ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

(2020-2021)

3η Σειρά Ασκήσεων

Ονοματεπώνυμο:

- Χρήστος Τσούφης

Αριθμός Μητρώου:

- 03117176

Στοιχεία Επικοινωνίας:

- el17176@mail.ntua.gr

1. Εισαγωγή

Στόχος της άσκησης αυτής είναι η εξοικείωση με τους μηχανισμούς συγχρονισμού και τα πρωτόκολλα συνάφειας κρυφής μνήμης (cache coherence protocols) σε σύγχρονες πολυπύρηνες αρχιτεκτονικές. Για το σκοπό αυτό σας δίνεται ένας πολυνηματικός κώδικας με τον οποίο θα υλοποιήσετε διάφορους μηχανισμούς συγχρονισμού, τους οποίους στη συνέχεια θα αξιολογήσετε σε ένα πολυπύρηνο προσομοιώμενο σύστημα με τη βοήθεια του προσομοιωτή "Sniper Multicore Simulator", ο οποίος αξιοποιεί το εργαλείο "PIN" που χρησιμοποιήσατε στις προηγούμενες ασκήσεις. Λεπτομέρειες και υλικό (παρουσιάσεις, κώδικα, manual κτλ) σχετικά με τον προσομοιωτή μπορείτε να βρείτε εδώ:

http://snipersim.org/w/The_Sniper_Multi-Core_Simulator

2. Εγκατάσταση και χρήση του προσομοιωτή

Βλ. οδηγίες στην εκφώνηση.

3. Υλοποίηση μηχανισμών συγχρονισμού

Καλείστε να υλοποιήσετε τους μηχανισμούς κλειδώματος Test-and-Set (TAS) και Test-and-Test-and-Set (TTAS), όπως τους διδαχτήκατε στις αντίστοιχες διαφάνειες του μαθήματος.

Συγκεκριμένα, στο αρχείο locks_scalability.c δίνεται έτοιμος κώδικας C ο οποίος χρησιμοποιεί τη βιβλιοθήκη Pthreads (Posix Threads) για τη δημιουργία και διαχείριση των νημάτων λογισμικού. Ο κώδικας δημιουργεί έναν αριθμό από νήματα, καθένα εκ των οποίων εκτελεί μια κρίσιμη περιοχή για ένα συγκεκριμένο αριθμό επαναλήψεων. Για όλα τα νήματα, η είσοδος στην κρίσιμη περιοχή ελέγχεται από μία κοινή μεταβλητή κλειδιού.

Πριν την είσοδο στην περιοχή, το κάθε νήμα εκτελεί την κατάλληλη ρουτίνα για την απόκτηση του κλειδιού (lock aqcuire), ώστε να εισέλθει σε αυτήν κατ' αποκλειστικότητα. Μέσα στην κρίσιμη περιοχή, το κάθε νήμα εκτελεί έναν dummy cpu-intensive υπολογισμό. Κατά την έξοδο από αυτήν εκτελεί την κατάλληλη ρουτίνα για την απελευθέρωση του κλειδιού (lock release). Δίνοντας τα κατάλληλα flags κατά τη μεταγλώττιση, μπορείτε να παράξετε κώδικα που χρησιμοποιεί κλήσεις σε κλειδώματα TAS, TTAS ή Pthread mutex (MUTEX). Ο τελευταίος από τους μηχανισμούς είναι ήδη υλοποιημένος στην βιβλιοθήκη Pthreads. Εσείς καλείστε να υλοποιήσετε τις ρουτίνες απόκτησης και απελευθέρωσης κλειδιού για τους μηχανισμούς TAS και TTAS.

3.1 Υλοποίηση κλειδωμάτων TAS και TTAS

Πιο συγκεκριμένα, οι ρουτίνες που θα πρέπει να υλοποιήσετε είναι οι spin_lock_tas_cas, spin_lock_tas_ts, spin_lock_ttas_cas και spin_lock_ttas_cas. Οι ορισμοί τους δίνονται, ημιτελείς, στο αρχείο lock.h και εσείς θα πρέπει υλοποιήσετε το κυρίως σώμα τους. Στο ίδιο αρχείο δίνεται ο ορισμός του τύπου για τη μεταβλητή κλειδιού (spinlock_t), ο ορισμός των σταθερών που σηματοδοτούν αν η κρίσιμη περιοχή είναι κλειδωμένη ή ελεύθερη (LOCKED, UNLOCKED), καθώς και ο ορισμός της συνάρτησης αρχικοποίησης της μεταβλητής κλειδιού η οποία καλείται πριν τη δημιουργία των νημάτων.

3.2 Περιγραφή του προγράμματος

Το πρόγραμμα locks_scalability.c δέχεται τα εξής ορίσματα γραμμής εντολών:

- *nthreads: ο αριθμός των threads που θα δημιουργηθούν*
- *iterations: ο αριθμός των επαναλήψεων που θα εκτελεστεί η κρίσιμη περιοχή από κάθε νήμα*
- *grain_size: καθορίζει τον όγκο των dummy, cpu-intensive υπολογισμών, και κατ' επέκταση το μέγεθος της κρίσιμης περιοχής*

Στον κώδικα του προγράμματος υπάρχουν κατάλληλα compile-time directives τα οποία ορίζουν τη συμπερίληψη ή όχι κατά το χρόνο μεταγλώττισης ενός τμήματος του κώδικα. Με αυτόν τον τρόπο μπορούμε να ενσωματώσουμε σε ένα μόνο αρχείο διαφορετικές εκδόσεις του προγράμματος. Τα directives που χρησιμοποιούνται, και η σημασία τους, είναι τα εξής:

- *SNIPER | REAL: ενεργοποιούν τα κατάλληλα τμήματα κώδικα για εκτέλεση του προγράμματος στον Sniper ή σε πραγματικό σύστημα, αντίστοιχα*
- *TAS_CAS | TAS_TS | TTAS_CAS | TTAS_TS | MUTEX: ενεργοποιούν τις κλήσεις στους μηχανισμούς συγχρονισμού TAS, TTAS και Pthread Mutex, αντίστοιχα*
- *DEBUG: ενεργοποιεί την εκτύπωση debugging μηνυμάτων*

Σε αρχικό στάδιο, στη συνάρτηση main γίνονται οι απαραίτητες αρχικοποιήσεις, όπως η δέσμευση δομών και η αρχικοποίηση της μεταβλητής κλειδιού. Τα δεδομένα εισόδου που προορίζονται για κάθε νήμα ξεχωριστά αποθηκεύονται στη δομή targs_t. Συγκεκριμένα, η δομή περιλαμβάνει το πεδίο my_id που εκφράζει την ταυτότητα ενός νήματος, και αρχικοποιείται σε μια τιμή ξεχωριστή για κάθε νήμα (0 έως nthreads-1).

Σε δεύτερη φάση, ακολουθεί η δημιουργία των νημάτων με χρήση της pthread_create. Κάθε νήμα που δημιουργείται αρχίζει και εκτελεί τη συνάρτηση thread_fn. Μέσα σε αυτήν, το νήμα αρχικά θέτει το cpu affinity του, ορίζει δηλαδή σε ένα bitmask (mask) το σύνολο των cpus του συστήματος ή του simulator όπου μπορεί να εκτελεστεί. Για τους σκοπούς της άσκησης θέλουμε μια "1-1" απεικόνιση ανάμεσα στα νήματα του προγράμματος και τις διαθέσιμες cpus, επομένως κάθε νήμα θέτει το affinity του σε μία και μόνο cpu, αυτή που έχει το ίδιο id με το my_id του. Δηλαδή, το νήμα 0 θα εκτελεστεί στη cpu 0, το νήμα 1 στη cpu 1, κ.ο.κ..

Στη συνέχεια, τα νήματα που έχουν δημιουργηθεί συγχρονίζονται (pthread_barrier_wait) ώστε ένα από αυτά (το πρώτο) να ενεργοποιήσει την λεπτομερή προσομοίωση και την καταγραφή των στατιστικών στην περιοχή που μας ενδιαφέρει (SimRoiStart()), αν η εκτέλεση γίνεται στον Sniper, ή να αρχίσει τη μέτρηση του χρόνου εκτέλεσης, αν η εκτέλεση γίνεται σε πραγματικό σύστημα (gettimeofday). Ακολουθεί ο βρόχος εντός του οποίου εκτελείται η κρίσιμη περιοχή, προστατευόμενη από την εκάστοτε υλοποίηση κλειδώματος. Ομοίως, αφού ολοκληρωθεί η εκτέλεση τα νήματα συγχρονίζονται και πάλι ώστε ένα από αυτά να απενεργοποιήσει τη λεπτομερή προσομοίωση ή την μέτρηση του χρόνου.

3.3 Μεταγλώττιση προγράμματος για προσομοίωση στον Sniper

Στο tarball που σας δίνεται υπάρχουν τα αρχεία locks_scalability.c, lock.h καθώς και το Makefile. Το τελευταίο μπορεί να παράξει είτε κώδικα για προσομοίωση στον Sniper ή για εκτέλεση σε πραγματικό μηχάνημα αναλόγως με την τιμή του IMPLFLAG. Μπορείτε να θέσετε το IMPLFLAG σε –DSNIPER ή –DREAL. Για την παραγωγή κώδικα που κάνει χρήση των μηχανισμών TAS_CAS, TAS_TS, TTAS_CAS, TTAS_TS ή Pthread Mutex, αρκεί να θέσετε στο Makefile το LFLAG σε –DTAS_CAS, -DTAS_TS, -DTTAS_CAS, -DTTAS_TS ή -DMUTEX, αντίστοιχα. Τέλος, αφού έχετε θέσει το SNIPER_BASE_DIR ώστε να δείχνει στο path στο οποίο έχετε μεταγλωττίσει τον sniper, δίνοντας την εντολή make στο τερματικό θα παραχθεί το εκτελέσιμο με το όνομα locks.

4. Αξιολόγηση επίδοσης

Σε αυτό το μέρος καλείστε να αξιολογήσετε τις επιδόσεις των υλοποιήσεών σας, τόσο ως προς την κλιμακωσιμότητά τους, όσο και ως προς την τοπολογία των νημάτων.

4.1 Σύγκριση υλοποιήσεων

Με τον όρο κλιμακωσιμότητα (scalability) εννοούμε πόσο καλά αποδίδει ένα παράλληλο πρόγραμμα καθώς αυξάνεται ο αριθμός των νημάτων από τα οποία αποτελείται. Ακόμα και αν ένα παράλληλο πρόγραμμα είναι σχεδιασμένο για να κλιμακώνει ιδανικά (π.χ., ο χρόνος εκτέλεσης με N νήματα να ισούται με 1/N του χρόνου με 1 νήμα), υπάρχουν διάφοροι εξωγενείς παράγοντες που μπορούν να περιορίσουν την κλιμακωσιμότητά του πολύ κάτω του ιδανικού. Τέτοιοι είναι για παράδειγμα το overhead που σχετίζεται με το πρωτόκολλο συνάφειας, το περιορισμένο bandwidth πρόσβασης στην κύρια μνήμη, κ.ο.κ.. Σε τέτοιες περιπτώσεις, το κόστος μιας λειτουργίας ενός νήματος (π.χ. προσπέλαση μνήμης) είναι πολύ μεγαλύτερο συνήθως όταν στην εκτέλεση συμμετέχουν πολλά νήματα, παρά όταν συμμετέχουν λίγα.

Ζητούμενο του ερωτήματος αυτού είναι η αξιολόγηση και σύγκριση της κλιμακωσιμότητας των μηχανισμών TAS_CAS, TAS_TS, TTAS_CAS, TTAS_TS και Pthread mutex για αριθμούς νημάτων 1, 2, 4, 8, 16. Για το λόγο αυτό θα προσομοιώσετε τα πολυπύρηνα συστήματα αντίστοιχου πλήθους πυρήνων που απεικονίζονται στο επόμενο σχήμα και υλοποιούν διαφορετικές πολιτικές διαμοιρασμού των caches. Κάθε τέτοιο σύστημα χρησιμοποιεί το MSI πρωτόκολλο συνάφειας κρυφής μνήμης.

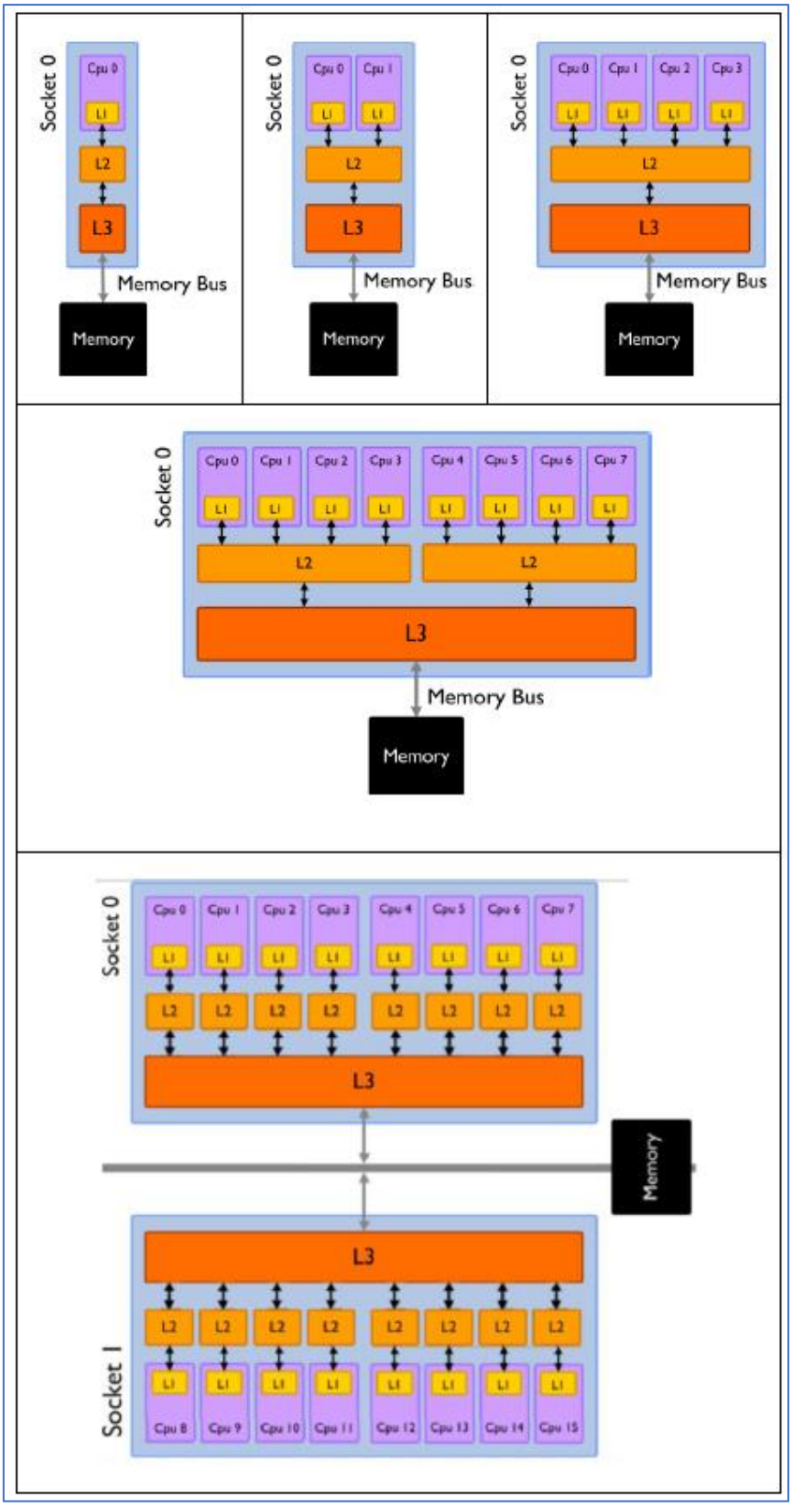
Καλείστε λοιπόν να εκτελέσετε προσομοιώσεις του προγράμματος για όλους τους ακόλουθους συνδυασμούς:

- *εκδόσεις προγράμματος: TAS_CAS, TAS_TS, TTAS_CAS, TTAS_TS, MUTEX*
- *iterations: 1000*
- *nthreads: 1, 2, 4, 8, 16 (σε σύστημα με ισάριθμους πυρήνες)*
- *grain_size: 1, 10, 100*

Για να εκτελέσετε μια προσομοίωση, τρέχετε τον Sniper με όρισμα το επιθυμητό εκτελέσιμο.

Το configuration script που θα χρησιμοποιήσετε είναι το ask3.cfg που δίνεται στο tarball.

ΠΡΟΣΟΧΗ: ο αριθμός των νημάτων που δίνετε σαν 1ο όρισμα στο εκτελέσιμο θα πρέπει να ταυτίζεται με τον αριθμό που δίνετε στο όρισμα -n του προσομοιωτή, ώστε το σύστημα που δημιουργεί ο Sniper να έχει ισάριθμους πυρήνες.

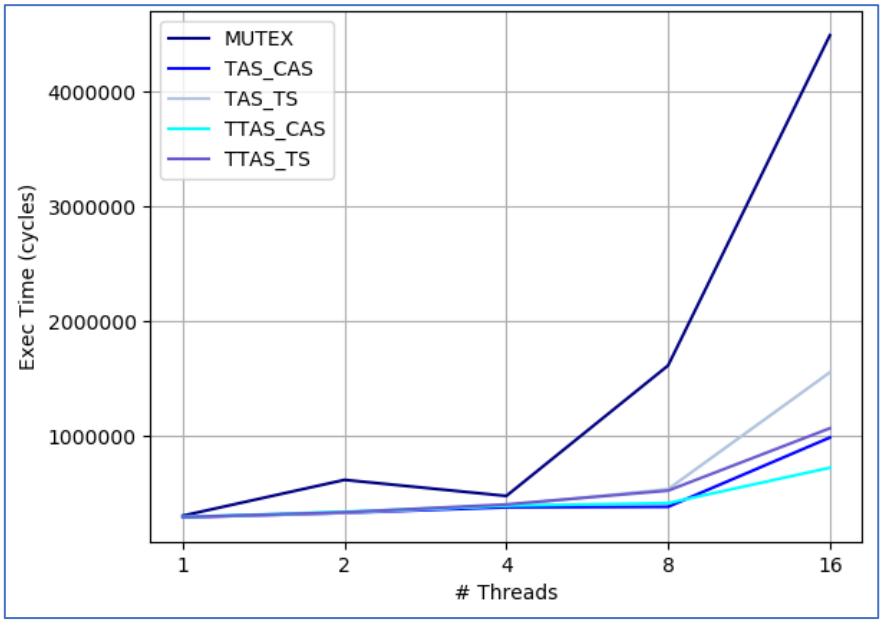

Socket 0
Cpu 0
L1
L2
L3
Memory Bus
Memory
Socket 0
Cpu 0
Cpu 1
L1
L2
L3
Memory Bus
Memory
Socket 0
Cpu 0
Cpu 1
Cpu 2
Cpu 3
L1
L2
L3
Memory Bus
Memory
Socket 0
Cpu 0
Cpu 1
Cpu 2
Cpu 3
Cpu 4
Cpu 5
Cpu 6
Cpu 7
L1
L2
L2
L3
Memory Bus
Memory
Socket 0
Cpu 0
Cpu 1
Cpu 2
Cpu 3
Cpu 4
Cpu 5
Cpu 6
Cpu 7
L1
L2
L3
Memory
L3
L2
L1
Cpu 8
Cpu 9
Cpu 10
Cpu 11
Cpu 12
Cpu 13
Cpu 14
Cpu 15
Socket 1

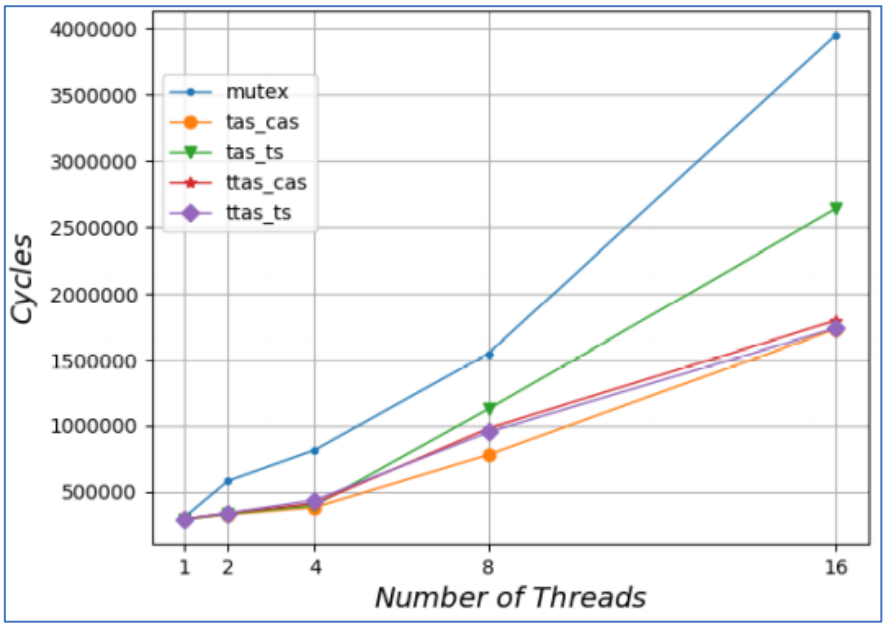
4.1.1. *Για κάθε grain size, δώστε το διάγραμμα της κλιμάκωσης του συνολικού χρόνου εκτέλεσης της περιοχής ενδιαφέροντος σε σχέση με τον αριθμό των νημάτων. Συγκεκριμένα, στον x-άξονα θα πρέπει να έχετε τον αριθμό των νημάτων και στον y-άξονα τον χρόνο εκτέλεσης σε κύκλους. Στο ίδιο διάγραμμα θα πρέπει να συμπεριλάβετε τα αποτελέσματα και για τις 5 εκδόσεις.*

Παρακάτω φαίνονται τα διαγράμματα για κάθε grain size (1, 10, 100 αντίστοιχα) του χρόνου εκτέλεσης της περιοχής ενδιαφέροντος σε σχέση με τον αριθμό των νημάτων.

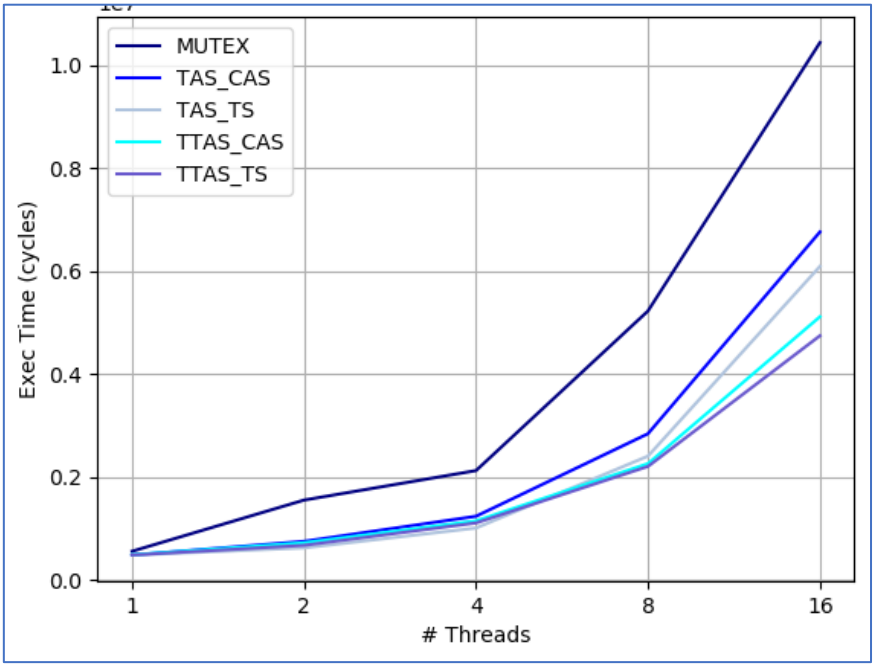
*Τα διαγράμματα με διαφορετικό χρώμα που φαίνονται παρακάτω, έτρεξαν σε διαφορετικό PC.

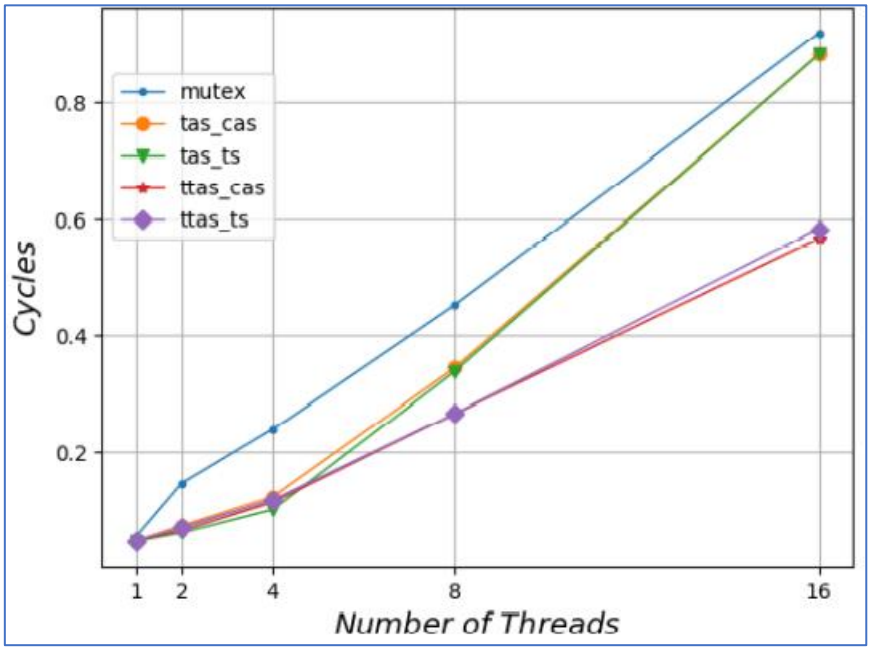
Grain size: 1



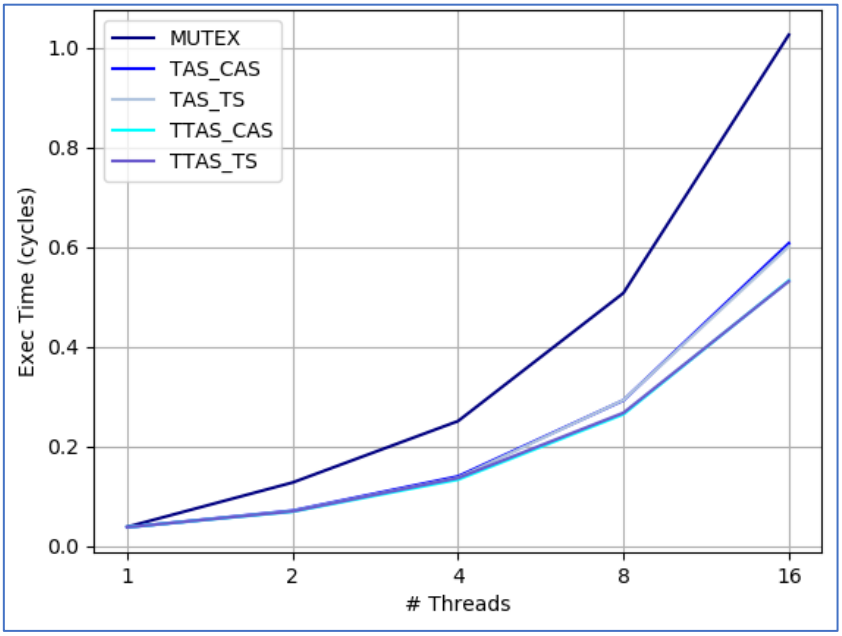


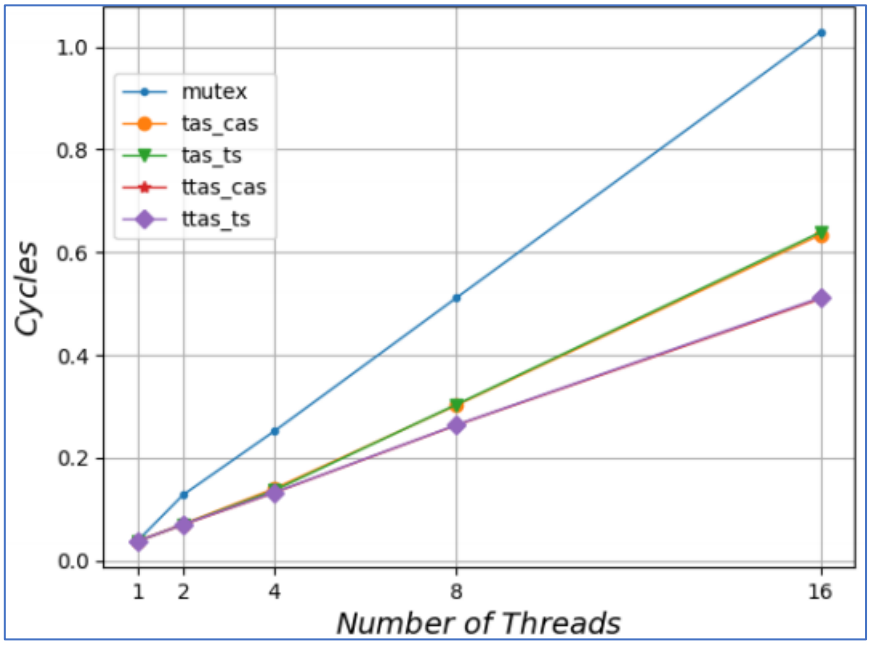
Grain size: 10





Grain size: 100





4.1.2. *Τι συμπεραίνετε για την κλιμάκωση του χρόνου εκτέλεσης σε σχέση με τη φύση της εκάστοτε υλοποίησης; Τι συμπεραίνετε για την κλιμάκωση του χρόνου εκτέλεσης σε σχέση με το grain size; Δικαιολογήστε τις απαντήσεις σας.*

Σχολιασμός & Συμπεράσματα:

Οι μηχανισμοί TTAS_CAS και TTAS_TS υλοποιούν πρωτόκολλο αποκλεισμού Test-and-Test-and-Set ενώ οι μηχανισμοί TAS_CAS και TAS_TS υλοποιούν πρωτόκολλο αποκλεισμού Test-and-Set. Σύμφωνα με το Test-and-Test-and-Set μια διεργασία δεν γράφει απευθείας στο lock μέσω Test-and-Set αλλά παρακολουθεί την τιμή του και επιχειρεί να γράψει σε αυτό μόνο όταν φαίνεται άδειο σε αντίθεση με το απλό Test-and-Set. Οπότε, δημιουργείται λιγότερη άχρηστη κίνηση στο δίαυλο και βελτιώνεται η επίδοση γεγονός που επιβεβαιώνεται από τις γραφικές παραστάσεις αφού οι επιδόσεις των TTAS είναι σταθερά καλύτερες από αυτές των TAS.

Ακολούθως, παρατίθεται ένας πίνακας στον οποίο φαίνονται συνοπτικά ποια πρωτόκολλα χρησιμοποιούνται από τους αντίστοιχους μηχανισμούς:

TAS_CAS, TAS_TS	TTAS_CAS, TTAS_TS
➢ Διαφορετικός τρόπος υλοποίησης του Test-and-Set ➢ Γράφουν πάνω στο lock ➢ Κοντινές επιδόσεις	➢ Υλοποίηση πρωτόκολλου Test-and-Test-and-Set ➢ ΔΕΝ γράφουν πάνω στο lock ➢ Καλύτερες επιδόσεις

Παρατηρείται ότι:

- Ο μηχανισμός mutex έχει σταθερά χειρότερη επίδοση και χειρότερη κλιμάκωση από όλους τους υπόλοιπους μηχανισμούς.
- Γενικά η επίδοση χειροτερεύει γραμμικά με την αύξηση των threads (εκ πρώτης όψεως φαίνεται εκθετική αλλά ο άξονας των x δεν αυξάνεται γραμμικά αλλά εκθετικά).
- Η καλύτερη κλιμάκωση για τους μηχανισμούς TAS και TTAS επιτυγχάνεται για grain size ίσο με 1 (μικρή κρίσιμη περιοχή) και όσο αυξάνεται το grain size χειροτερεύει η κλιμάκωση. Αυτό φαίνεται από τις κλίσεις που έχουν οι γραμμές των γραφικών παραστάσεων (όσο πιο απότομη κλίση, τόσο χειρότερη κλιμακωσιμότητα).
- Η αύξηση του grain size (μεγέθους κρίσιμης περιοχής) επιφέρει σημαντική επιδείνωση της επίδοσης αφού οι τιμές του κατακόρυφου άξονα αυξάνονται κατά μία τάξη μεγέθους με κάθε αύξηση του grain size.
- Ο χρόνος εκτέλεσης αυξάνεται γραμμικά σχεδόν με το πλήθος των νημάτων.
- Οι γραφικές παραστάσεις κάθε μηχανισμού έχουν διαφορετική κλίση.
- όταν αυξάνεται το grain size αυξάνονται οι κύκλοι εκτέλεσης, ,ανεξάρτητα από το κλείδωμα που χρησιμοποιείται, με αποτέλεσμα να μειώνεται η κλίση, δηλαδή να βελτιώνεται η κλιμακωσιμότητα. Αυτό προκύπτει γιατί οι υπολογισμοί απαιτούν περισσότερο χρόνο, ενώ ο ανταγωνισμός του lock λιγότερο, με αποτέλεσμα να βελτιώνεται επίδοση.

* Τα τελευταία 3 σχόλια αφορούν τα πολύχρωμα διαγράμματα παραπάνω.

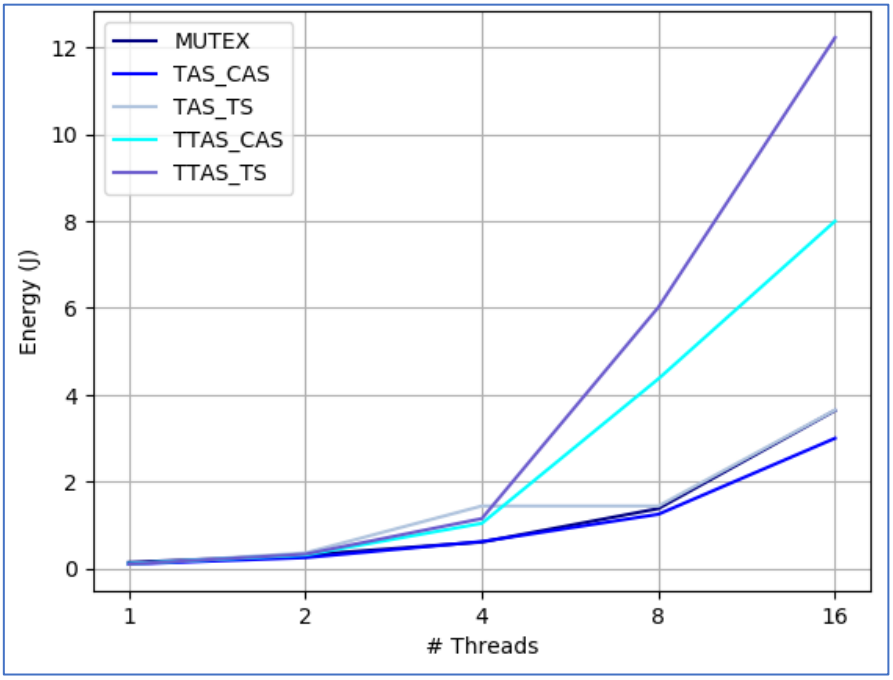
Συνεπώς, συγκρίνοντας τις υλοποιήσεις CAS (compare-and-swap) με τις TS(test-and-set) δεν παρατηρείται σταθερή συμπεριφορά επομένως δεν μπορούν να εξαχθούν ασφαλή συμπεράσματα.

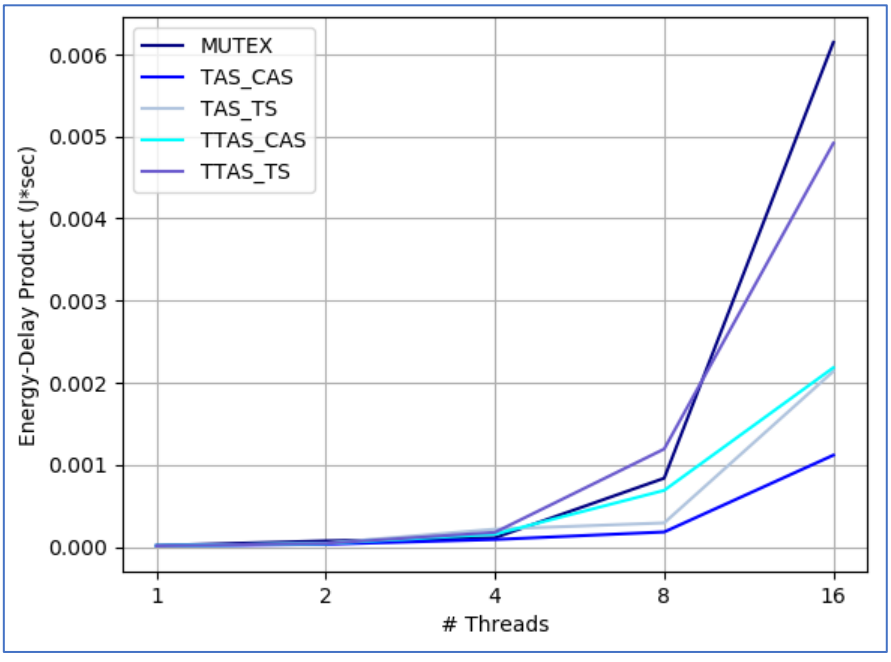
4.1.3. *Χρησιμοποιώντας όπως και στην προηγούμενη άσκηση το McPAT, συμπεριλάβετε στην ανάλυση σας εκτός από το χρόνο εκτέλεσης και την κατανάλωση ενέργειας (Energy, EDP κτλ.)*

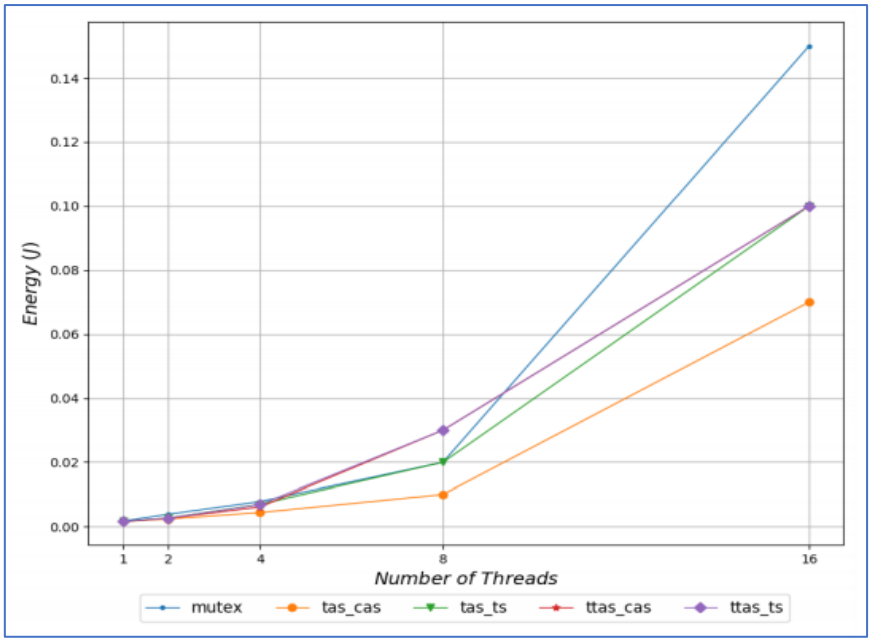
Παρακάτω φαίνονται τα διαγράμματα για κάθε grain size (1, 10, 100 αντίστοιχα) της μετρικής Ενέργειας και Energy-Delay Product σε σχέση με τον αριθμό των νημάτων.

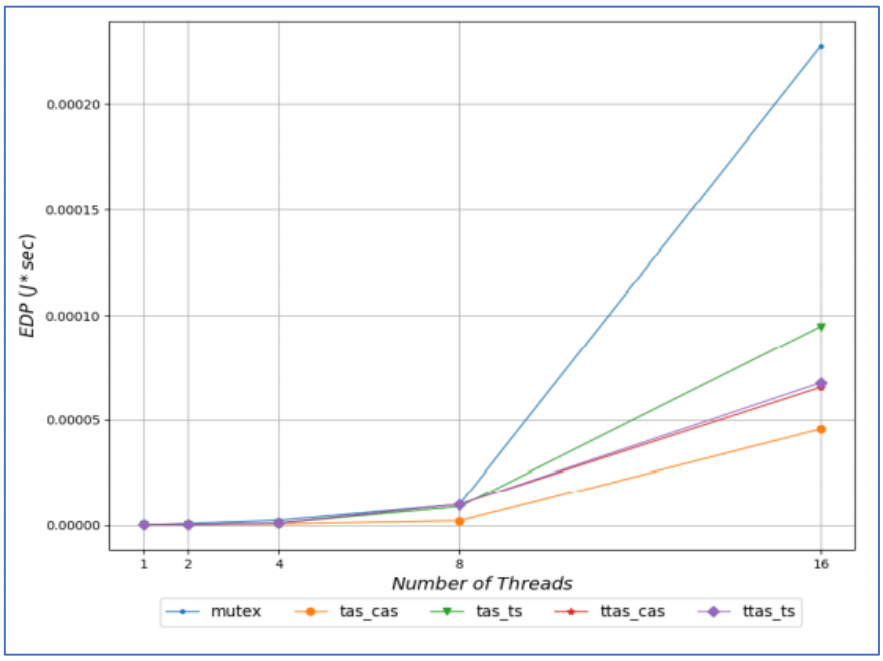
* Τα πολύχρωμα διαγράμματα που φαίνονται παρακάτω, έτρεξαν σε διαφορετικό PC.

Grain size: 1

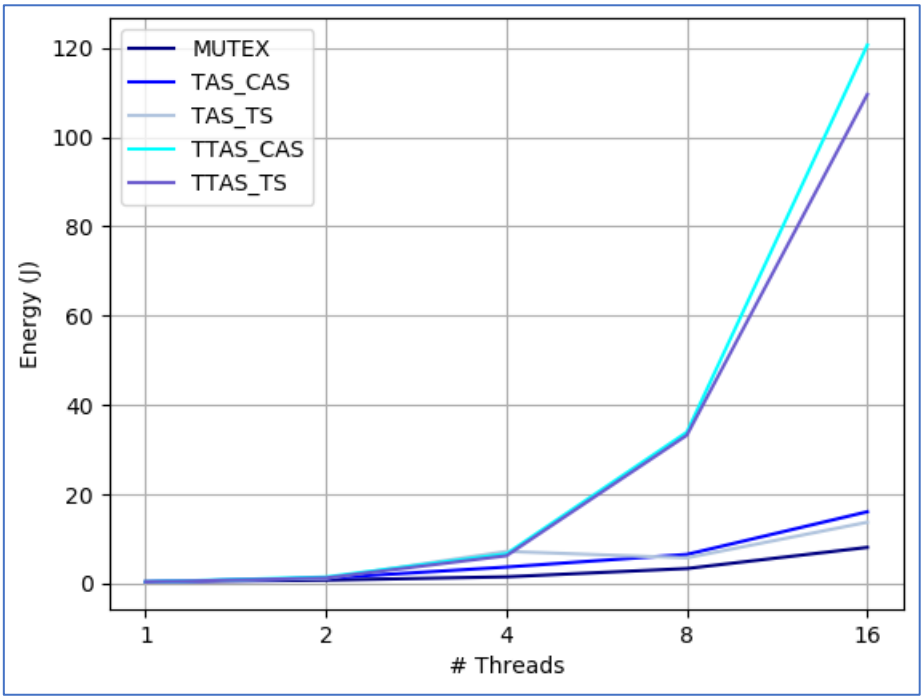


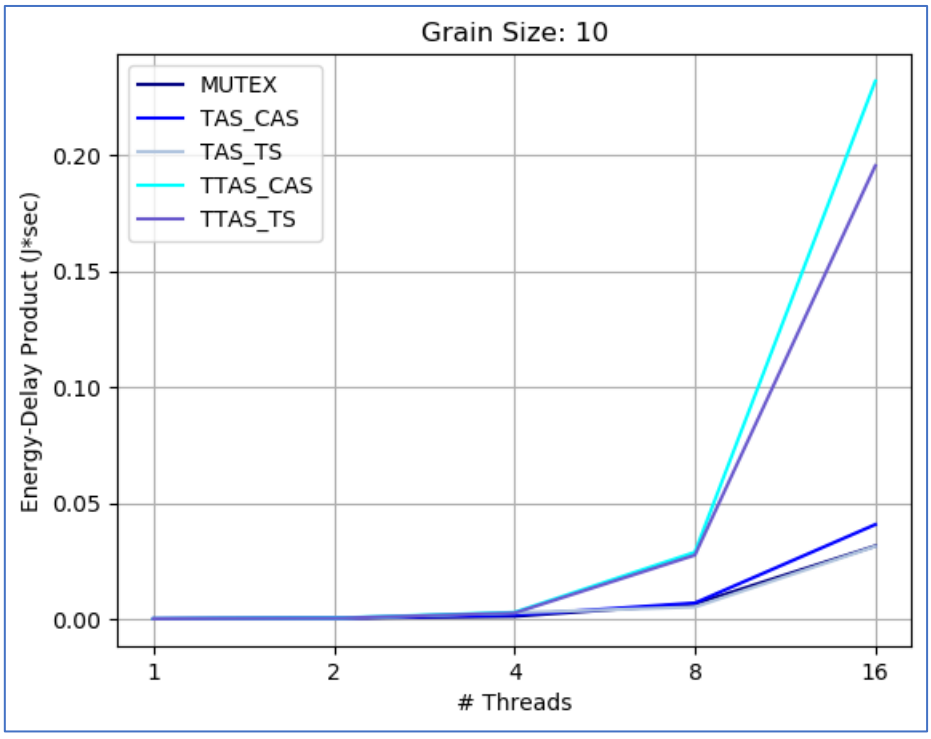


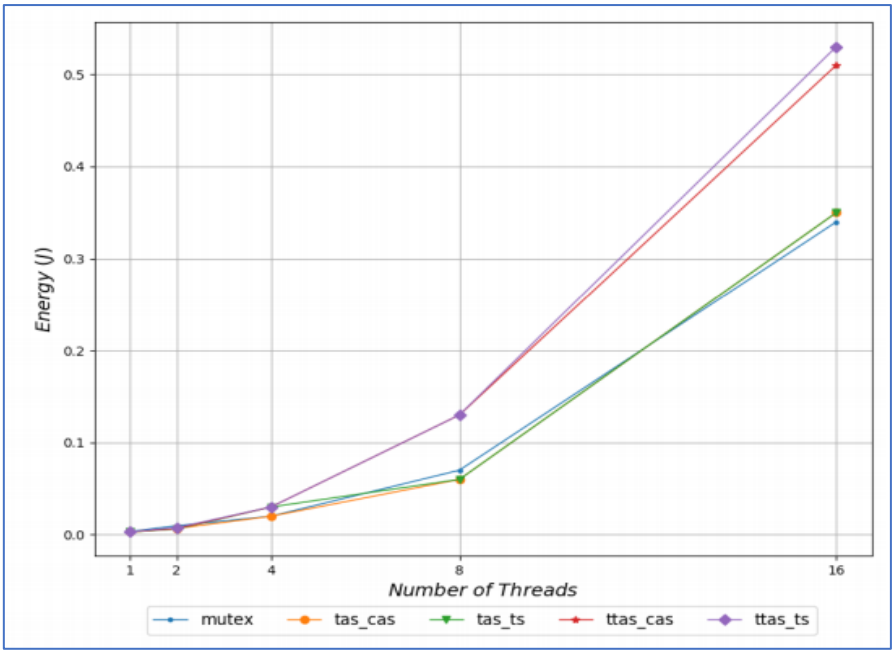

0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Energy (J)
1
2
4
8
16
Number of Threads
mutex
tas_cas
tas_ts
ttas_cas
ttas_ts

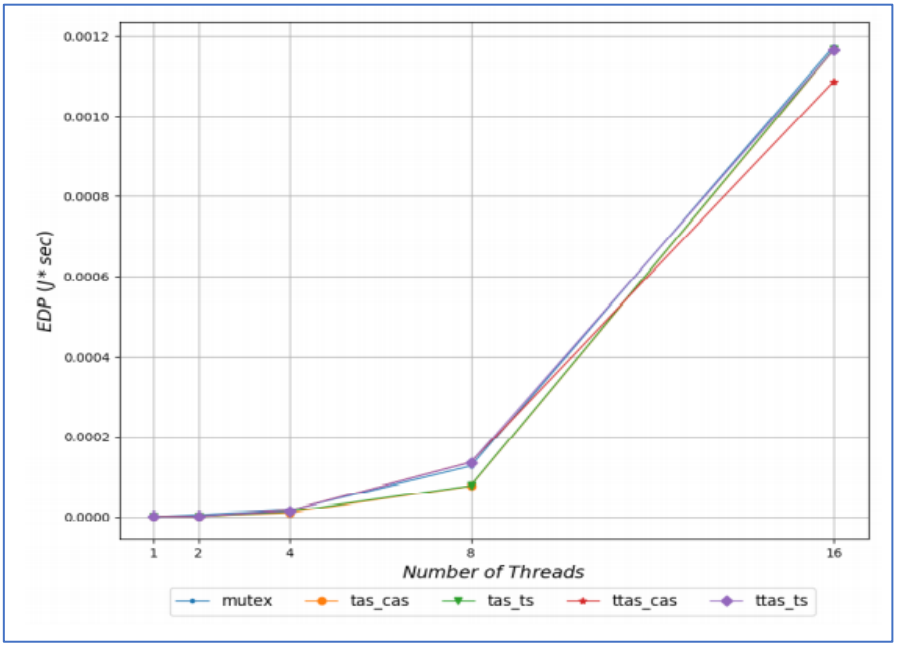

0.00020
0.00015
0.00010
0.00005
0.00000
EDP (J*sec)
1
2
4
8
16
Number of Threads
mutex
tas_cas
tas_ts
ttas_cas
ttas_ts

Grain size: 10

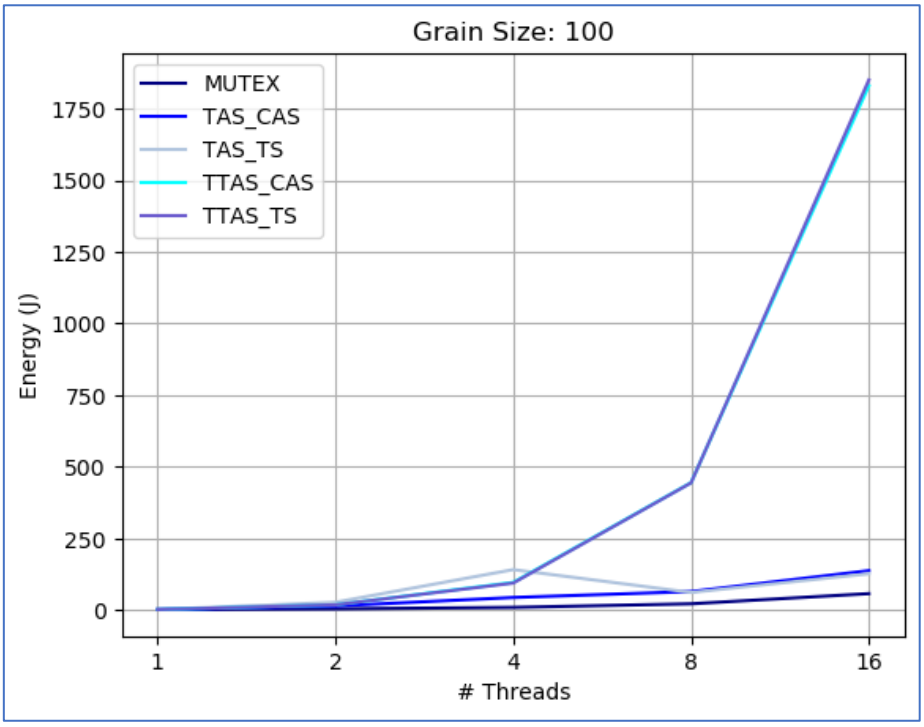


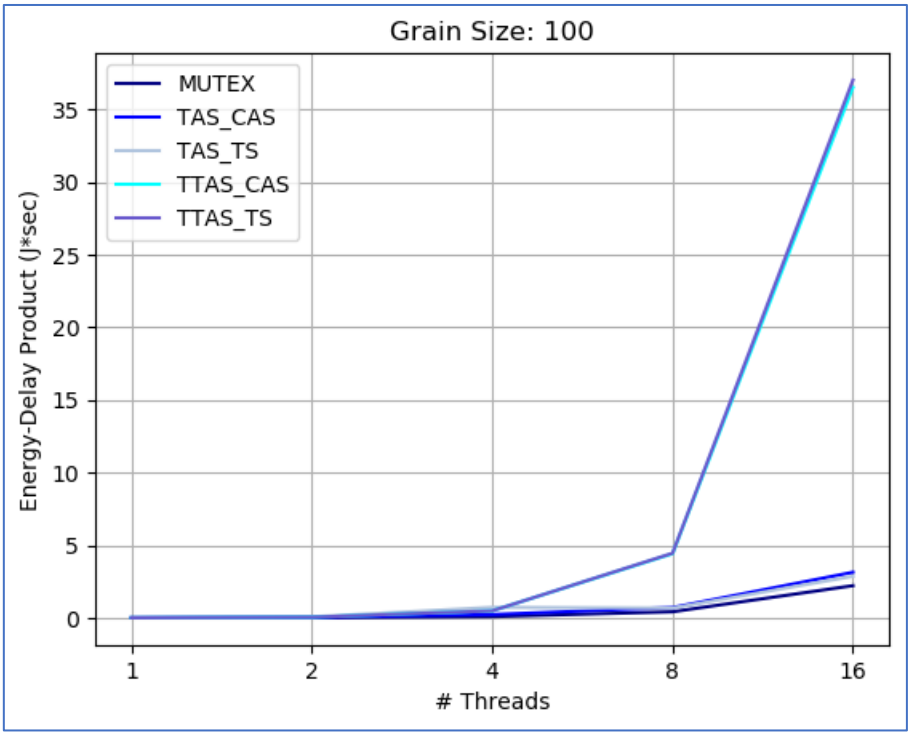


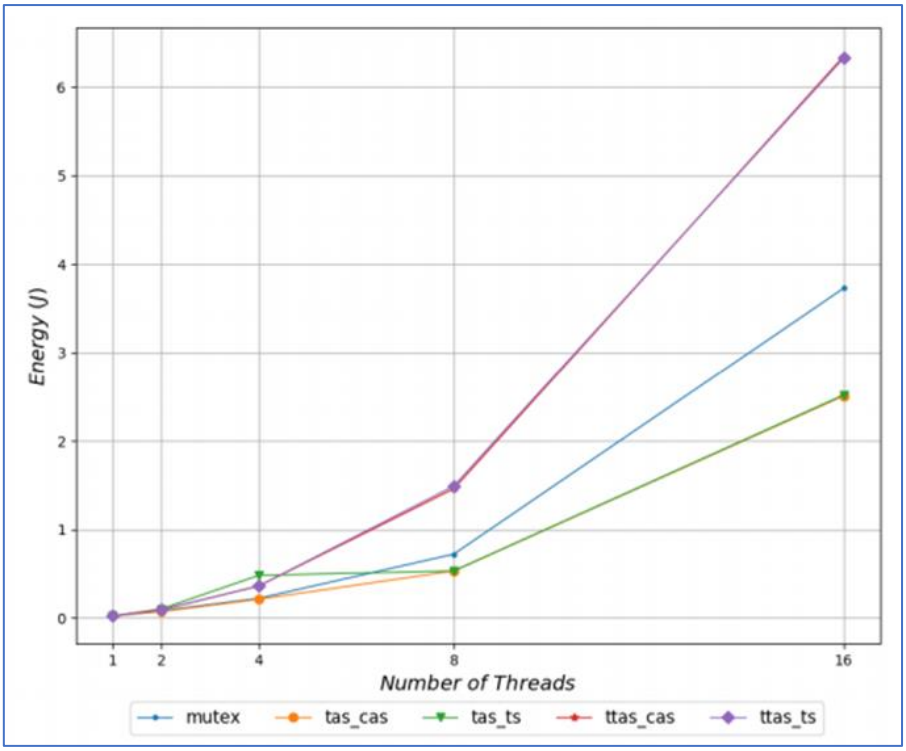

Energy (J)
0.0
0.1
0.2
0.3
0.4
0.5
1
2
4
8
16
Number of Threads
mutex
tas_cas
tas_ts
ttas_cas
ttas_ts

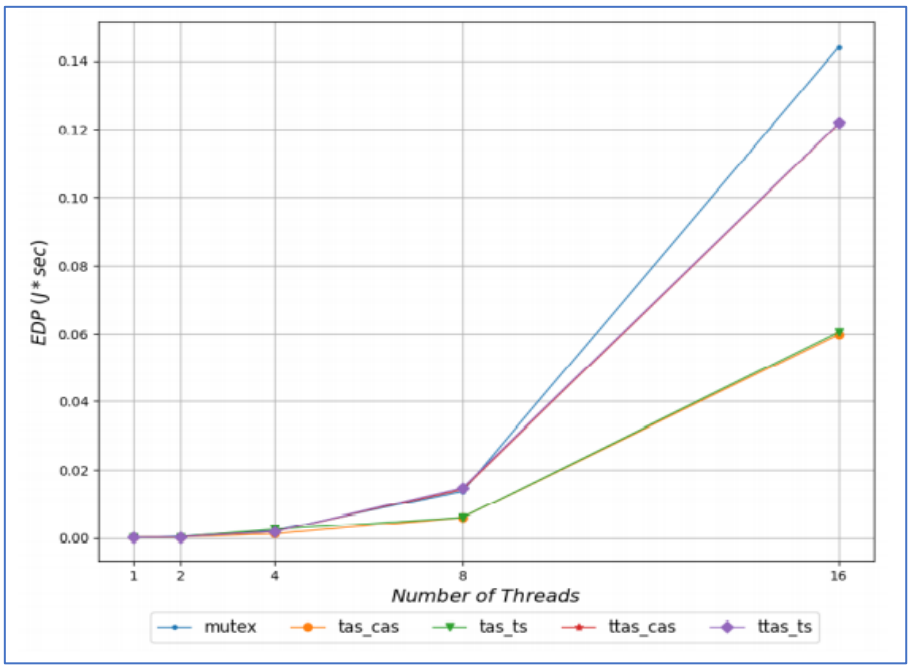

EDP (J* sec)
0.0000
0.0002
0.0004
0.0006
0.0008
0.0010
0.0012
1
2
4
8
16
Number of Threads
mutex
tas_cas
tas_ts
ttas_cas
ttas_ts

Grain size: 100






Energy (J)
0
1
2
3
4
5
6
1
2
4
8
16
Number of Threads
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
mutex
tas_cas
tas_ts
ttas_cas
ttas_ts

Σχολιασμός & Συμπεράσματα:

Παρατηρείται ότι:

- Από τα διαγράμματα Ενέργειας φαίνεται ότι η αύξηση του grain size κατά μία τάξη μεγέθους αυξάνει και την ενέργεια κατά μία τάξη μεγέθους.
- Από τα διαγράμματα της μετρικής Energy-Delay Product φαίνεται ότι η συμπεριφορά των μηχανισμών είναι παρόμοια όταν ο αριθμός των threads είναι από 4 και κάτω και η μετρική έχει αρκετά μικρή τιμή ώστε όλες οι γραμμές να ταυτίζονται με το 0 για τις πιο πάνω γραφικές παραστάσεις.
- Επιπλέον, με κάθε 10πλασιασμό στο grain size το EDP αυξάνεται κατά δύο τάξεις μεγέθους επομένως το μέγεθος της κρίσιμης περιοχής έχει πολύ μεγάλη επίδραση στην επίδοση για τη συγκεκριμένη μετρική.
- Τέλος, και από τις δύο μετρικές ότι οι μηχανισμοί TTAS κλιμακώνουν πολύ χειρότερα από τους υπόλοιπους, ειδικά για 16 threads, και αυτό οφείλεται στα reads που εκτελούν οι διεργασίες που προσπαθούν να καταλάβουν τα locks.
- Η ενέργεια προσεγγίζει εκθετική αύξηση συναρτήσει των νημάτων, με την πιο απότομη αύξηση να παρατηρείται στην μετάβαση από τα 8 νήματα στα 16.
- Για grain size 10 και 100 παρατηρείται από τους μηχανισμούς TTAS_CAS και TTAS_TS μεγάλη κατανάλωση ενέργειας για μεγάλο αριθμό threads, γιατί κάθε νήμα που θέλει να πάρει το κατειλημμένο lock κάνει sniping στην cache του με συνεχή reads, τα οποία επιφέρουν πολλή κατανάλωση ενέργειας σε σχέση με τα stalls, που οφείλονται σε κατειλημμένο bus ή misses στους μηχανισμούς Test-and-Set.
- Για grain size ίσο με 10 φαίνεται ότι οι καμπύλες τείνουν να ταυτιστούν, γεγονός που θα καθιστούσε τους μηχανισμούς εξίσου καλούς με κριτήριο τη μετρική Energy Delay Product (EDP).

* Τα τελευταία 3 σχόλια αφορούν τα πολύχρωμα διαγράμματα παραπάνω.

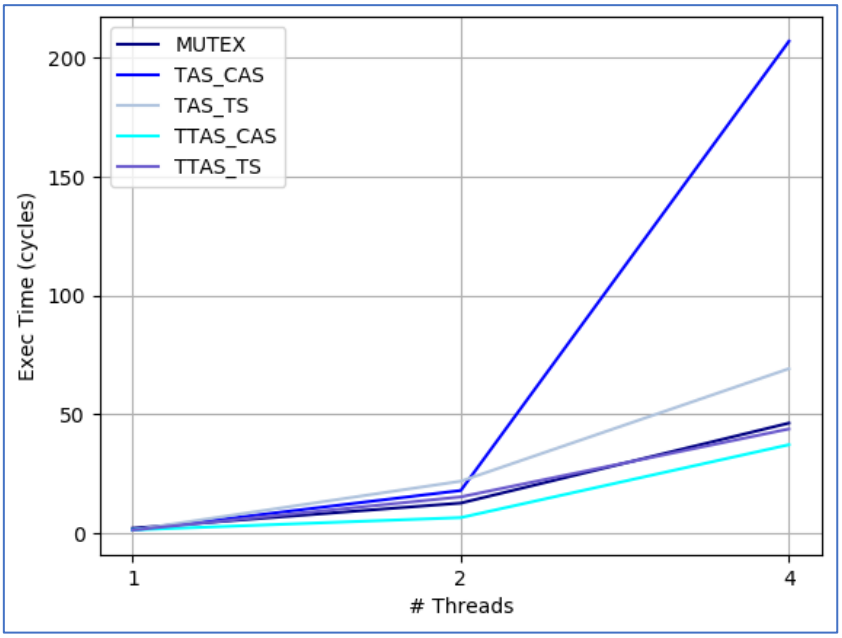
Συνεπώς, προκύπτει ότι ο μηχανισμός TAS είναι οικονομικότερος ενεργειακά, καθώς και ότι το MUTEX έχει τη χειρότερη επίδοση κρίνοντας με τη μετρική EDP για πλήθος νημάτων μεγαλύτερο των 8, ανεξαρτήτως grain size.

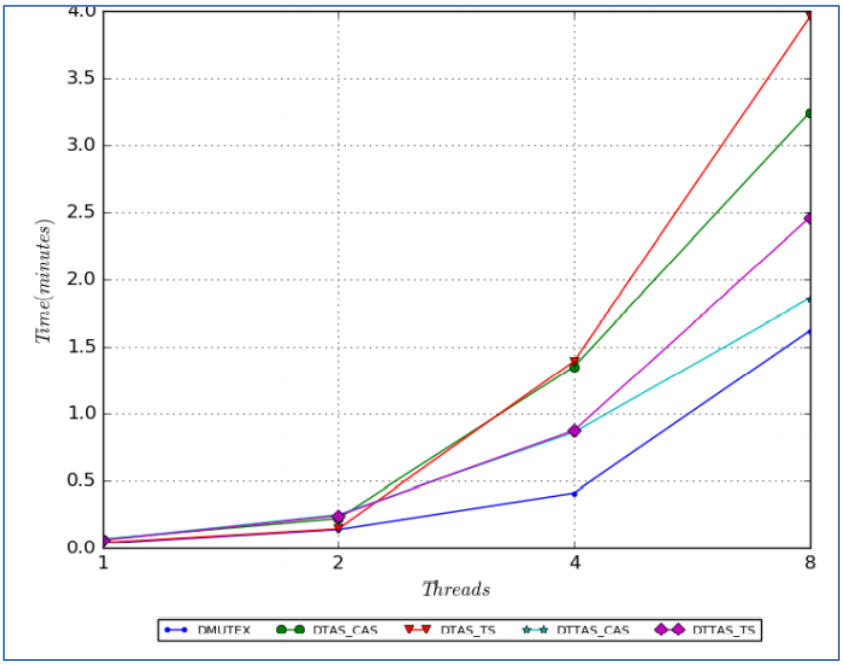
4.1.4. *Μεταγλωττίστε τις διαφορετικές εκδόσεις του κώδικα για πραγματικό σύστημα. Εκτελέστε τα ίδια πειράματα με πριν είτε σε ένα πραγματικό σύστημα, που διαθέτει πολλούς πυρήνες, είτε σε ένα πολυπύρηνο VM σε περίπτωση που έχετε πρόσβαση σε κάποιο. Χρησιμοποιήστε τους ίδιους αριθμούς νημάτων (με μέγιστο αριθμό νημάτων ίσο με τον αριθμό των πυρήνων που διαθέτει το μηχάνημά σας) και τα ίδια grain sizes με πριν. Αυτή τη φορά όμως δώστε έναν αρκετά μεγαλύτερο αριθμό επαναλήψεων ώστε ο χρόνος της εκτέλεσης να είναι επαρκώς μεγάλος για να μπορεί να μετρηθεί με ακρίβεια (π.χ. φροντίστε ώστε η εκτέλεση με 1 νήμα να είναι της τάξης των μερικών δευτερολέπτων). Δώστε τα ίδια διαγράμματα με το ερώτημα 4.1.1. Πώς συγκρίνεται η κλιμακωσιμότητα των διαφορετικών υλοποιήσεων στο πραγματικό σύστημα σε σχέση με το προσομοιωμένο; Δικαιολογήστε τις απαντήσεις σας. Για την εκτέλεση σε πραγματικό μηχάνημα, αφού έχετε μεταγλωττίσει το αρχείο locks με IMPLFLAG=-DREAL, αρκεί να τρέξετε την παρακάτω εντολή:/path/to/binary/locks <nthreads> 1000 <grain_size>*

Παρακάτω φαίνονται τα διαγράμματα για κάθε grain size (1, 10, 100) του χρόνου εκτέλεσης της περιοχής ενδιαφέροντος σε σχέση με τον αριθμό των νημάτων για πραγματικό σύστημα με 4 πυρήνες.

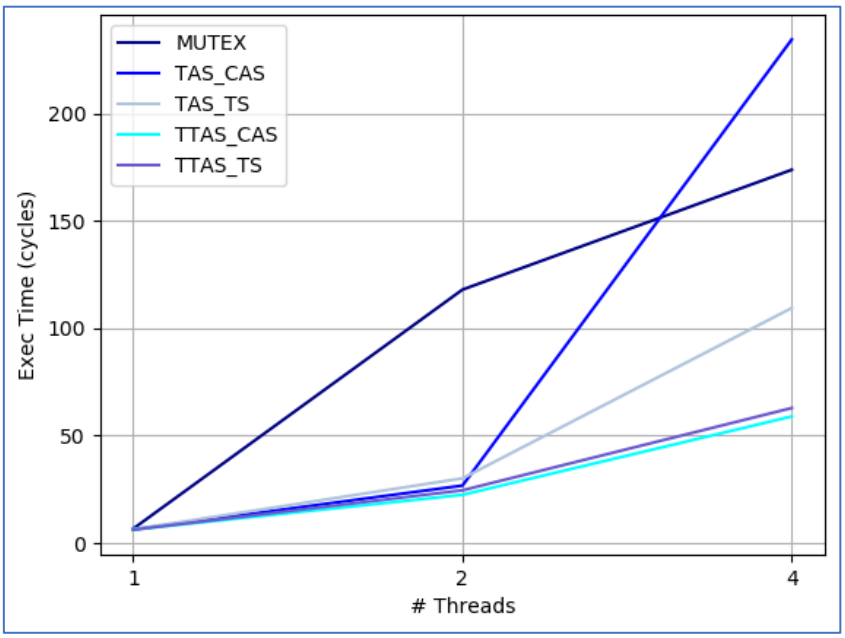
* Τα πολύχρωμα διαγράμματα που φαίνονται παρακάτω, έτρεξαν σε διαφορετικό PC.

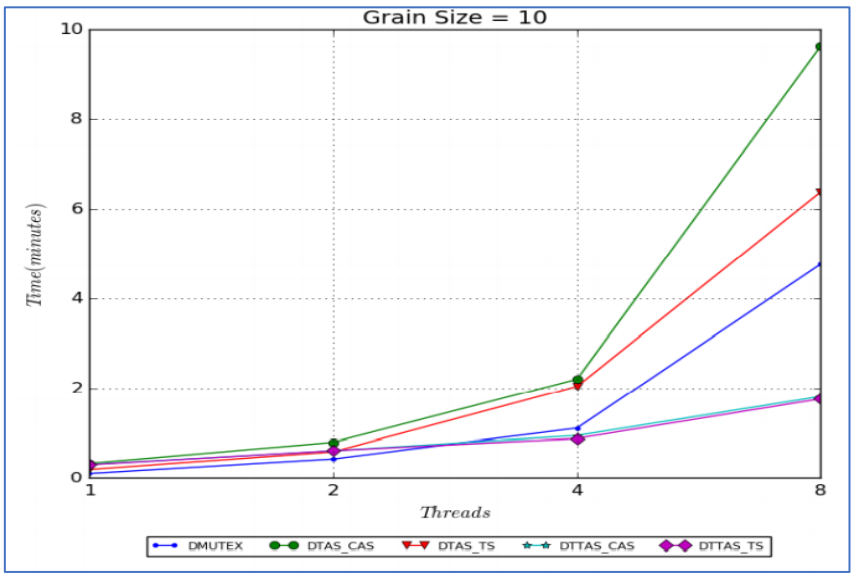
Grain size: 1



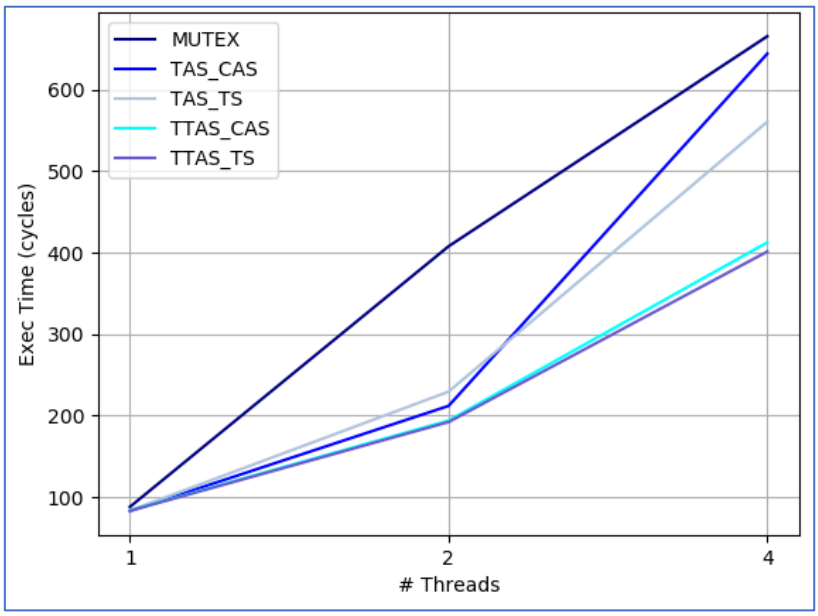


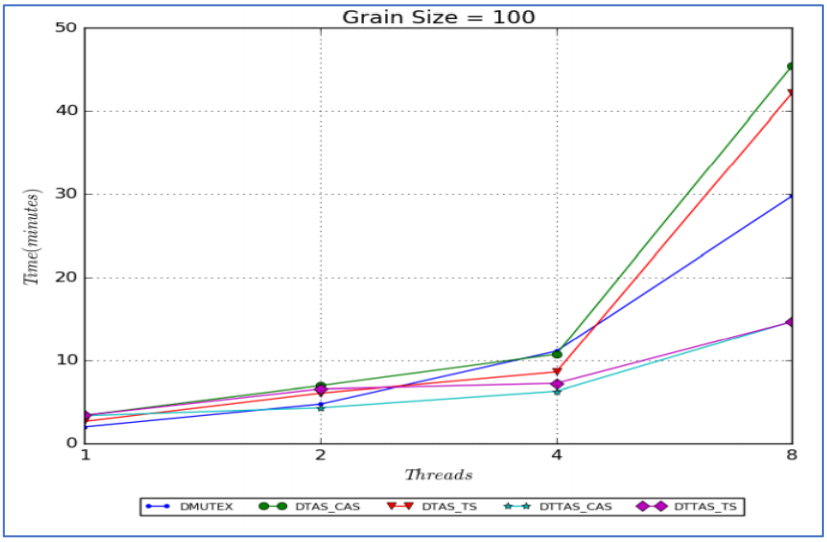
Grain size: 10





Grain size: 100





Σχολιασμός & Συμπεράσματα:

- Καλύτερη κλιμακωσιμότητα επιτυγχάνεται για grain size ίσο με 1 ενώ επίσης φαίνεται ότι και σε αυτή την περίπτωση η αύξηση του grain size επιδεινώνει σημαντικά την επίδοση.
- Πάλι ο μηχανισμός Mutex έχει μία από τις χειρότερες επιδόσεις όταν αυξάνεται ο αριθμός των threads. Όμως, στα πιο πάνω διαγράμματα δεν φαίνεται έντονα η σημαντική υπεροχή των μηχανισμών TTAS έναντι των TAS, αυτό οφείλεται στο μικρό εύρος τιμών για τον αριθμό των threads.
- Γενικά, υπάρχουν έντονες διαφορές συγκριτικά με την εκτέλεση σε προσομοιωμένο σύστημα.
- Τέλος, όσο το grain size αυξάνεται δεν παρατηρείται το ίδιο μοτίβο συμπεριφοράς για τους μηχανισμούς αντιστοίχως.

* Τα τελευταία 2 σχόλια αφορούν τα πολύχρωμα διαγράμματα παραπάνω.

4.2 Τοπολογία νημάτων

Στόχος του ερωτήματος αυτού είναι η αξιολόγηση της κλιμάκωσης των διαφόρων υλοποιήσεων όταν τα νήματα εκτελούνται σε πυρήνες με διαφορετικά χαρακτηριστικά ως προς το διαμοιρασμό των πόρων. Συγκεκριμένα, θεωρούμε τις εξής πειραματικές παραμέτρους:

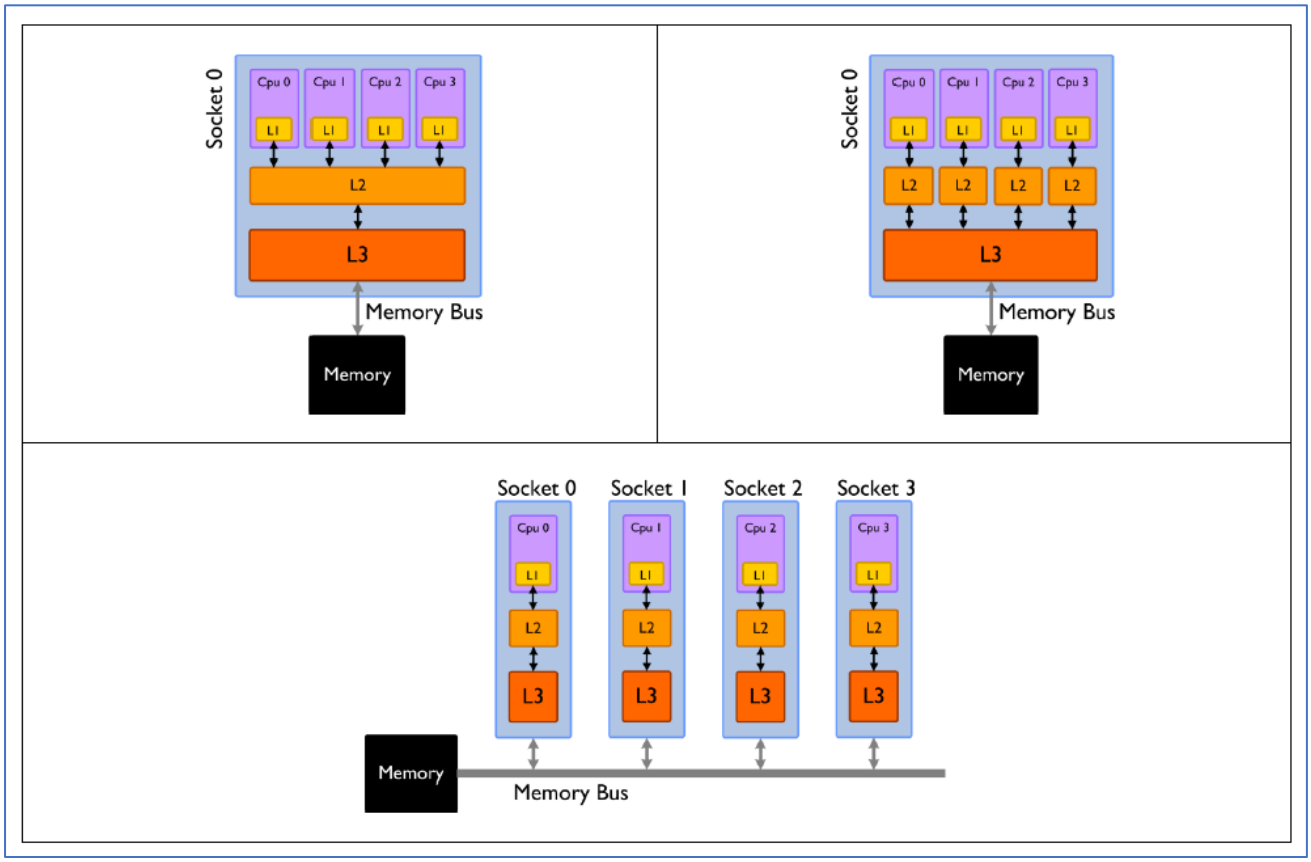
εκδόσεις προγράμματος: TAS_CAS, TAS_TS, TTAS_CAS, TTAS_TS, MUTEX

- *iterations: 1000*
- *nthreads: 4*
- *grain_size: 1*

και εξετάζουμε τις ακόλουθες τοπολογίες:

- *share-all: και τα 4 νήματα βρίσκονται σε πυρήνες με κοινή L2 cache*
- *share-L3: και τα 4 νήματα βρίσκονται σε πυρήνες με κοινή L3 cache, αλλά όχι κοινή L2*
- *share-nothing: και τα 4 νήματα βρίσκονται σε πυρήνες με διαφορετική L3 cache*

Οι τοπολογίες αυτές φαίνονται στα παρακάτω σχήματα.



Για την εκτέλεση των προσομοιώσεων χρησιμοποιήστε και αυτή τη φορά το configuration script ask3.cfg, όπως δείξαμε στην ενότητα 4.1, επανακαθορίζοντας όμως συγκεκριμένες παραμέτρους του script που ορίζουν τον τρόπο διαμοιρασμού συγκεκριμένων επιπέδων της ιεραρχίας μνήμης.

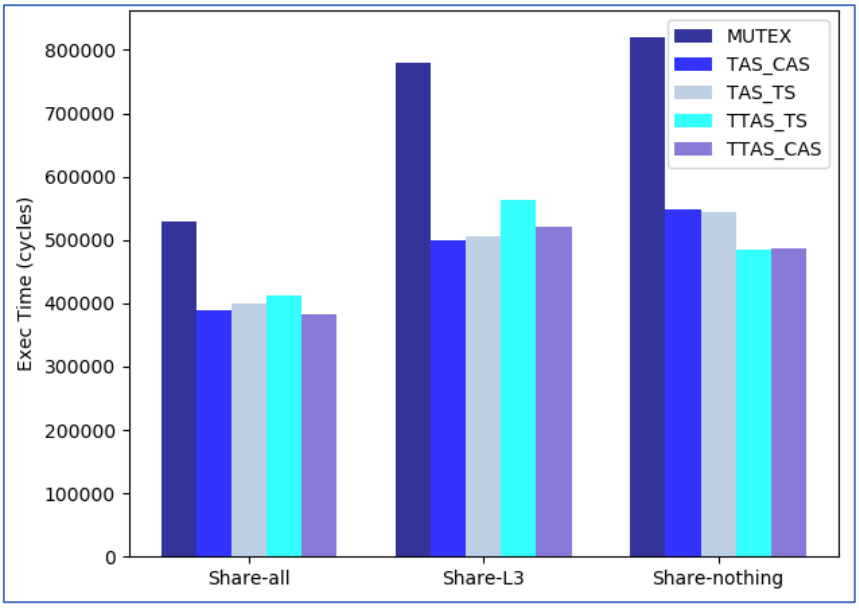
4.2.1. *Για τις παραπάνω τοπολογίες, δώστε σε ένα διάγραμμα το συνολικό χρόνο εκτέλεσης της περιοχής ενδιαφέροντος για όλες τις υλοποιήσεις. Χρησιμοποιώντας το McPAT συμπεριλάβετε στην αξιολόγηση σας και την κατανάλωση ενέργειας (Energy, EDP κτλ.). Τι συμπεράσματα βγάζετε για την απόδοση των μηχανισμών συγχρονισμού σε σχέση με την τοπολογία των νημάτων; Δικαιολογήστε τις απαντήσεις σας.*

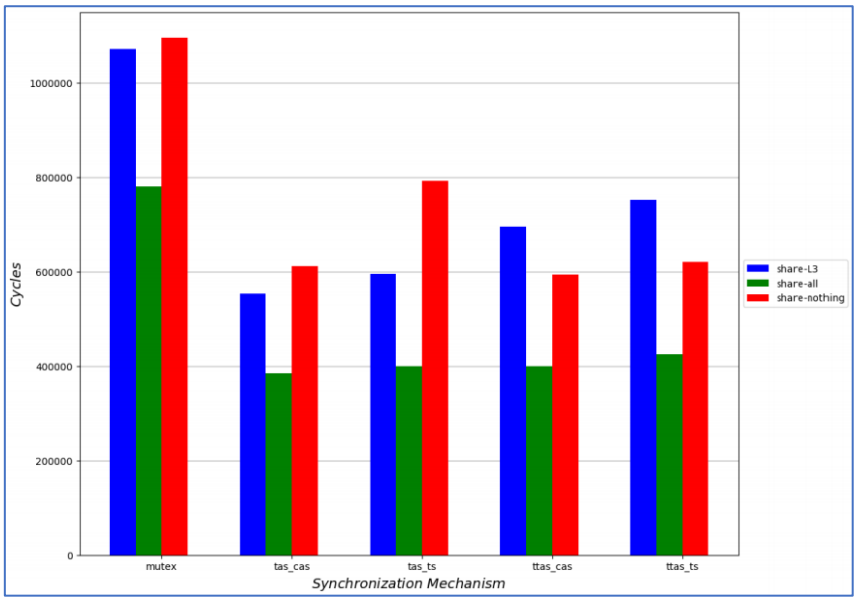
Εδώ, μελετάται η συμπεριφορά της επίδοσης των μηχανισμών ως προς την τοπολογία των νημάτων. Συγκεκριμένα εξετάζονται οι εξής τοπολογίες:

- share-all: και τα 4 νήματα βρίσκονται σε πυρήνες με κοινή L2 cache
- share-L3: και τα 4 νήματα βρίσκονται σε πυρήνες με κοινή L3 cache, αλλά όχι κοινή L2
- share-nothing: και τα 4 νήματα βρίσκονται σε πυρήνες με διαφορετική L3 cache

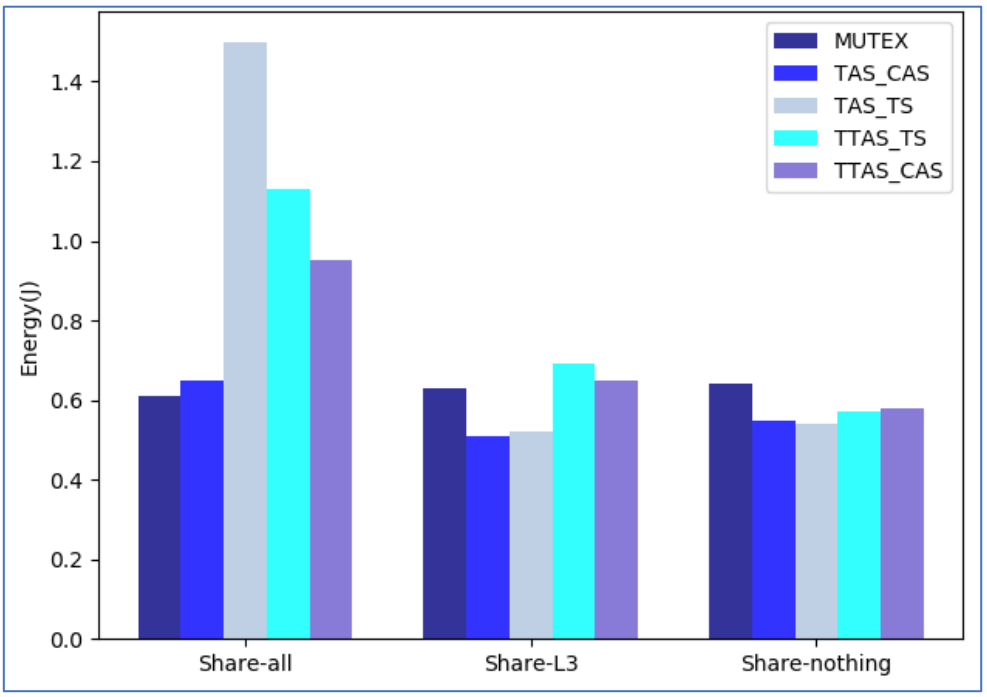
* Τα πολύχρωμα διαγράμματα που φαίνονται παρακάτω, έτρεξαν σε διαφορετικό PC.

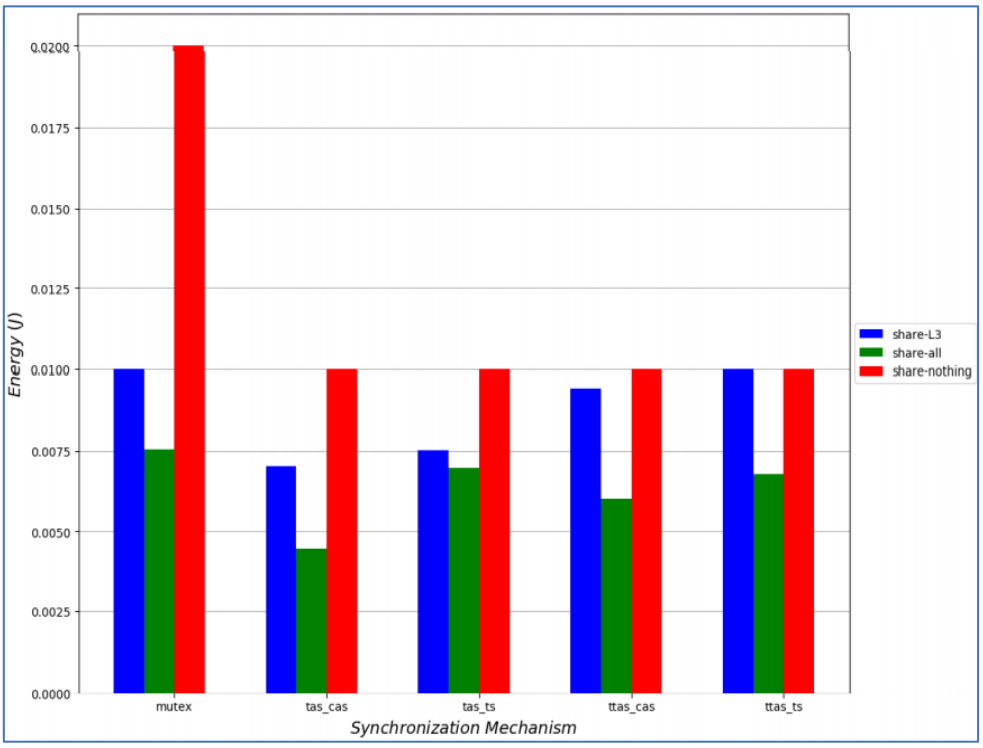
Χρόνος Εκτέλεσης:



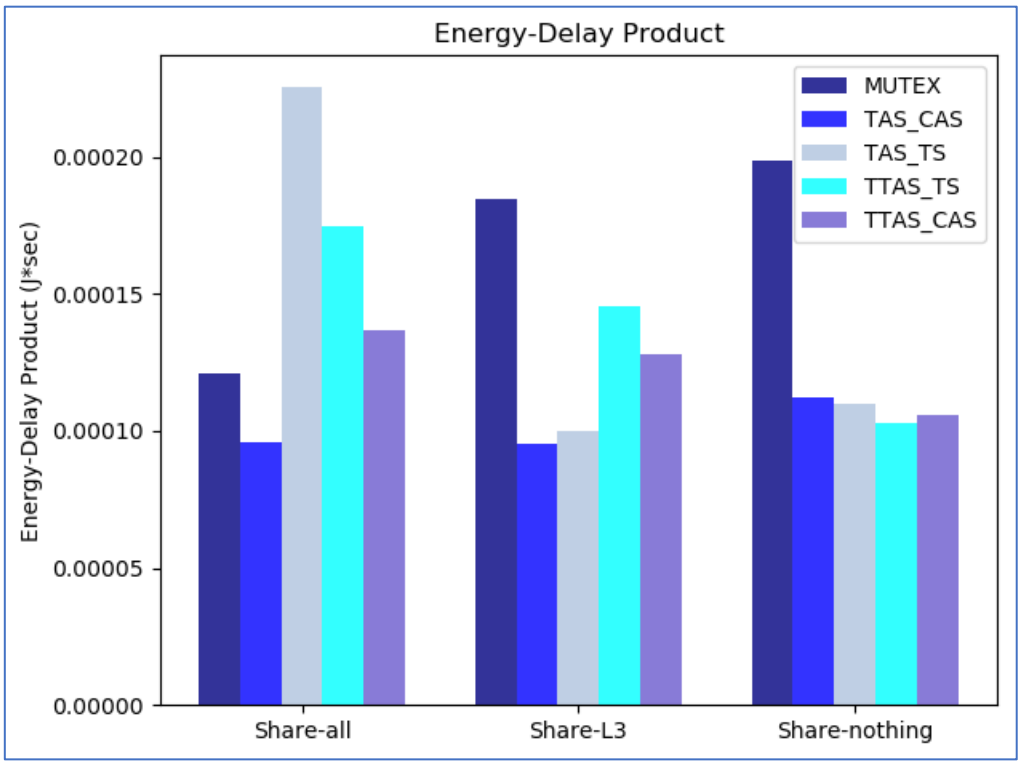


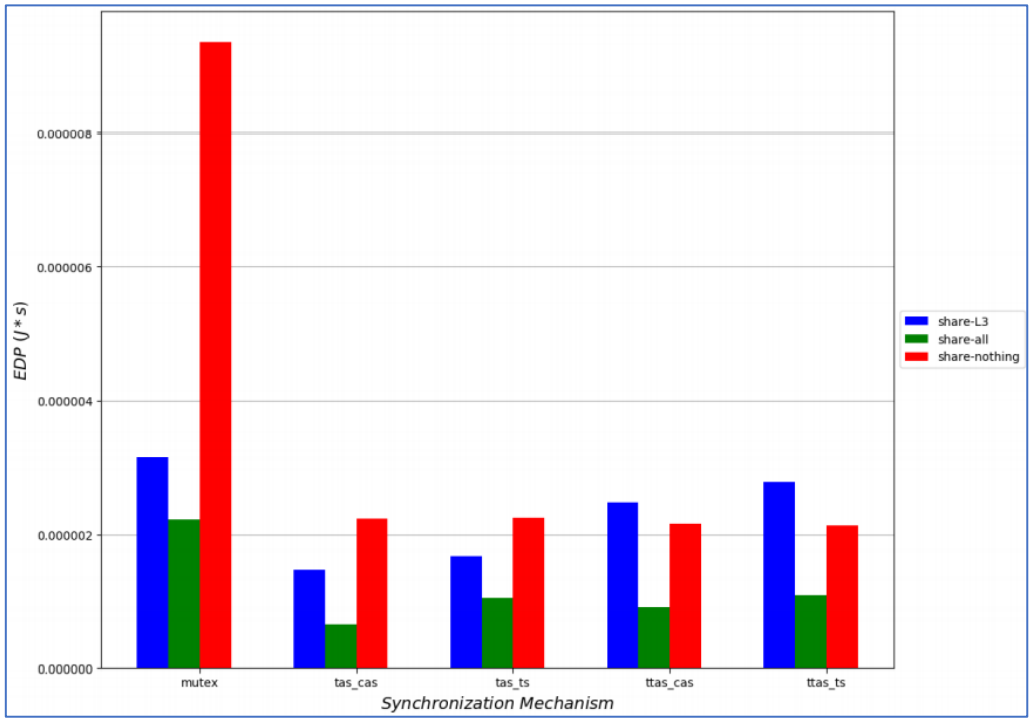
Κατανάλωση Ενέργειας:





EDP:


Energy-Delay Product
Energy-Delay Product (J*sec)
0.00020
0.00015
0.00010
0.00005
0.00000
MUTEX
TAS_CAS
TAS_TS
TTAS_TS
TTAS_CAS
Share-all
Share-L3
Share-nothing


EDP (J * s)
0.000008
0.000006
0.000004
0.000002
0.000000
share-L3
share-all
share-nothing
mutex
tas_cas
tas_ts
ttas_cas
ttas_ts
Synchronization Mechanism

Σχολιασμός & Συμπεράσματα:

Όσον αφορά τον χρόνο εκτέλεσης, φαίνεται ότι η share-all τοπολογία είναι σημαντικά καλύτερη από τις υπόλοιπες, αυτό οφείλεται στο γεγονός ότι οι ζητούμενες invalid cache lines είναι πιο πιθανό να βρίσκονται σε κάποια από τις κοινές μνήμες και αποφεύγεται σε μεγάλο βαθμό η αναζήτηση στην κύρια μνήμη. Οι share-L3 και share-nothing τοπολογίες έχουν παρόμοια συμπεριφορά. Σε κάθε περίπτωση ο χρόνος εκτέλεσης του μηχανισμού mutex είναι χειρότερος από κάθε άλλο. Αξίζει να σημειωθεί ότι για την τοπολογία share-all η καλύτερη επίδοση επιτυγχάνεται από το μηχανισμό TTAS_CAS αλλά η χειρότερη (εκτός mutex) είναι αυτή του TTAS_TS.

Σχετικά με την κατανάλωση ενέργειας, φαίνεται ότι η επίδοση των mutex και TAS_CAS παραμένει σχεδόν σταθερή. Για τους άλλους μηχανισμούς παρατηρείται ότι κατανάλωση είναι πολύ μεγαλύτερη για share-all τεχνολογία. Παρατηρείται επίσης ότι ο μηχανισμός με την καλύτερη επίδοση γενικά είναι ο TAS_CAS. Παράλληλα φαίνεται ότι οι TTAS μηχανισμοί έχουν γενικά χειρότερη επίδοση από τους TAS.

Τέλος, για την μετρική EDP, φαίνεται ότι οι μηχανισμοί TTAS συμπεριφέρονται καλύτερα όταν η τοπολογία δεν χρησιμοποιεί καθόλου κοινές μνήμες ενώ ο μηχανισμός Mutex συμπεριφέρεται καλύτερα όταν οι μνήμες είναι κοινές. Οι μηχανισμοί TAS έχουν παρόμοια συμπεριφορά ανεξάρτητα από την τοπολογία με εξαίρεση τον μηχανισμό TAS_TS που για share-all τοπολογία έχει πολύ κακή συμπεριφορά. Επίσης επιβεβαιώνεται για μία ακόμη φορά ότι ο μηχανισμός mutex έχει γενικά τη χειρότερη συμπεριφορά. Με βάση αυτή τη μετρική, ο μηχανισμός TAS_CAS φαίνεται να έχει γενικά την καλύτερη συμπεριφορά.

Από τις παραπάνω πολύχρωμες γραφικές παραστάσεις προκύπτουν τα εξής συμπεράσματα:

Ο μηχανισμός συγχρονισμού share-all έχει την καλύτερη επίδοση, ως προς το χρόνο και ως προς τη μετρική EDP, ενώ ταυτόχρονα καταναλώνει και την λιγότερη ενέργεια. Αυτό συμβαίνει γιατί σε αυτές πρέπει να γίνει μεταφορά του block του lock μεταξύ των μνημών cache των επεξεργαστών, και όσο πιο “μακριά” στην ιεραρχία βρίσκεται μία μνήμη από τον επεξεργαστή τόσο πιο αργή είναι, με αποτέλεσμα να αυξάνει τον απαιτούμενο χρόνο.

Ο μηχανισμός συγχρονισμού share-nothing έχει τη χειρότερη επίδοση, ενώ ταυτόχρονα καταναλώνει τη περισσότερη ενέργεια.

Ο μηχανισμός συγχρονισμού share-L3 έχει μια ικανοποιητική επίδοση χωρίς όμως να είναι η βέλτιστη, όσον αφορά τους μηχανισμούς MUTEX, TAS_TS, TAS_CAS. Όμως, παρουσιάζει τη χειρότερη απόδοση από τους τρεις μηχανισμούς συγχρονισμού, για τους μηχανισμούς TTAS_TS και TTAS_CAS.