Προηγμένα Θέματα Αρχιτεκτονικής Υπολογιστών
3η άσκηση

Νικόλαος Παγώνας, el18175

1 Εισαγωγή

Στην άσκηση αυτή θα μελετήσουμε τους μηχανισμούς συγχρονισμού και τα πρωτόκολλα συνάφειας κρυφής μνήμης σε σύγχρονες πολυπύρηνες αρχιτεκτονικές, με τη βοήθεια δεδομένου πολυνηματικού κώδικα και του προσομοιωτή `sniper` (ο οποίος αξιοποιεί το `PIN`).

2 Εγκατάσταση και χρήση του προσομοιωτή

2.1 Λήψη, εγκατάσταση και παραμετροποίηση του sniper

Για την εκτέλεση της άσκησης χρησιμοποιούμε την έκδοση 7.3 του `sniper`, μια έκδοση του `PIN` που περιλαμβάνει και το εργαλείο `pinplay`. Αμφότερες οι εκδόσεις βρίσκονται στο site του εργαστηρίου. Μεταγλωττίζουμε τον sniper σε Ubuntu 22.04, με έκδοση gcc 7. Αφού μεταγλωττίσουμε με επιτυχία τον `sniper`, τον εκτελούμε μέσω του `run-sniper`. Για την παραμετροποίηση χρησιμοποιούμε τα κατάλληλα `.cfg` αρχεία, ή την επιλογή `-g` του `run-sniper`.

2.2 McPAT

Για τον υπολογισμό της κατανάλωσης ενέργειας χρησιμοποιούμε το `advcomparch_mcpat.py`, μία τροποποιημένη έκδοση του `mcpat.py` που περιέχεται στον `sniper`. Για να το χρησιμοποιήσουμε, το αντιγράφουμε στον φάκελο `sniper-7.3/tools/`. Με αυτό το script μοντελοποιούμε χαρακτηριστικά όπως η κατανάλωση ενέργειας του επεξεργαστή, καθώς και το μέγεθος που καταλαμβάνουν οι δομικές του μονάδες πάνω στο τσιπ.

2.3 Energy-Delay Product

Μία μετρική που θα χρησιμοποιήσουμε στην ανάλυση που ακολουθεί είναι το Energy-Delay Product (EDP). Ορίζεται ως εξής:

$$EDP = Energy(J) \cdot runtime(sec)$$

και χρησιμοποιείται ώστε να δίνουμε βάρος όχι μόνο στην κατανάλωση ενέργειας, αλλά και στην επίδοση του επεξεργαστή. Παραλλαγές αποτελούν τα ED^2P, ED^3P κ.ο.κ.:

$$ED^2P = Energy(J) \cdot runtime^2(sec)$$

$$ED^3P = Energy(J) \cdot runtime^3(sec)$$

3 Υλοποίηση μηχανισμών συγχρονισμού

Σε αυτό το σημείο καλούμαστε να υλοποιήσουμε τους μηχανισμούς κλειδώματος **Test-and-Set (TAS)** και **Test-and-Test-and-Set (TTAS)**. Στο αρχείο `locks_scalability.c` βρίσκεται κώδικας που χρησιμοποιεί Posix Threads για τη διαχείριση των νημάτων. Κάθε νήμα εκτελεί μία κρίσιμη περιοχή (η είσοδος ελέγχεται από μία κοινή μεταβλητή κλειδιού) για έναν συγκεκριμένο αριθμό επαναλήψεων. Για την απόκτηση και την απελευθέρωση του κλειδιού εκτελούνται οι κατάλληλες ρουτίνες, οι οποίες μπορούν να τροποποιηθούν κατά τη μεταγλώττιση μέσω των flags:

```
-DMUTEX
-DTAS_CAS
-DTAS_TS
-DTTAS_CAS
-DTTAS_TS
```

3.1 Υλοποίηση κλειδωμάτων TAS και TTAS

Συμπληρώνουμε τις απαιτούμενες υλοποιήσεις 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, 0, 1));
}

static inline void spin_lock_ttas_cas(spinlock_t *spin_var)
{
    do {
        while (*spin_var == LOCKED);
    } while (__sync_val_compare_and_swap(spin_var, 0, 1));
}

static inline void spin_lock_tas_ts(spinlock_t *spin_var)
{
    while (__sync_lock_test_and_set(spin_var, 1));
}
```

```
static inline void spin_lock_ttas_ts(spinlock_t *spin_var)
{
    do {
        while (*spin_var == LOCKED);
    } while (__sync_lock_test_and_set(spin_var, 1));
}

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

#endif
```

3.2 Μεταγλώττιση για πραγματικό μηχάνημα ή για προσομοίωση στον sniper

Τέλος, σημειώνεται ότι κατά τη μεταγλώττιση μπορούν να δοθούν τα flags:

```
-DSNIPER
-DREAL
```

ανάλογα με το αν επιθυμούμε η εκτέλεση να γίνει σε πραγματικό μηχάνημα, ή να κάνουμε προσομοίωση στον sniper.

4 Αξιολόγηση επίδοσης

4.1 Σύγκριση υλοποιήσεων

Πολλές φορές οι ρεαλιστικοί περιορισμοί επιβάλλουν κλιμακωσιμότητα χειρότερη της θεωρητικής (που προκύπτει δηλαδή με απλή ανάλυση του αλγορίθμου), ακόμα και για προγράμματα που επιβάλλουν ιδανική κλιμακωσιμότητα[1] εκ κατασκευής. Παράγοντες όπως το overhead που προσθέτει το πρωτόκολλο συνάφειας ή το περιορισμένο bandwidth πρόσβασης στην κύρια μνήμη έχουν ως αποτέλεσμα το κόστος μίας λειτουργίας για ένα νήμα (π.χ. πρόσβαση στη μνήμη) να είναι πολύ μεγαλύτερο, όταν συμμετέχουν πολλά νήματα στην εκτέλεση. Για αυτόν τον λόγο ελέγχουμε πειραματικά, τόσο σε προσομοίωση, όσο και στο δικό μας μηχάνημα, την πραγματική κλιμακωσιμότητα του προγράμματος που μεταγλωττίσαμε παραπάνω, ελέγχοντας πλήθος διαφορετικών υλοποιήσεων (διαφορετικοί μηχανισμοί συγχρονισμού, διαφορετικός αριθμός νημάτων, διαφορετικό `grain_size`). Οι συνδυασμοί παραμέτρων και οι αρχιτεκτονικές διαμοιρασμού που χρησιμοποιούνται για τους υπολογισμούς αναγράφονται αναλυτικά στην εκφώνηση της άσκησης.

4.1.1

Σε αυτό το μέρος της άσκησης, για κάθε grain size, απεικονίζουμε τον χρόνο εκτέλεσης σε κύκλους συναρτήσει του αριθμού των νημάτων. Σε κάθε διάγραμμα συμπεριλαμβάνονται και οι

[1] Με τον όρο "ιδανική κλιμακωσιμότητα" αναφερόμαστε σε προγράμματα που έχουν σχεδιαστεί ώστε ο χρόνος εκτέλεσης με N νήματα να είναι $1/N$ του χρόνου εκτέλεσης για 1 νήμα.

5 μηχανισμοί συγχρονισμού (μία καμπύλη για κάθε μηχανισμό). Ακολουθούν τα διαγράμματα **για grain size ίσο με 1, 10 και 100** αντίστοιχα. Προσοχή στον κατακόρυφο άξονα, τα μεγέθη είναι πολλαπλασιασμένα $\times 10^6$, $\times 10^8$ κλπ.:

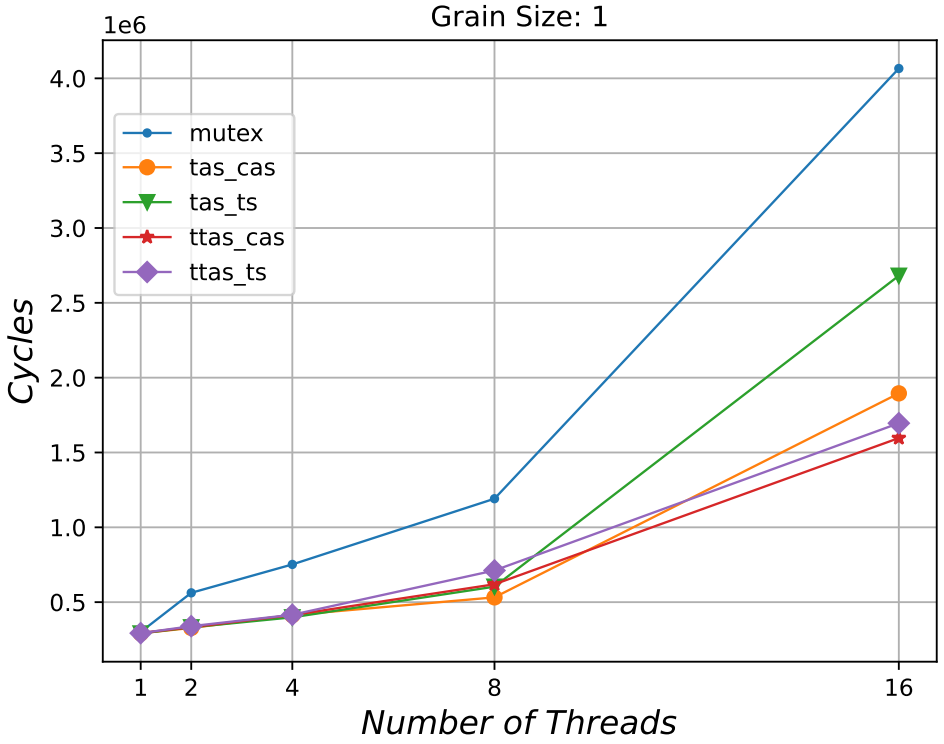

Figure 1: Χρόνος εκτέλεσης σε κύκλους συναρτήσει του αριθμού νημάτων, για grain size 1

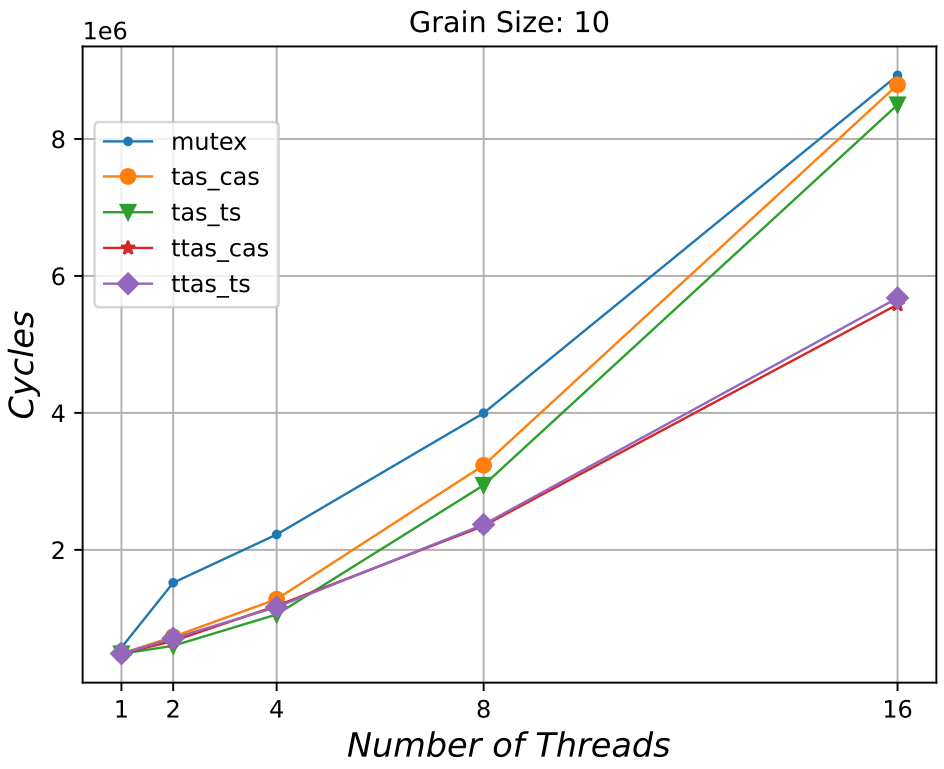

Figure 2: Χρόνος εκτέλεσης σε κύκλους συναρτήσει του αριθμού νημάτων, για grain size 10

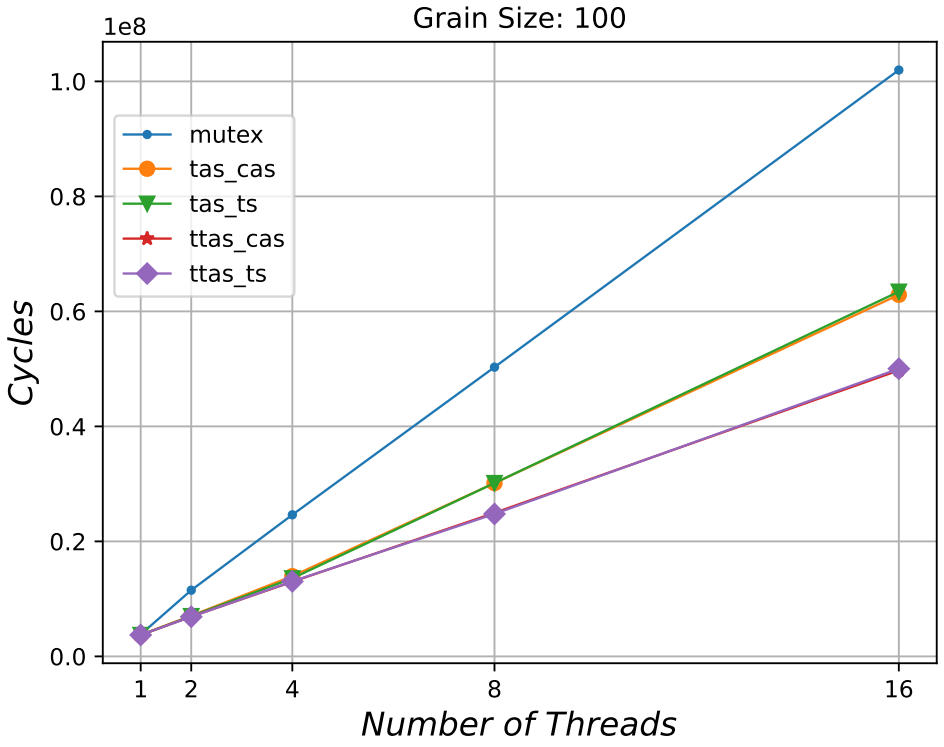

Figure 3: Χρόνος εκτέλεσης σε κύκλους συναρτήσει του αριθμού νημάτων, για grain size 100

4.1.2

Παρατηρούμε ότι οι μηχανισμοί TAS_CAS και TAS_TS παρουσιάζουν επιδόσεις κοντινές μεταξύ τους, κάτι που είναι λογικό, αφού και οι δύο αποτελούν υλοποιήσεις του μηχανισμού Test-and-Set. Όταν δύο ή περισσότεροι επεξεργαστές προσπαθούν να αποκτήσουν το κλείδωμα, οδηγούμαστε σε συνεχόμενες εναλλαγές καταστάσεων (Modified - Invalid) της cache line που περιέχει το κλείδωμα, οπότε δημιουργείται αυξημένη περιττή κίνηση πάνω στο bus.

Αντίστοιχα οι μηχανισμοί TTAS_CAS και TTAS_TS υλοποιούν τον μηχανισμό Test-and-Test-and-Set, οπότε και πάλι οι επιδόσεις τους είναι πολύ παραπλήσιες. Η διαφορά με τον μηχανισμό Test-and-Set είναι ότι δεν γράφουν συνεχώς πάνω στο κλείδωμα. Αντίθετα, εκτελούν πρώτα ένα load για να δουν αν είναι ελεύθερο το κλείδωμα (test), και μόνο όταν είναι ελεύθερο δοκιμάζουν να το γράψουν/δεσμεύσουν ατομικά (test-and-set). Έτσι, όταν το κλείδωμα δεν είναι διαθέσιμο οι επεξεργαστές κάνουν spin τοπικά (στην cache τους), οπότε αποφεύγεται η ανούσια ευρυεκπομπή στον διάδρομο. Γι' αυτό τον λόγο είναι αναμενόμενο να οι μηχανισμοί TTAS να έχουν καλύτερη κλιμακωσιμότητα από τους άλλους.

Από τα παραπάνω διαγράμματα φαίνεται ότι οι χρόνος εκτέλεσης είναι (σχεδόν) γραμμική συνάρτηση του πλήθους των νημάτων (μικρή απόκλιση από την γραμμικότητα παρατηρείται για grain size ίσο με 1). Βέβαια, οι κλίσεις των ευθειών αλλάζουν ανάλογα τον μηχανισμό συγχρονισμού.

Τη χειρότερη κλιμακωσιμότητα έχει ο μηχανισμός MUTEX, για όλα τα grain sizes (αν και οι μηχανισμοί TAS_CAS & TAS_TS πλησιάζουν το MUTEX για grain size ίσο με 10). Όπως είναι λογικό, ακολουθούν οι μηχανισμοί TAS_CAS & TAS_TS (καλύτεροι από MUTEX αλλά χειρότεροι από TTAS_CAS & TTAS_TS), ενώ οι καλύτεροι είναι οι μηχανισμοί TTAS_CAS & TTAS_TS.

Επιπλέον παρατηρούμε ότι η αύξηση του grain size (πλήθος dummy υπολογισμών) προκαλεί βελτίωση τόσο της γραμμικότητας, όσο και της κλίσης των ευθειών (ιδιαίτερα το MUTEX βελτιώνεται πολύ, αν συγκρίνουμε την μεγάλη του απόκλιση από την γραμμικότητα για grain size ίσο με 1). Αυτό συμβαίνει επειδή αφιερώνεται περισσότερος χρόνος σε "ωφέλιμους" υπολογισμούς, και λιγότερος χρόνος σε ανταγωνισμό των threads για απόκτηση του κλειδώματος, οπότε η επίδοση βελτιώνεται.

Φυσικά, περισσότεροι dummy υπολογισμοί σημαίνουν μεγαλύτερος χρόνος εκτέλεσης (αύξηση $\approx$1 τάξη μεγέθους ανά δεκαπλασιασμό του grain size).

4.1.3

Εδώ, χρησιμοποιούμε το McPAT για να απεικονίσουμε την κατανάλωση ενέργειας και το EDP, **για grain size ίσο με 1, 10 και 100** αντίστοιχα.

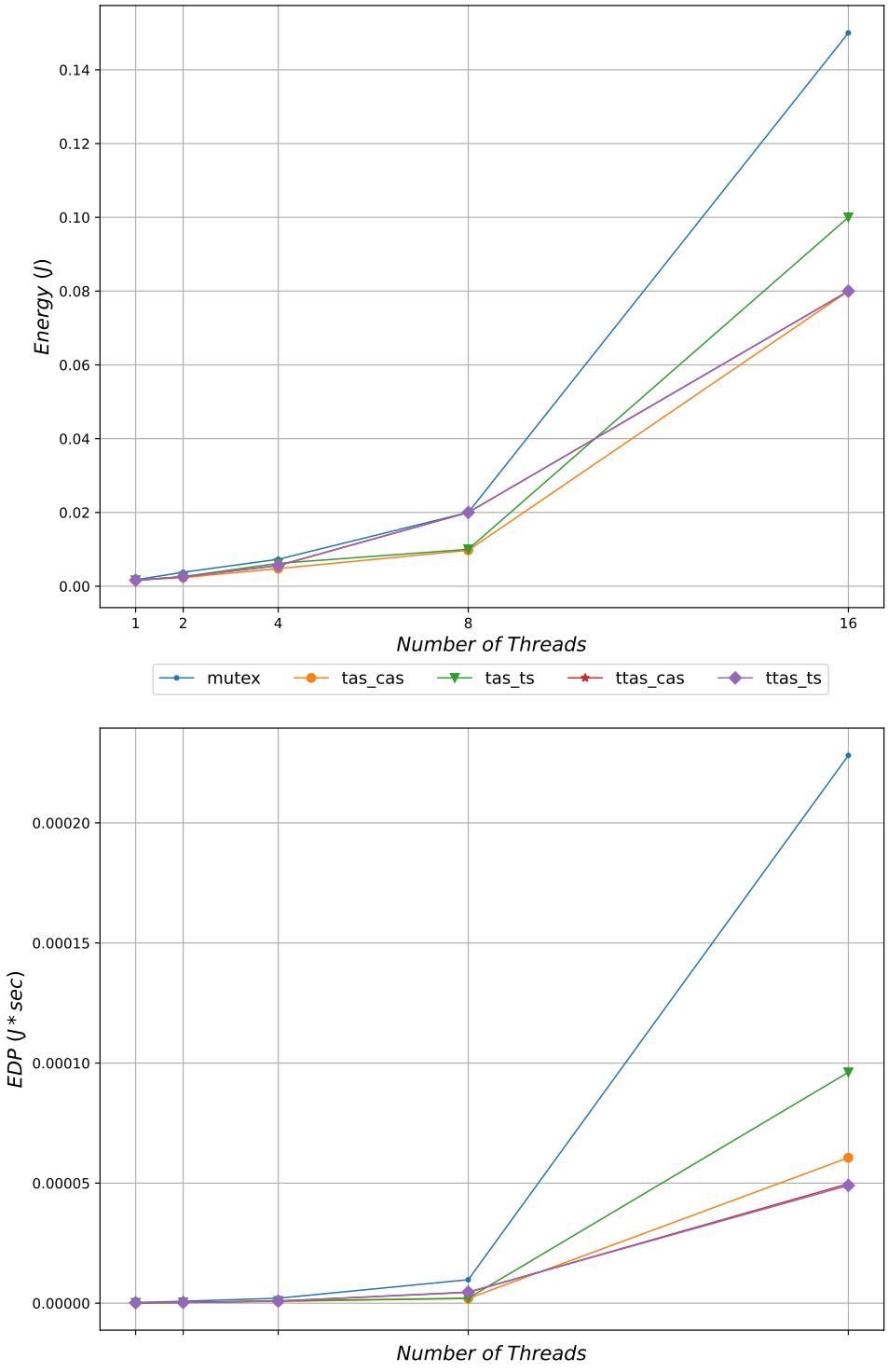

Figure 4: Κατανάλωση ενέργειας και EDP, για grain size ίσο με 1

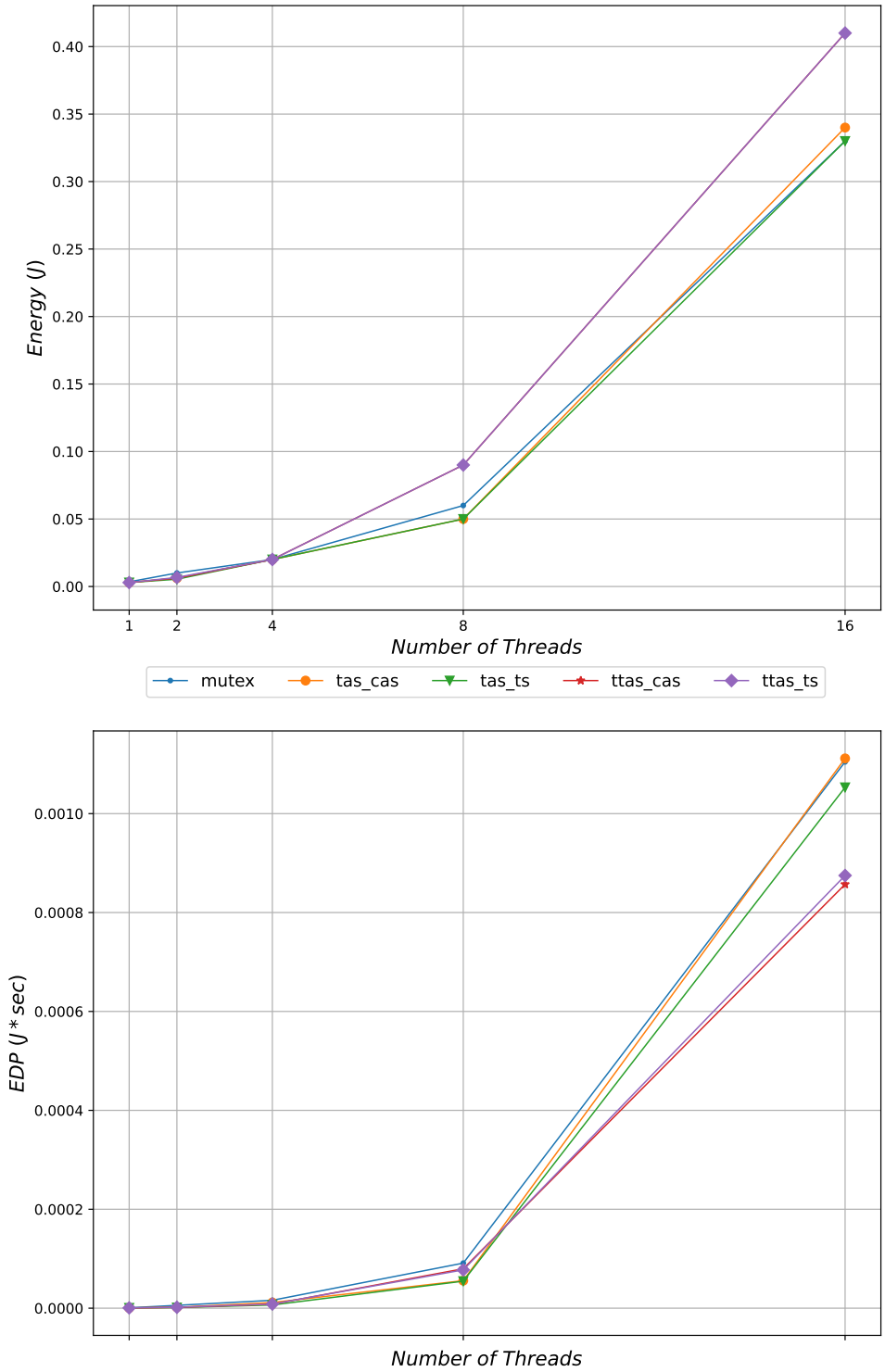

Figure 5: Κατανάλωση ενέργειας και EDP, για grain size ίσο με 10

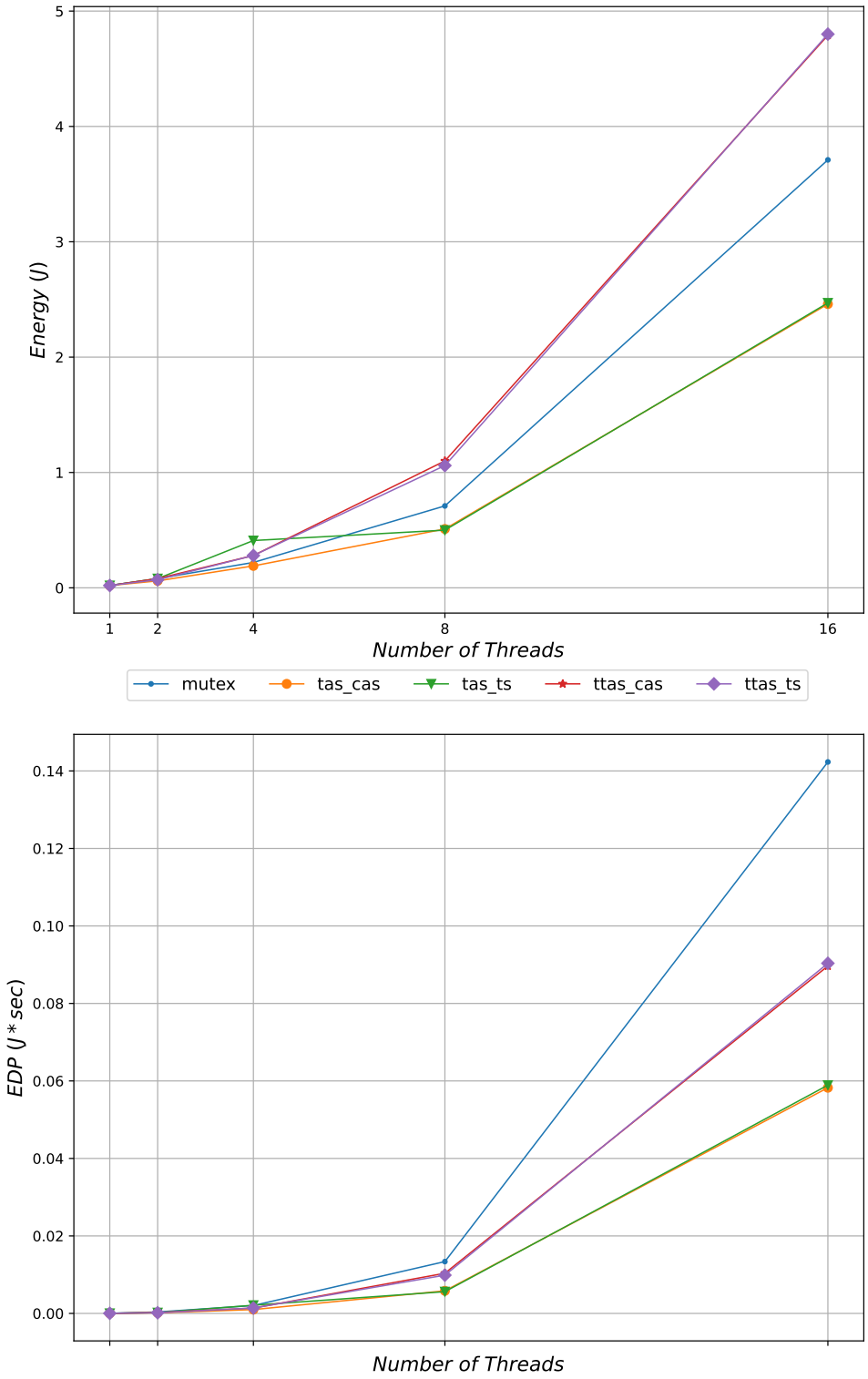

Figure 6: Κατανάλωση ενέργειας και EDP, για grain size ίσο με 100

Παρατηρήσεις - Συμπεράσματα

Παρατηρούμε μία αύξηση της ενέργειας που θα μπορούσε να χαρακτηριστεί εκθετική. Ειδικά η αύξηση της ενέργειας από 8 σε 16 νήματα είναι αξιοσημείωτη. Επιπλέον, για grain size 10 και 100, οι μηχανισμοί `TTAS_CAS` & `TTAS_TS` καταναλώνουν αρκετά περισσότερη ενέργεια

σε σχέση με τους υπόλοιπους μηχανισμούς, για μεγάλο αριθμό νημάτων (8-16). Αυτό είναι λογικό, αφού οι μηχανισμοί αυτοί εκτελούν συνεχόμενα reads, τα οποία έχουν μεγαλύτερο ενεργειακό κόστος σε σχέση με τα stalls που προκαλεί ένα cache miss/κατειλημμένο bus στις `Test-and-Set` υλοποιήσεις. Γενικά, φθηνότερος από άποψη ενεργειακής κατανάλωσης φαίνεται πως είναι ο μηχανισμός `TAS`.

Όσον αφορά το EDP, οι μηχανισμοί έχουν παραπλήσιες επιδόσεις για grain size 10, ωστόσο καλύτερο EDP γενικά επιτυγχάνουν οι μηχανισμοί `TAS`. Τέλος, ο μηχανισμός `MUTEX` έχει σταθερά το χειρότερο EDP, για μεγάλο πλήθος νημάτων (8-16).

4.1.4

Σε αυτό το σημείο μεταγλωττίζουμε τις διαφορετικές εκδόσεις του κώδικα για **πραγματικό** σύστημα. Να σημειωθεί εδώ ότι **το πραγματικό μηχάνημα μας επιτρέπει να χρησιμοποιήσουμε μέχρι 8 νήματα** (αντί για 16 που είχαμε στα προηγούμενα πειράματα). Χρησιμοποιούμε τις ίδιες παραμέτρους με πριν, με τη διαφορά ότι ο αριθμός επαναλήψεων είναι κατά πολύ αυξημένος (150.000.000), ώστε να μπορούμε να μετρήσουμε με ακρίβεια τον χρόνο εκτέλεσης.

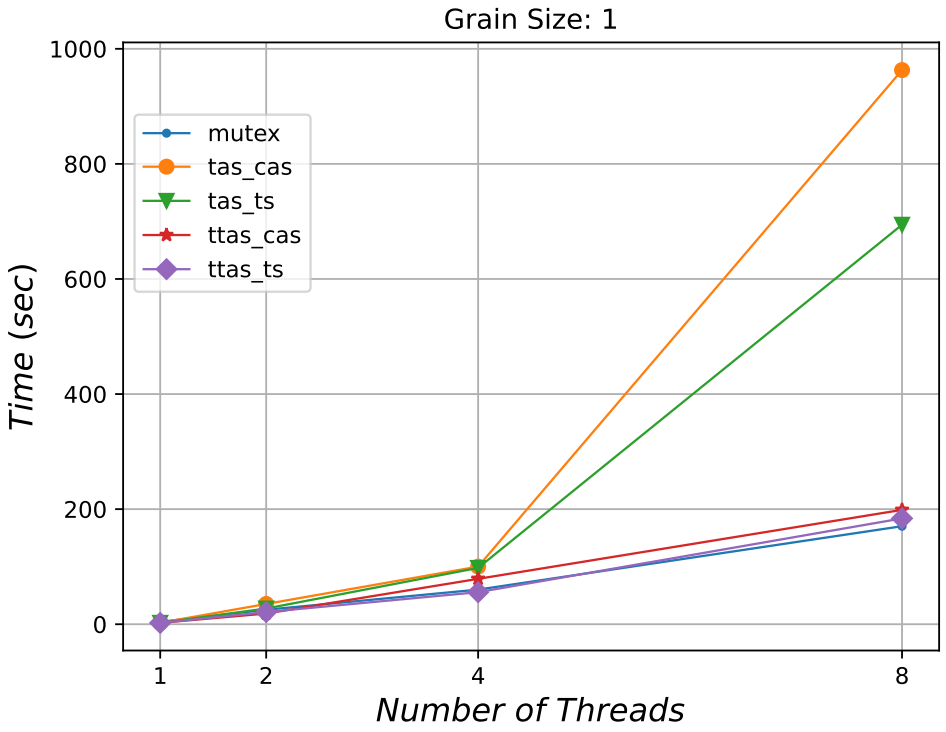

Figure 7: Χρόνος εκτέλεσης σε δευτερόλεπτα (πραγματικό σύστημα), για grain size ίσο με 1

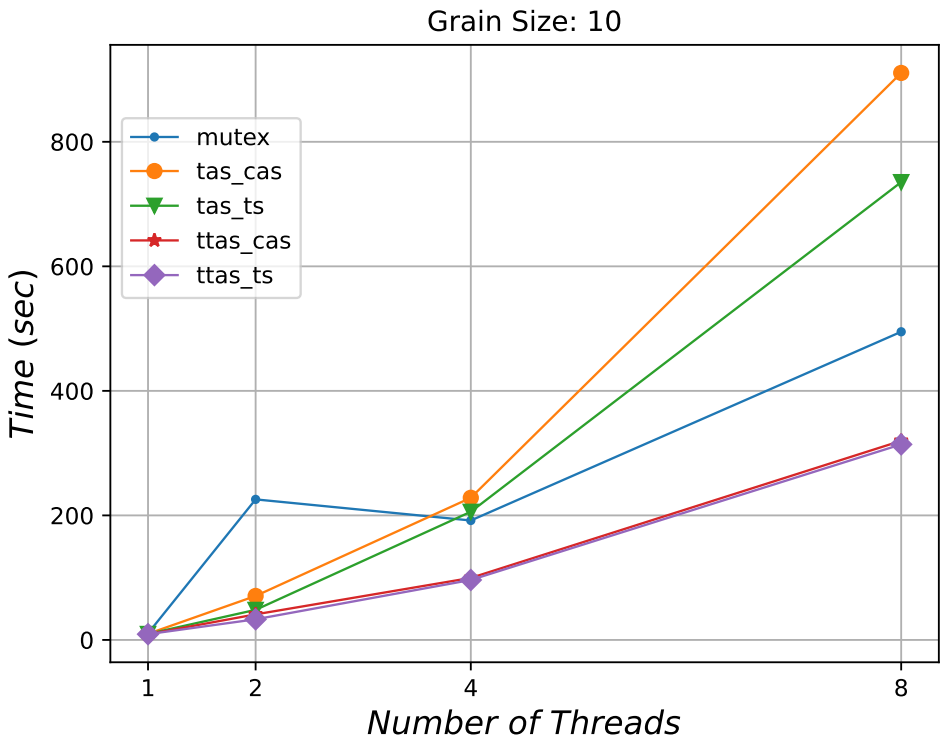

Figure 8: Χρόνος εκτέλεσης σε δευτερόλεπτα (πραγματικό σύστημα), για grain size ίσο με 10

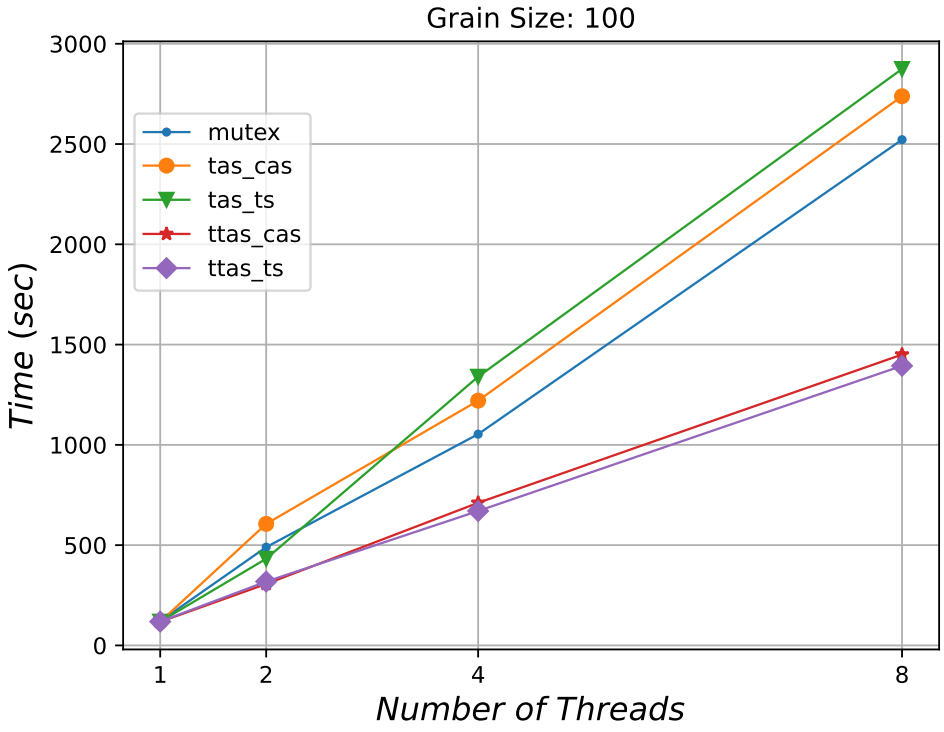

Figure 9: Χρόνος εκτέλεσης σε δευτερόλεπτα (πραγματικό σύστημα), για grain size ίσο με 100

Παρατηρήσεις - Συμπεράσματα

Κι εδώ παρατηρούμε γραμμική κλιμάκωση, όπως και στην περίπτωση που είχαμε προσομοίωση με τον sniper. Σημαντική απόκλιση από την γραμμικότητα έχουμε μόνο κατά τη μετάβαση από 4 σε 8 νήματα για grain size ίσο με 1, και για τους μηχανισμούς `TAS_CAS & TAS_TS`. Αυτή τη φορά βέβαια, οι μηχανισμοί `TTAS` έχουν σημαντικά καλύτερη επίδοση. Ειδικά για grain size ίσο με 1, ο χρόνος παραμένει σχεδόν σταθερός με αύξηση των νημάτων.

Άξιο προσοχής είναι επίσης το γεγονός ότι οι δύο μηχανισμοί `TAS_CAS & TAS_TS` αποκλίνουν σημαντικά στην επίδοση τους, όσο αυξάνεται ο αριθμός των νημάτων. Αυτή η διαφορά φαίνεται να εκφυλίζεται όσο αυξάνεται το grain size. Μάλιστα, η επίδοση των `TAS` μηχανισμών είναι χειρότερη από αυτή του `MUTEX`. Τέλος, σημαντικό είναι επίσης το ότι ο μηχανισμός `MUTEX` έχει παραπλήσια επίδοση με τους `TTAS` μηχανισμούς, για grain size ίσο με 1.

4.2 Τοπολογία νημάτων

4.2.1

Στο ερώτημα αυτό σκοπός είναι να αξιολογήσουμε την κλιμάκωση των διαφόρων υλοποιήσεων όταν τα νήματα εκτελούνται σε πυρήνες με διαφορετικά χαρακτηριστικά ως προς τον διαμοιρασμό των πόρων. Επιλέγουμε 1000 iterations, 4 threads, και grain size ίσο με 1. Εξετάζουμε τις 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 cache
- **share-nothing**: και τα 4 νήματα βρίσκονται σε πυρήνες με διαφορετική L3 cache

Ακολουθούν κατά σειρά τα διαγράμματα:

- Κύκλοι ρολογιού
- Κατανάλωση ενέργειας
- EDP

Για κάθε μηχανισμό συγχρονισμού, απεικονίζονται οι 3 τοπολογίες share-all/L3/nothing. Προσοχή στον κατακόρυφο άξονα, τα μεγέθη είναι πολλαπλασιασμένα $\times 10^6$, $\times 10^{-6}$ κλπ:

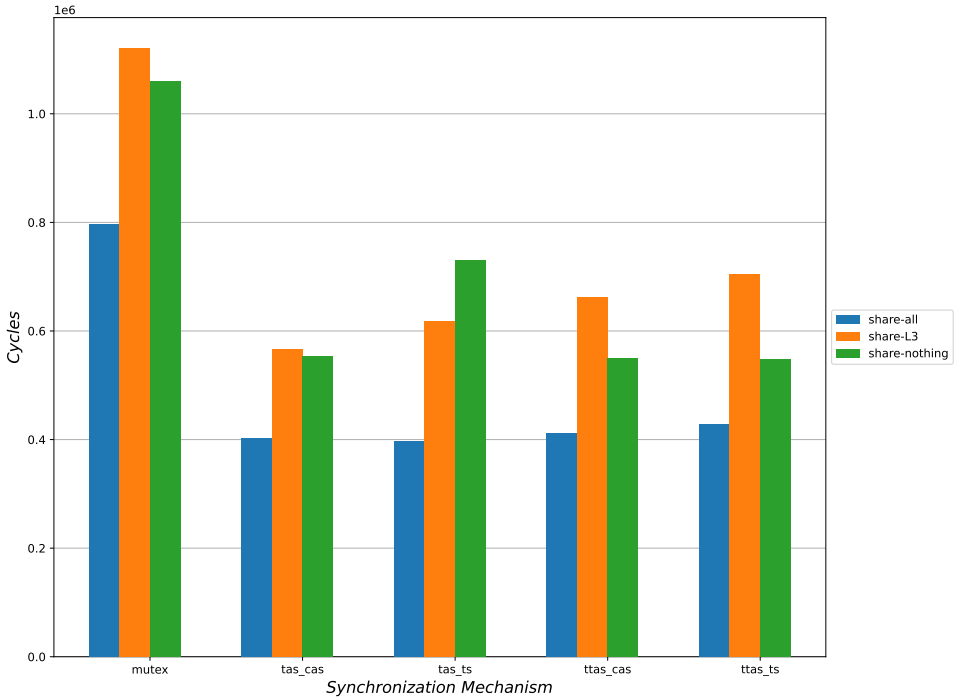

Figure 10: Χρόνος εκτέλεσης σε κύκλους

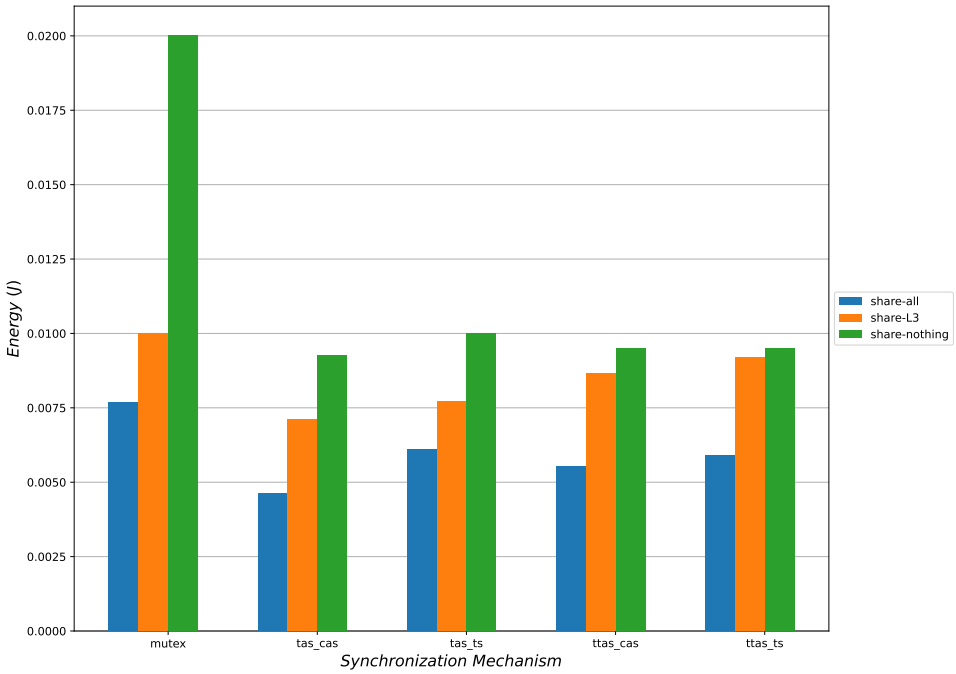

Figure 11: Κατανάλωση ενέργειας σε J

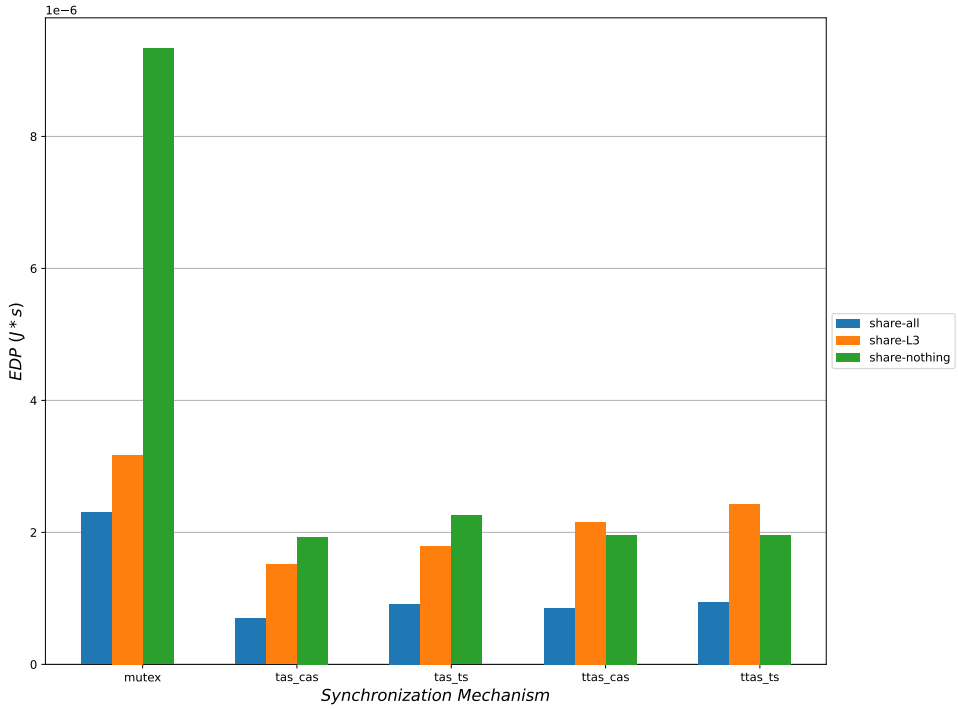

Figure 12: EDP σε J × sec

Παρατηρήσεις - Συμπεράσματα

Όσον αφορά τον χρόνο εκτέλεσης, για όλους τους μηχανισμούς συγχρονισμού η καλύτερη επίδοση επιτυγχάνεται με την τοπολογία share-all. Για τους μηχανισμούς `MUTEX`, `TAS_TS`, `TAS_CAS` ακολουθεί η τοπολογία share-L3 και ύστερα η share-nothing. Αντίθετα, για τους μηχανισμούς `TTAS_TS & TTAS_CAS` ακολουθεί η τοπολογία share-nothing, και ύστερα η share-L3.

Ο λόγος για τον οποίο η share-all τοπολογία είναι βέλτιστη είναι επειδή αν ο επεξεργαστής ζητήσει ένα invalid cache line, τότε θα ενημερωθεί η ιεραρχία μέχρι την L2, και ύστερα θα διαβαστεί από εκεί το δεδομένο, ενώ στο share-L3 η invalid cache line θα ζητηθεί από την L3, και αντίστοιχα στο share-nothing θα χρειαστεί να φτάσουμε μέχρι την κύρια μνήμη, οπότε ο χρόνος που απαιτείται αυξάνεται σημαντικά.

Όσον αφορά την μετρική της ενέργειας, η τοπολογία share-all έχει την μικρότερη κατανάλωση, ενώ η share-nothing έχει την μεγαλύτερη κατανάλωση. Αντίστοιχα για το EDP, καλύτερη επίδοση έχει η τοπολογία share-all (ελάχιστο προβάδισμα έχει η τοπολογία share-L3 στις περιπτώσεις `TTAS_CAS & TTAS_TS`), ενώ την χειρότερη η τοπολογία share-nothing.