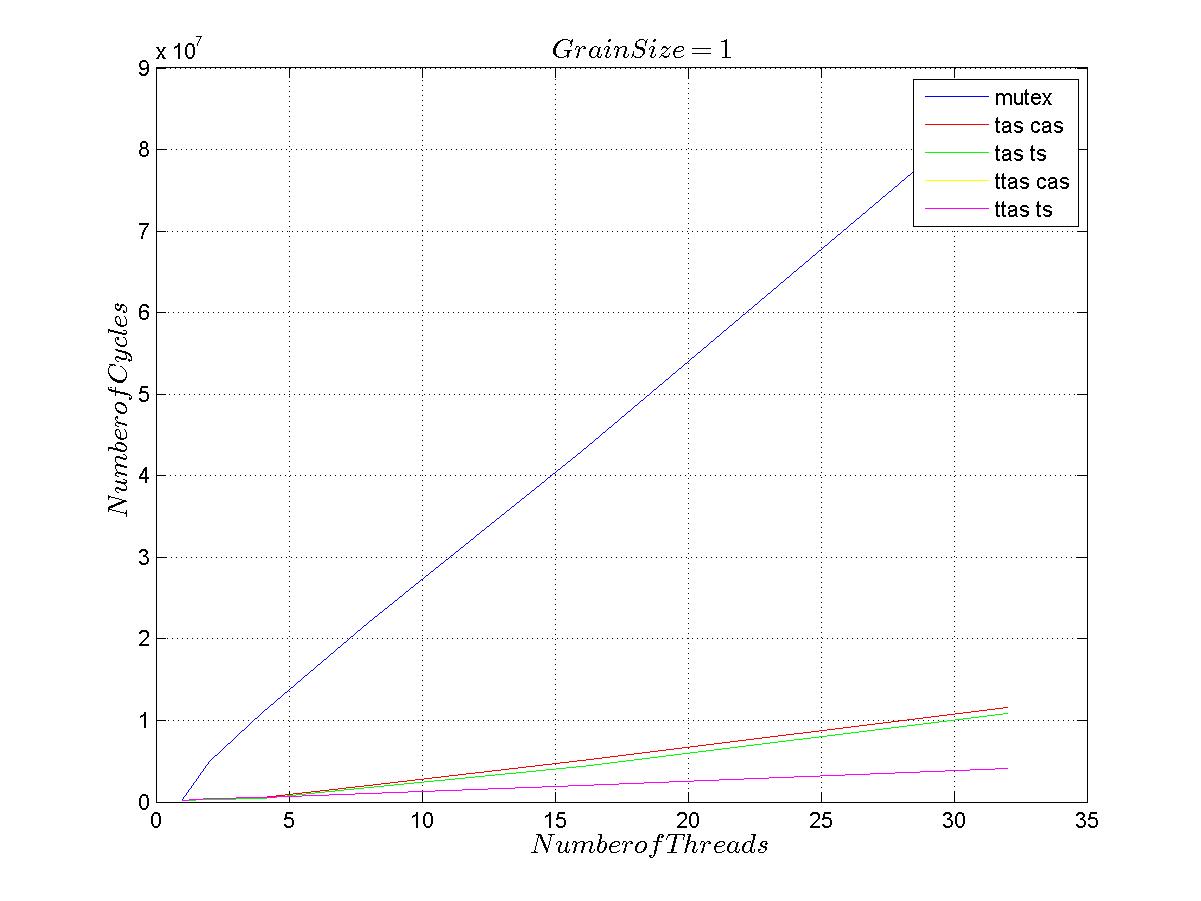
**Περιγραφή και Σκοπός της Άσκησης**

Σ’ αυτή την άσκηση καλούμαστε να εκτελέσουμε μια σειρά μετρήσεων προκειμένου να εκτιμήσουμε την επίδραση των διάφορων μηχανισμών συγχρονισμού σε πολυπύρηνα συστήματα. Η μελέτη αυτή θα γίνει σε συνδυασμό και με την εκτίμηση άλλων παραμέτρων όπως ο διαμοιρασμός των επιπέδων κρυφής μνήμης, ο αριθμός των threads που χρησιμοποιούνται κάθε φορά καθώς και για το αν βρισκόμαστε σε πραγματικό σύστημα ή προσομοιωτή.

**3.1.1-2 Μελέτη Κλιμακωσιμότητας**

Στο πρώτο αυτό σκέλος της άσκησης για τους διάφορους μηχανισμούς κλειδωμάτων που αναφέρονται στην εκφώνηση παρουσιάζουμε το χρόνο εκτέλεσης του προγράμματος για διαφορετικό πλήθος νημάτων.( αλλά και σε συστήματα με αντίστοιχο πλήθος επεξεργαστών)

Ταυτόχρονα διαφοροποιούμε και το μέγεθος του κρίσιμου τμήματος. Στη συνέχεια ακολουθούν συγκεντρωτικές γραφικές παραστάσεις για τις διαφορετικές τιμές του κρίσιμου τμήματος.

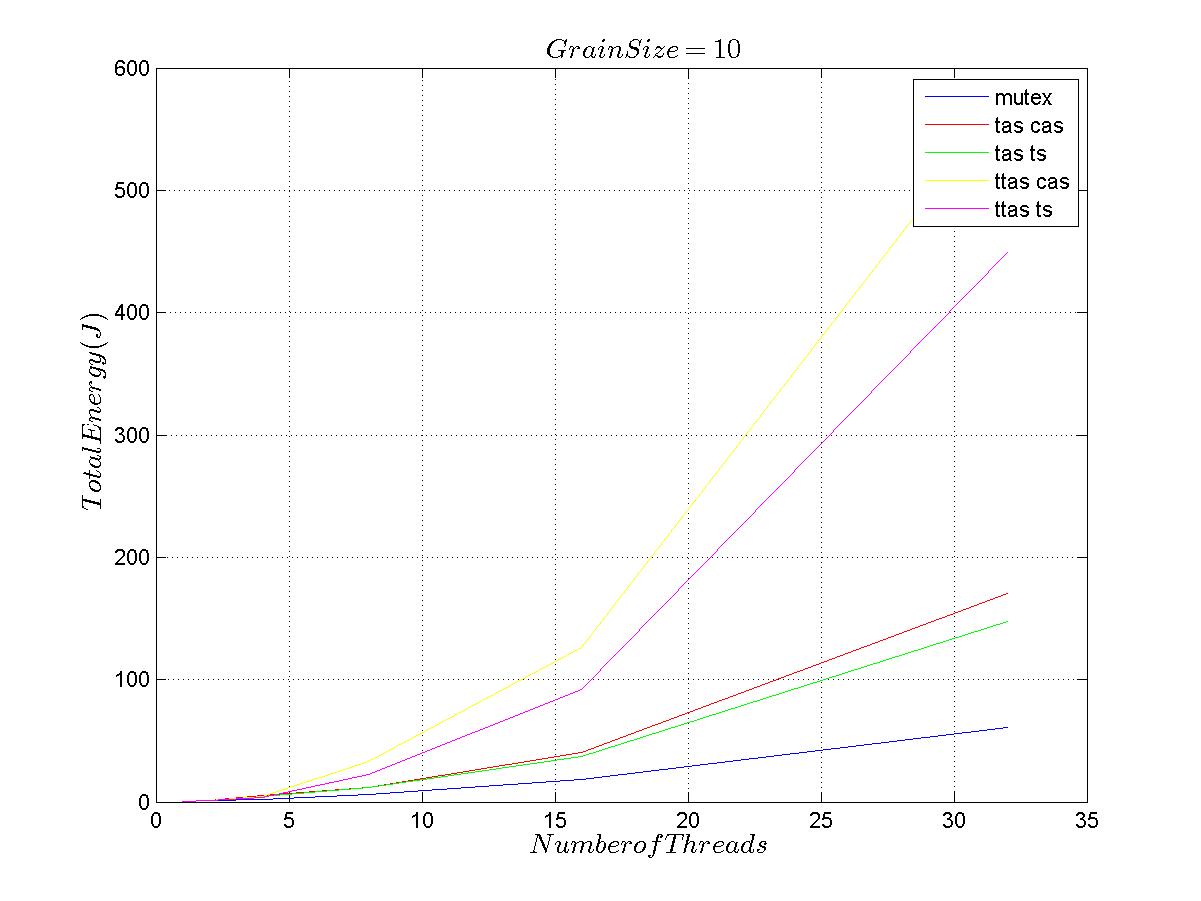
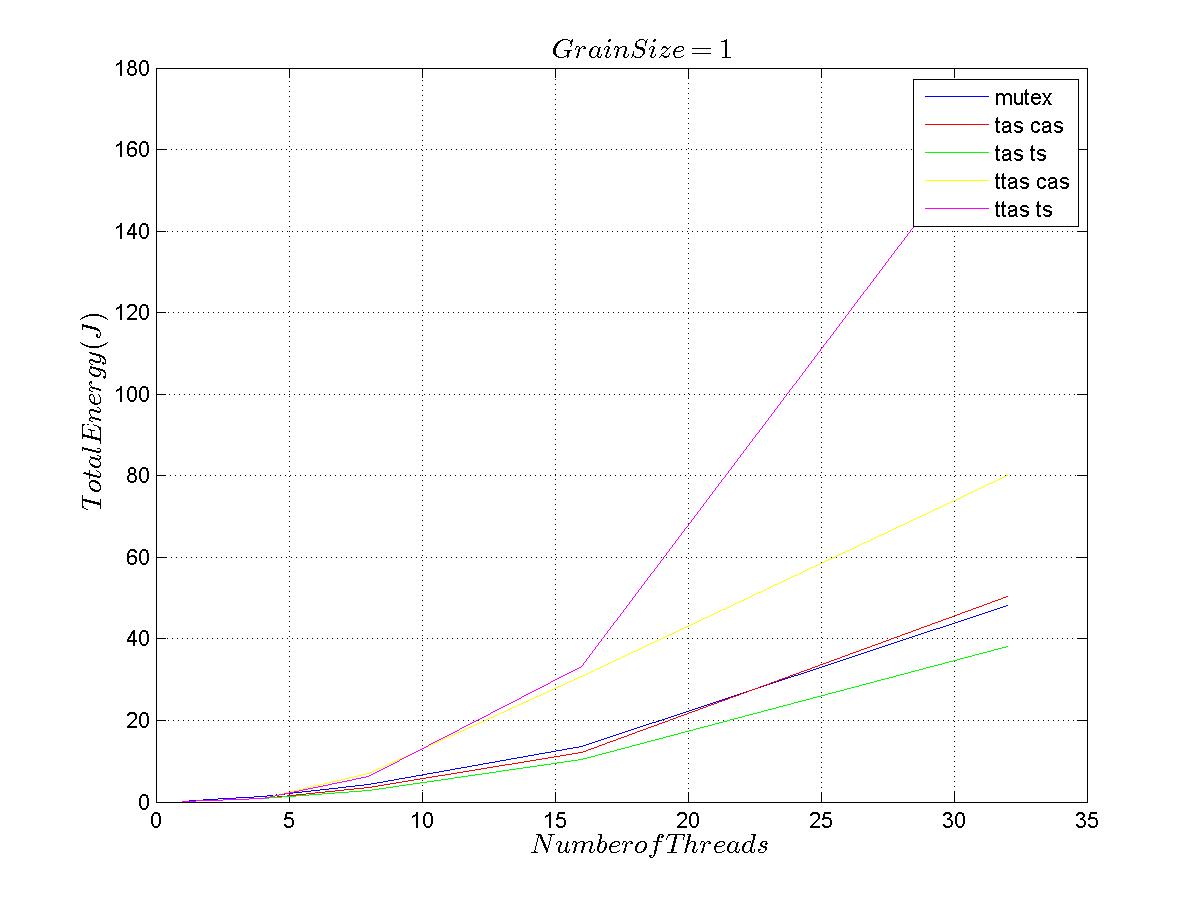
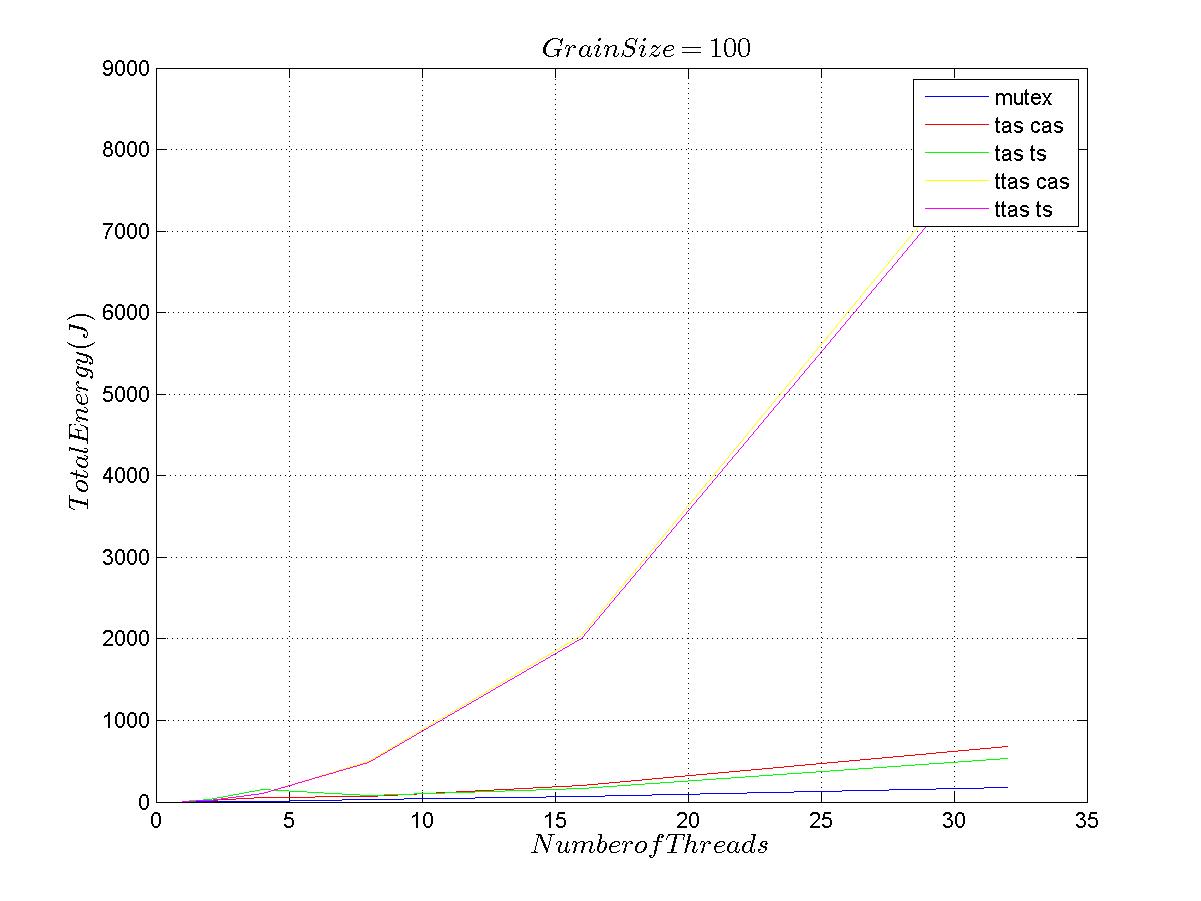


Αυτό που φαίνεται από τα παραπάνω διαγράμματα είναι ότι για μεγάλο αριθμό νημάτων, ο μηχανισμός MUTEX είναι σταθερά πιο αργός σε όλους τους συνδυασμούς, καθώς εκεί το φαινόμενο του busy lock εμφανίζεται αρκετά συχνά.

Παρατηρούμε επίσης ότι οι TAS έχουν χειρότερη απόδοση από τους TTAS. Αυτό έχει προφανή εξήγηση το γεγονός ότι ο TAS σε κάθε εκτέλεσή του από ένα νήμα εκτελεί μία εντολή store η οποία μπορεί να «χαλάει» την cache των πυρήνων που εκτελούν τα υπόλοιπα νήματα αλλά και να δημιουργεί αχρείαστη κίνηση στον δίαυλο. Άρα θα έχει μεγαλύτερες καθυστερήσεις από τον TTAS, ο οποίος παρακολουθεί την μεταβλητή «κλειδί» και μόνο όταν δει ότι αυτή είναι ελεύθερη κινείται να την πάρει, και άρα μόνο τότε εκτελεί store. Αυτό σημαίνει ότι η υλοποίηση που χρησιμοποιεί TAS θα κάνει πολλές store και όταν κάποια στιγμή θα μπαίνει στην κρίσιμη περιοχή το εκάστοτε νήμα θα εκτελεί εντολές, ενώ με τον TTAS περίπου για κάθε store που θα εκτελεί ένα νήμα, θα μπαίνει και στην κρίσιμη περιοχή.

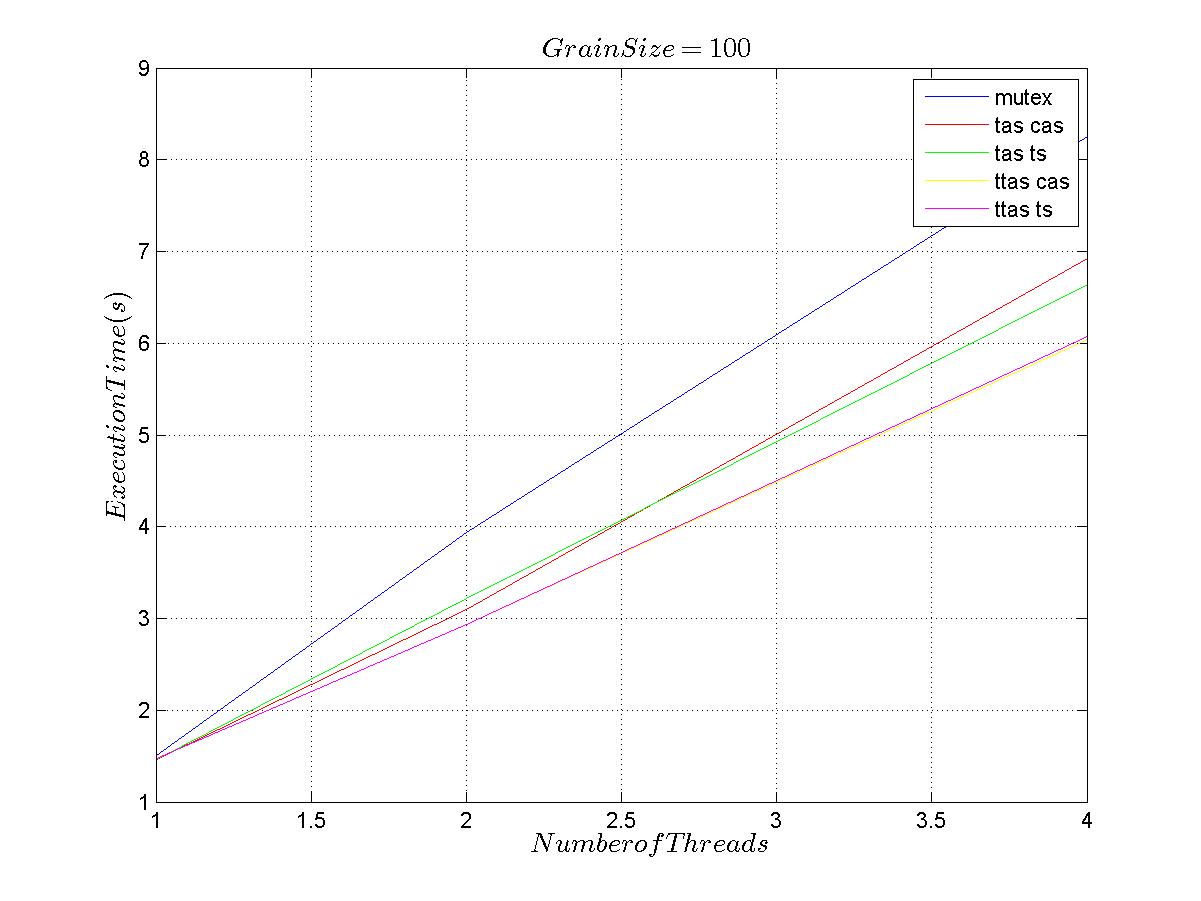
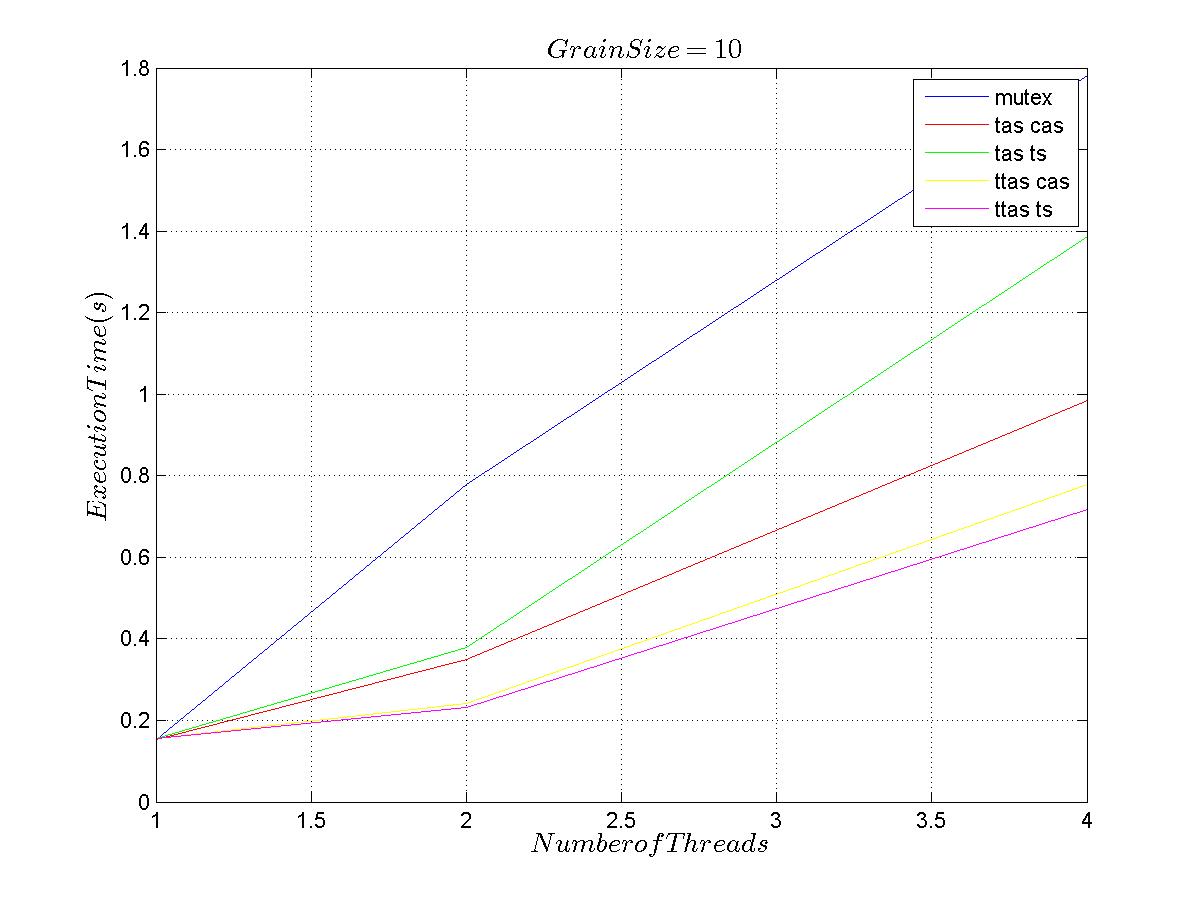
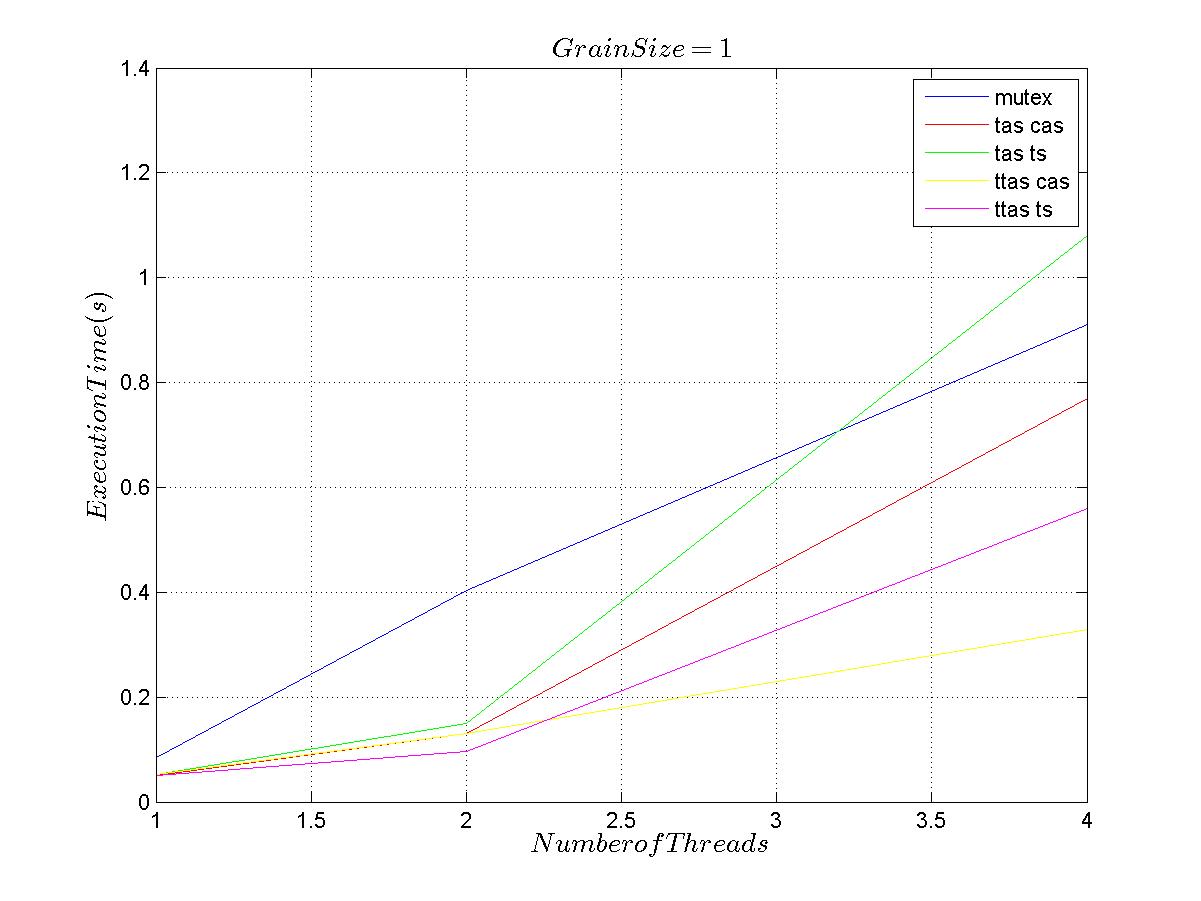
Κάτι άλλο που παρατηρείται σε αυτά τα διαγράμματα είναι ότι για μικρό αριθμό νημάτων, δεν υπάρχει μεγάλη διαφοροποίηση απόδοσης ανάμεσα στους TAS και τους TTAS, κάτι που είναι λογικό με βάση αυτά που εξηγήσαμε νωρίτερα για την λειτουργία των δύο μηχανισμών. Όσο τα νήματα γίνονται περισσότερα, τόσο περισσότερα γίνονται και τα νήματα που θα είναι κάθε φορά εκτός κρίσιμης περιοχής και άρα θα προκαλούν καθυστερήσεις στους TAS για λόγους τους οποίους εξηγήσαμε. Ενώ στους TTAS αυτές οι καθυστερήσεις θα είναι σαφώς μικρότερες. Παρόλ’ αυτά ο συγκεκριμένος κώδικας που προσομοιώνουμε δεν αποδίδει καλύτερα σε περισσότερους πυρήνες, και άρα γι’ αυτό παρατηρούμε αύξηση του χρόνου εκτέλεσης με την αύξηση των νημάτων. Απλά με βάση αυτά που είπαμε προηγουμένως ο χρόνος στους TTAS θα αυξηθεί λιγότερο (πάλι όμως θα αυξηθεί).

Τέλος η αύξηση του grain size επιφέρει και συνολική αύξηση στον αριθμό των απαιτούμενων κύκλων , αλλά αυτό φαίνεται να συμβαίνει με ομοειδή τρόπο για όλα τα κλειδώματα.

**3.1.3 Κατανάλωση ενέργειας**Σ’ αυτό το σκέλος της άσκησης θα καταγράψουμε την διακύμανση της ενεργειακής κατανάλωσης για τις ίδιες παραμέτρους που λάβαμε υπόψη μας και προηγουμένως στην περίπτωση της απόδοσης.  
  


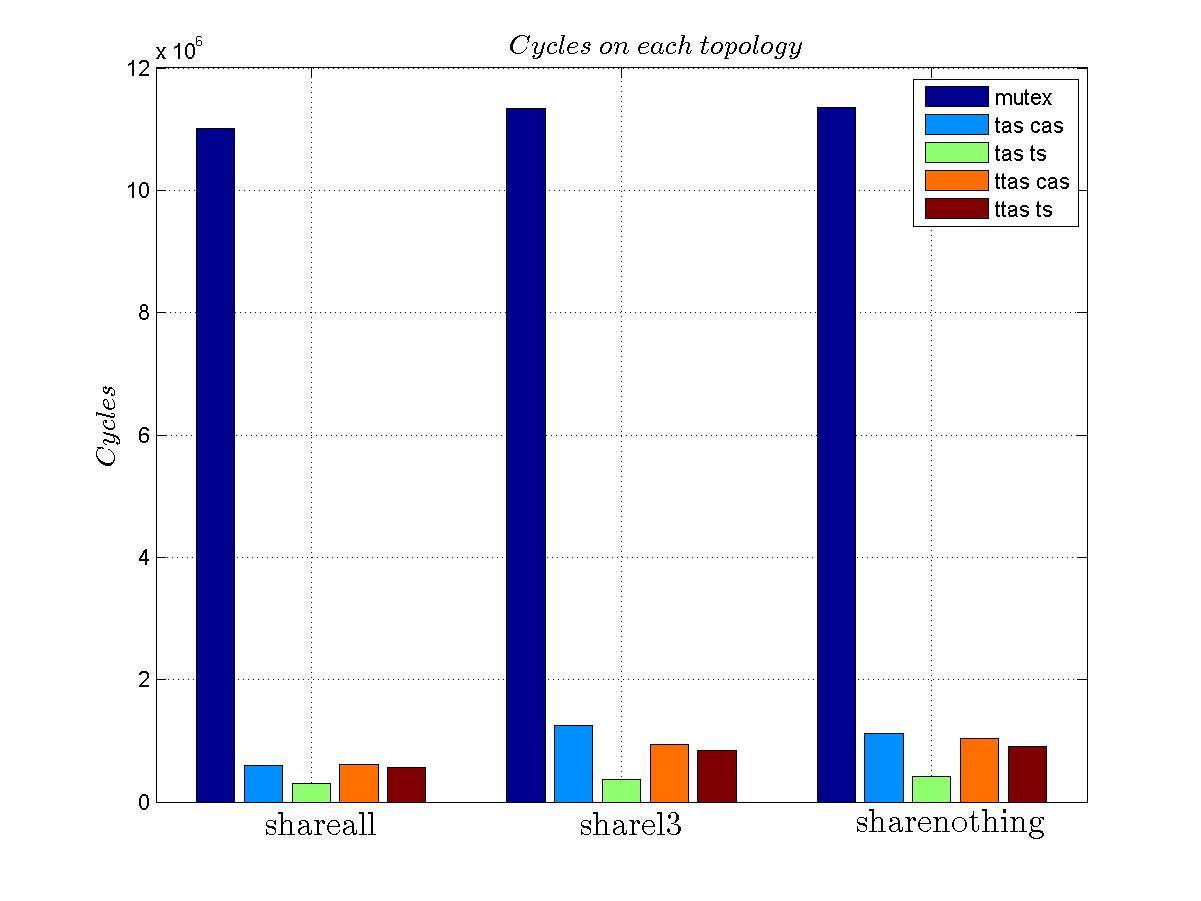
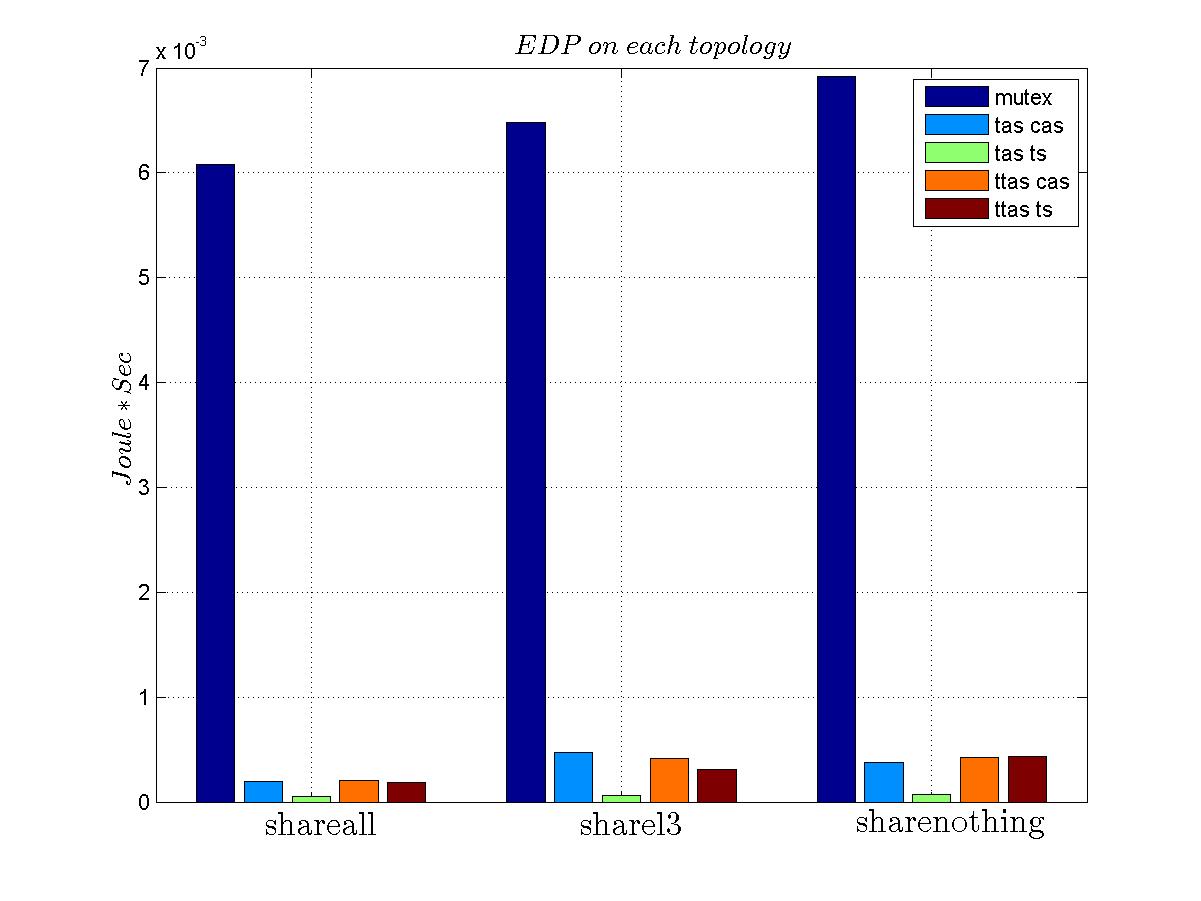
Από την πλευρά της ενέργειας παρατηρούμε ότι λιγότερο αποδοτικός είναι ο μηχανισμός TTAS γιατί το busy-wait που κάνει με την πρόσθετη συνθήκη απασχολεί αρκετά περισσότερο τον επεξεργαστή με αποτέλεσμα να καταναλώνει περισσότερη ενέργεια. Αντίθετα ο πιο αποδοτικός μηχανισμός όσο αφορά την ενέργεια είναι ο MUTEX, εκτιμάμε όμως πως αν συνεκτιμηθεί και ο χρόνος στην παραπάνω γραφική (Joule \* sec ) το MUTEX θα εμφανίζει υψηλότερες τιμές.

**3.1.4 Εκτέλεση σε πραγματικό Σύστημα**Υλοποιήσαμε τις ίδιες μετρήσεις και γραφικές με το πρώτο ερώτημα αλλά πλέον όλα αυτά συνέβησαν σε πραγματικό σύστημα. Το σύστημα μας διέθετε τέσσερις φυσικούς πυρήνες (χωρίς hyper threading).



Πλέον έχουμε τρέξει τον ίδιο κώδικα σε πραγματικό μηχάνημα, σε αντίθεση με το προηγούμενο ερώτημα, όπου τον είχαμε προσομοιώσει με τον sniper. Εδώ παρατηρούμε ότι έχουμε έντονες διαφορές σε σχέση με τις προσομοιώσεις. Καταρχάς οι χρόνοι εκτέλεσης αυξάνουν συνολικά όσο αυξάνει το grain size, κάτι που είναι λογικό και δεν συνέβαινε στον ίδιο βαθμό στις προσομοιώσεις. Ακόμα βλέπουμε πως με την αύξηση του αριθμού των νημάτων αυξάνει απότομα ο αριθμός των κύκλων εκτέλεσης για όλους τους μηχανισμούς κυκλώματος και όχι μόνο για το Mutex όπως συνέβαινε στο πρώτο ερώτημα.  
  
 Επίσης παρατηρούμε ότι ο Μutex και εδώ είναι ο χειρότερος μηχανισμός σε κάθε περίπτωση, ενώ δεν υπάρχει καλύτερος αφού εναλλάσσονται διάφοροι μηχανισμοί ανάλογα το Grain size, αρκετά διαφορετικά αποτελέσματα δηλαδή από αυτά που είχαμε από τον sniper. Αυτό ίσως μπορεί να εξηγηθεί από την αρχιτεκτονική υλοποίηση του πραγματικού επεξεργαστή πάνω στον οποίον τρέξαμε τον κώδικα, για λόγους παρόμοιους με αυτούς που εξετάζονται παρακάτω, στο επόμενο ερώτημα.

**3.2 Διαμοιρασμός Πόρων**Στο τελευταίο σκέλος αυτής της άσκησης μελετήσαμε πως επηρεάζει η διαφορετική διάταξη της μνήμης τόσο την απόδοση όσο και την κατανάλωση ενέργειας για τους διάφορους μηχανισμούς κυκλωμάτων. Συγκεκριμένα προσομοιώσαμε τρεις διαφορετικές διατάξεις, στην πρώτη οι διάφοροι πυρήνες μοιράζονται τόσο το δεύτερο όσο και το τρίτο επίπεδο κρυφής μνήμης, στη δεύτερη μόνο το τρίτο επίπεδο, ενώ στην τελευταία διάταξη κανένα επίπεδο κρυφής μνήμης δεν είναι κοινό.

****  
****

Παρατηρούμε καταρχάς ότι η shareall τοπολογία νημάτων είναι πιο γρήγορη σε κάθε περίπτωση (λιγότερα cycles) . Αυτό είναι λογικό καθώς στις υπόλοιπες περιπτώσεις απαιτείται η μεταφορά του block του Lock μεταξύ των cache των επεξεργαστών.  
  
 Στην share-nothing θα πρέπει να επικοινωνήσουν μέσω της κύριας μνήμης, κάτι που είναι πολύ αργό σαν διαδικασία, τάξης μεγέθους πάνω από την επικοινωνία εντός κάποιου επιπέδου κρυφής μνήμης. Παρατηρούμε επίσης ότι οι TTAS σε σχέση με τους αντίστοιχους TAS είναι πιο αργοί, ενώ οι TS μηχανισμοί είναι πιο γρήγοροι από τους CAS.   
  
Τέλος, όσο αφορά την ενέργεια (ΕDP = Joule \* sec) την μεγαλύτερη κατανάλωση έχει το share-nothing και την λιγότερη το share-all, κάτι που είναι πολύ λογικό βάση της εξήγησης που δώσαμε και παραπάνω για την απόδοση. Τέλος o μηχανισμός MUTEX εμφανίζει πολύ μεγαλύτερη τιμή σε σχέση με όλους τους υπόλοιπους, κάτι που οφείλεται κυρίως στον παράγοντα του χρόνου εκτέλεσης που υπεισέρχεται σ’ αυτή τη γραφική .

**Κώδικας**

Τέλος συμπεριλαμβάνουμε και τον κώδικα που συμπληρώσαμε στο αρχείο lock.h :

#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));

}

static inline void spin\_lock\_ttas\_cas(spinlock\_t \*spin\_var)

{

do{

while(\*spin\_var == LOCKED);

}while(\_\_sync\_val\_compare\_and\_swap(spin\_var, UNLOCKED, LOCKED));

}

static inline void spin\_lock\_tas\_ts(spinlock\_t \*spin\_var)

{

while(\_\_sync\_lock\_test\_and\_set(spin\_var, LOCKED));

}

static inline void spin\_lock\_ttas\_ts(spinlock\_t \*spin\_var)

{

do{

while(\*spin\_var == LOCKED);

}while(\_\_sync\_lock\_test\_and\_set(spin\_var,LOCKED));

}

static inline void spin\_unlock(spinlock\_t \*spin\_var)

{

\_\_sync\_lock\_release(spin\_var);

}

#endif