ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

4η ΑΣΚΗΣΗ

Ακαδημαϊκό Έτος 2019-2020

Εξάμηνο 8ο

Χρόνης Σάκος

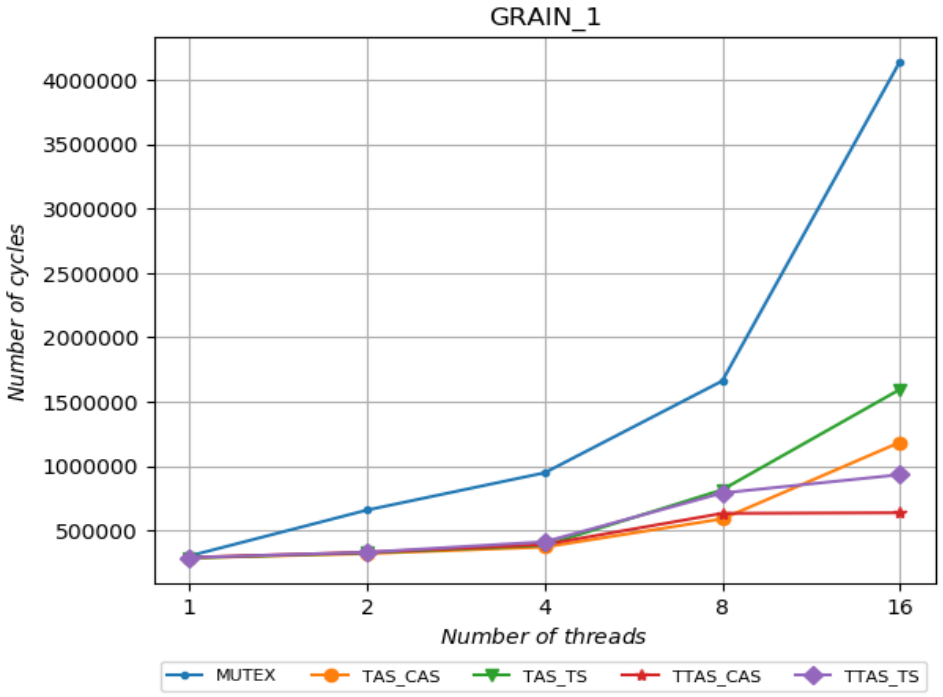
03116168

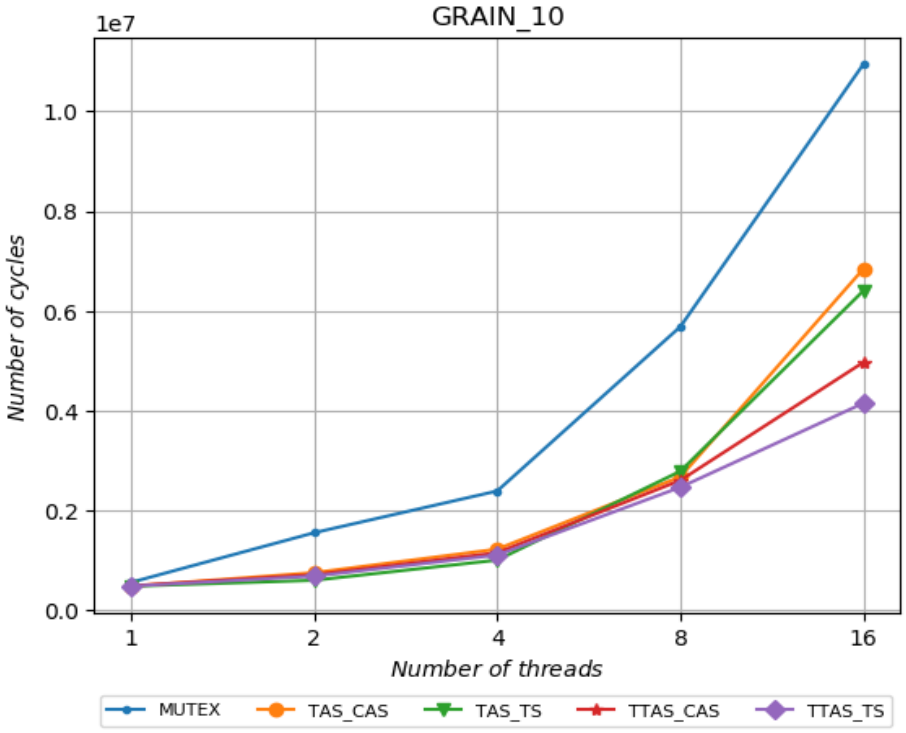
Υλοποίηση μηχανισμών συγχρονισμού

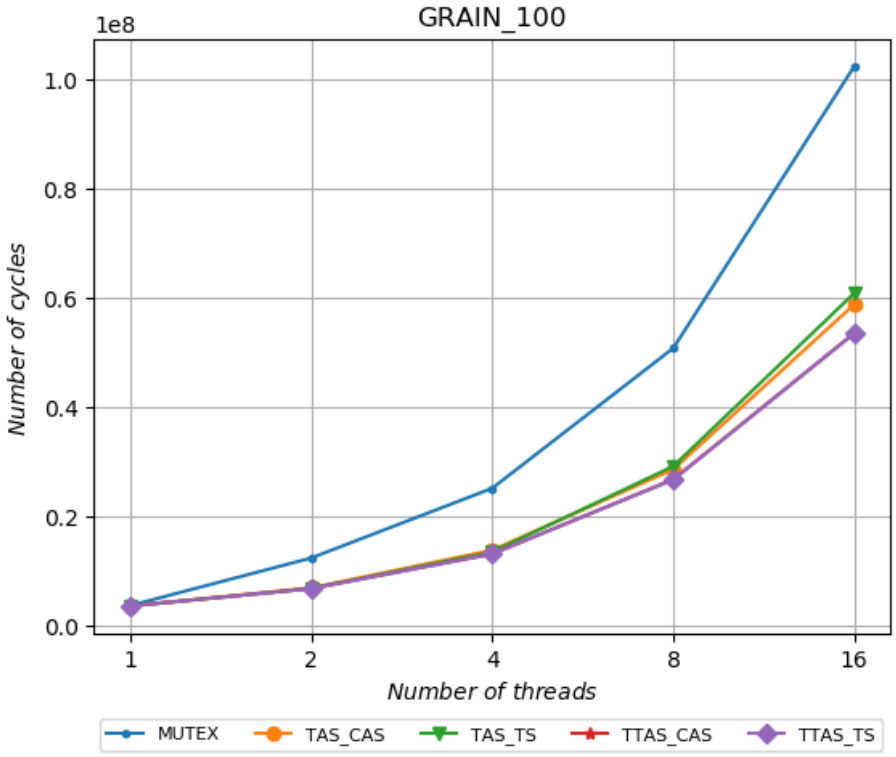
Στόχος της εργασίας είναι η αξιολόγηση των επιδόσεων διαφόρων μηχανισμών συγχρονισμού, τόσο ως προς την κλιμακωσιμότητά τους, όσο και ως προς την τοπολογία των νημάτων. Με τον όρο κλιμακωσιμότητα (scalability) εννοούμε πόσο καλά αποδίδει ένα παράλληλο πρόγραμμα καθώς αυξάνεται ο αριθμός των νημάτων από 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.

Ερώτημα 3.1.1

Για κάθε grain size παραθέτουμε το διάγραμμα της κλιμάκωσης του συνολικού χρόνου εκτέλεσης της περιοχής ενδιαφέροντος σε σχέση με τον αριθμό των νημάτων.




GRAIN_10
1e7
1.0
0.8
0.6
0.4
0.2
0.0
Number of cycles
1
2
4
8
16
Number of threads
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS

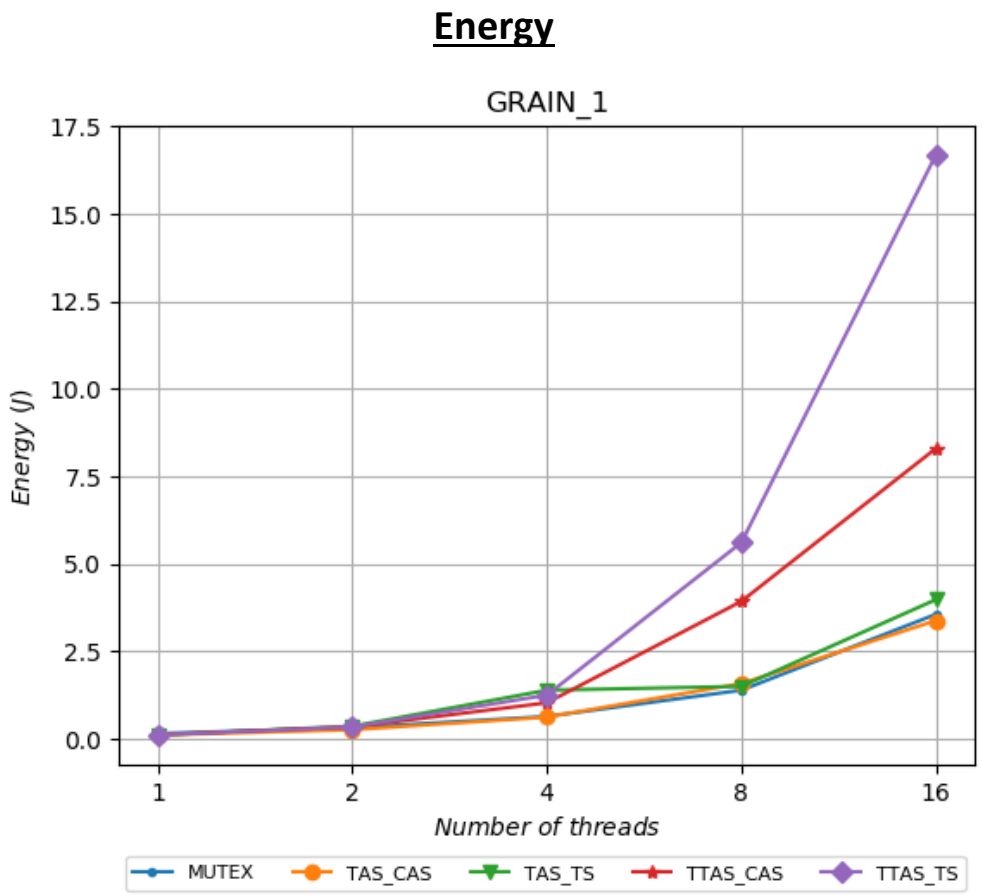

GRAIN_100
1e8
1.0
0.8
0.6
0.4
0.2
0.0
Number of cycles
1
2
4
8
16
Number of threads
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS

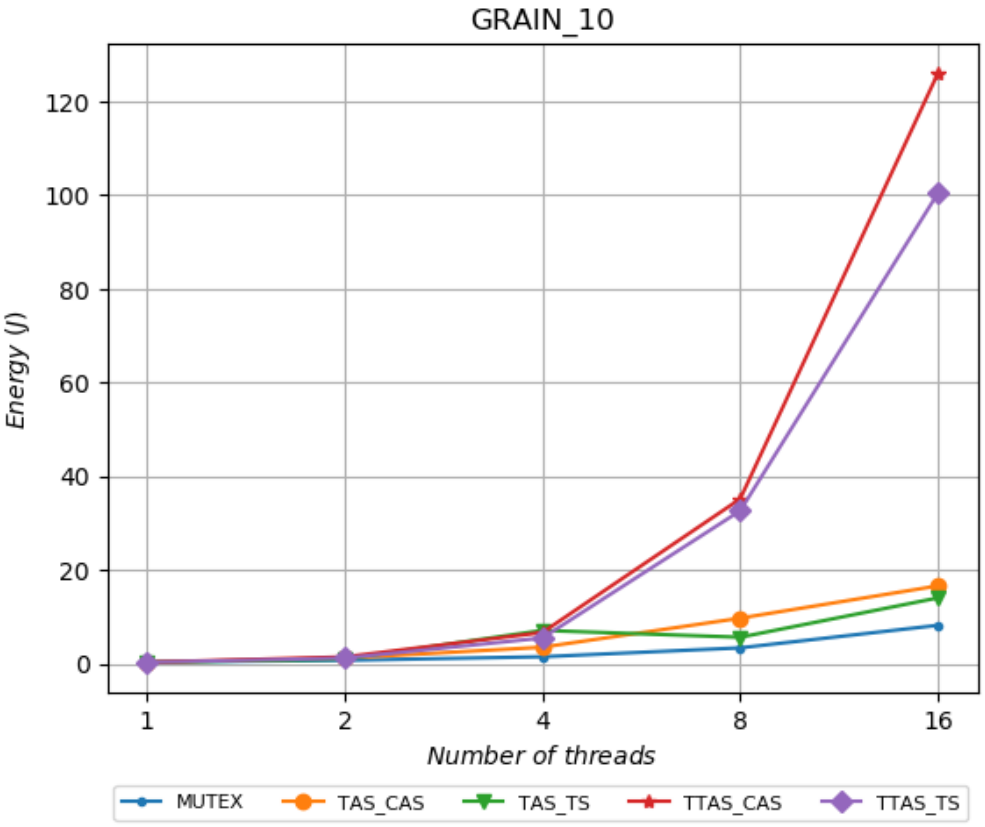
Ερώτημα 3.1.2

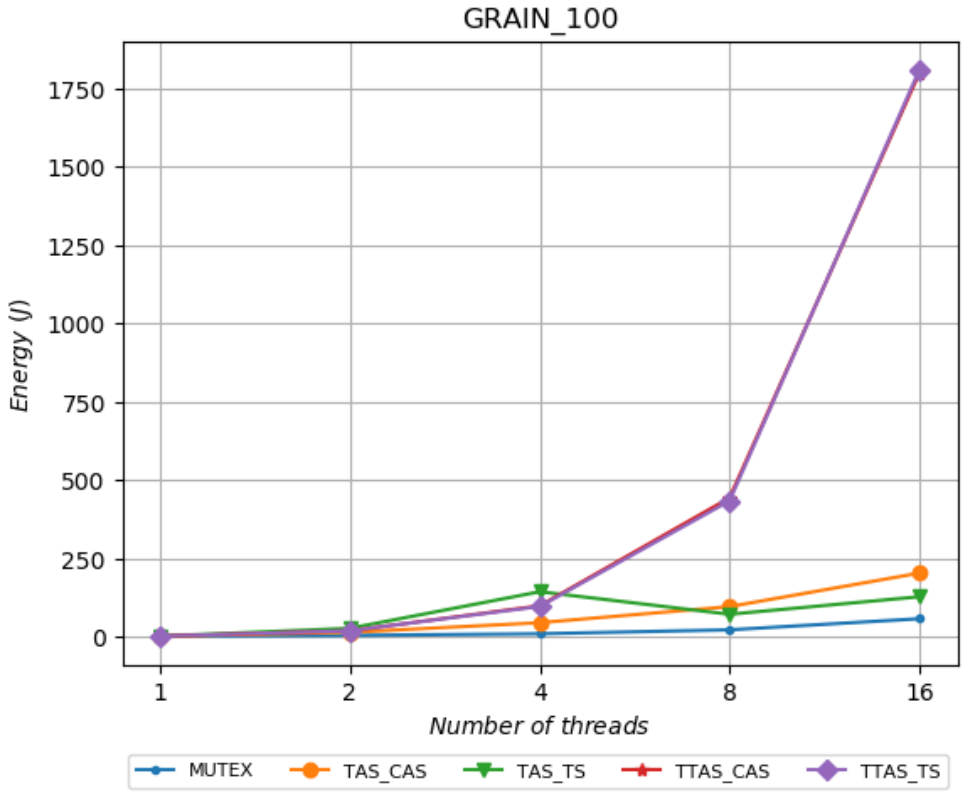
Συνολικά, παρατηρούμε πως η αύξηση του grain size προκαλεί αύξηση των απαιτούμενων κύκλων για κάθε μηχανισμό συγχρονισμού με οποιοδήποτε αριθμό νημάτων. Ωστόσο, για σταθερό grain size βλέπουμε ότι ο αριθμός των νημάτων επηρεάζει καθοριστικά την απόδοση του συστήματος. Για μικρό αριθμό νημάτων παρατηρούμε πολύ μικρή διαφορά απόδοσης μεταξύ TAS και TTAS. Όταν όμως τα νήματα αυξηθούν, τότε θα έχουμε περισσότερα από αυτά εκτός κρίσιμης περιοχής και θα προκαλούνται καθυστερήσεις, μεγαλύτερες στα TAS και αρκετά μικρότερες στα TTAS. Συμπεραίνουμε, δηλαδή, ότι τα TTAS έχουν καλύτερη απόδοση από τα TAS. Αντίστοιχα, βλέπουμε πως ο μηχανισμός MUTEX παρουσιάζει τις μεγαλύτερες καθυστερήσεις, ειδικά όσο αυξάνεται ο αριθμός των νημάτων, κάτι που οφείλεται στο φαινόμενο busy lock.

Όσον αφορά τα TAS, σε κάθε εκτέλεσή τους από ένα νήμα, εκτελούν μια εντολή store, η οποία μεταβάλλει την cache των πυρήνων που εκτελούν τα υπόλοιπα νήματα και προκαλούν περιττή κίνηση στο bus. Αντίθετα, τα TTAS παρακολουθούν μια μεταβλητή και μόνο όταν είναι ελεύθερη την λαμβάνει και εκτελεί την εντολή store. Επομένως, για κάθε store εντολή που εκτελεί ένα νήμα θα εισέρχεται στην κρίσιμη περιοχή. Αντίθετα, μια υλοποίηση που χρησιμοποιεί TAS θα κάνει πολλά store και όταν κάποια στιγμή εισέλθει στην κρίσιμη περιοχή , το εκάστοτε νήμα εκτελεί εντολές.

Ερώτημα 3.1.3

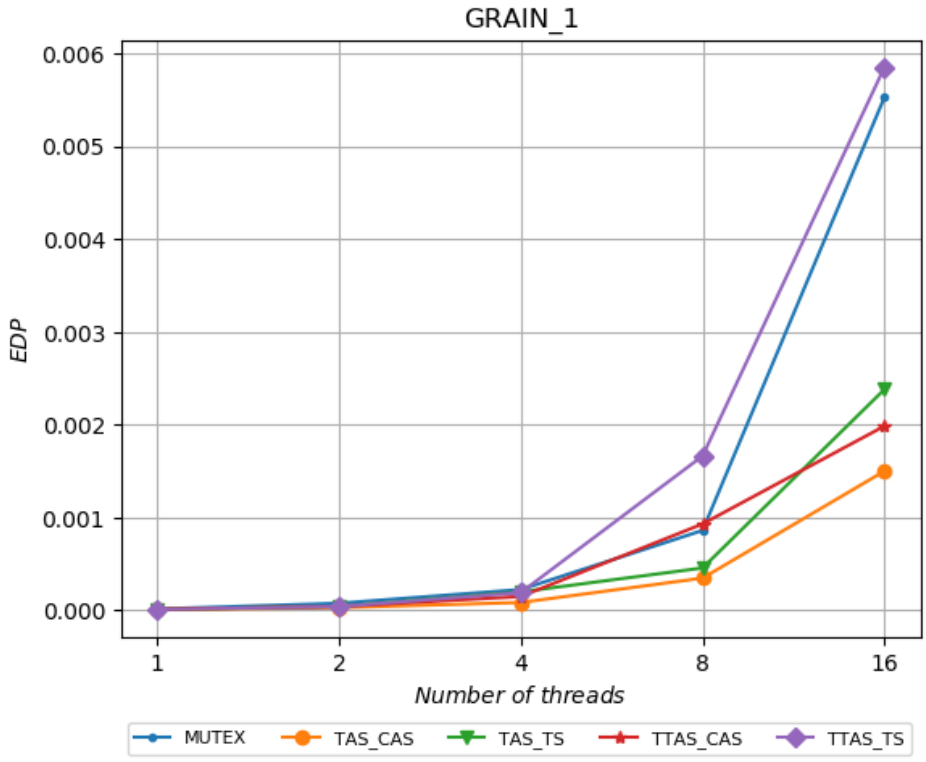


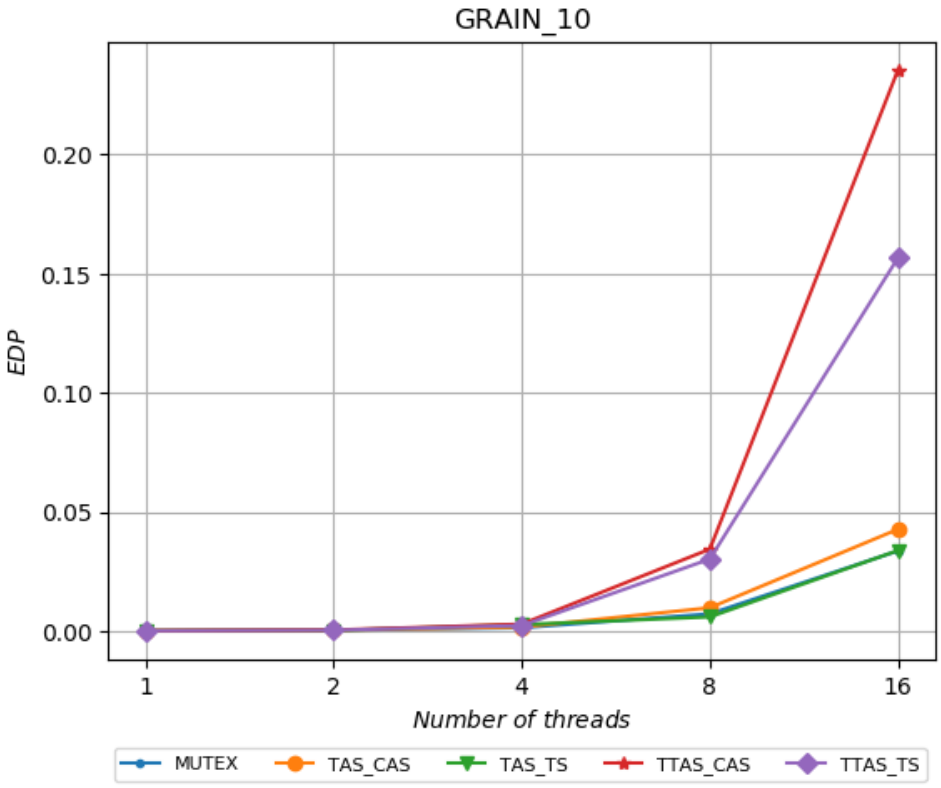

GRAIN_10
Energy (J)
120
100
80
60
40
20
0
1
2
4
8
16
Number of threads
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS

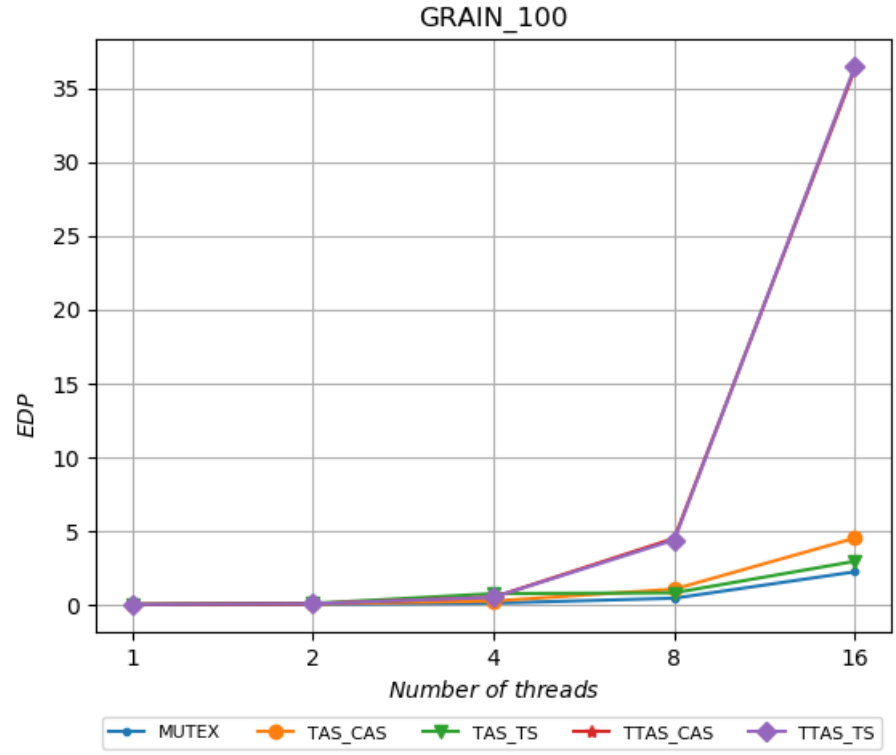

GRAIN_100
Energy (J)
1750
1500
1250
1000
750
500
250
0
1
2
4
8
16
Number of threads
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS

Όσον αφορά την κατανάλωση ενέργειας, παρατηρούμε ότι ο πιο οικονομικός μηχανισμός είναι ο Mutex ακόμα και αν είναι ο πιο αργός. Πολύ κοντά βρίσκονται και οι δύο μηχανισμοί TAS, με τους TTAS να εμφανίζουν μακράν τη περισσότερη κατανάλωση ενέργειας, κάτι που οφείλεται στο γεγονός πως το busy wait που κάνει με την πρόσθετη συνθήκη, απασχολεί περισσότερο τον επεξεργαστή. Επίσης, γίνεται αντιληπτό πως η αύξηση της κατανάλωση ενέργειας είναι ανάλογη της αύξησης του grain size, δηλαδή όταν δεκαπλασιάζεται το grain size αντίστοιχα δεκαπλασιάζεται και η κατανάλωση ενέργειας από κάθε μηχανισμό. Τέλος, όσον αφορά τον αριθμό των νημάτων, είναι λογικό πως όσο περισσότερα χρησιμοποιεί ένας επεξεργαστής τόσο μεγαλύτερη θα είναι η απαιτούμενη ενέργεια.

EDP

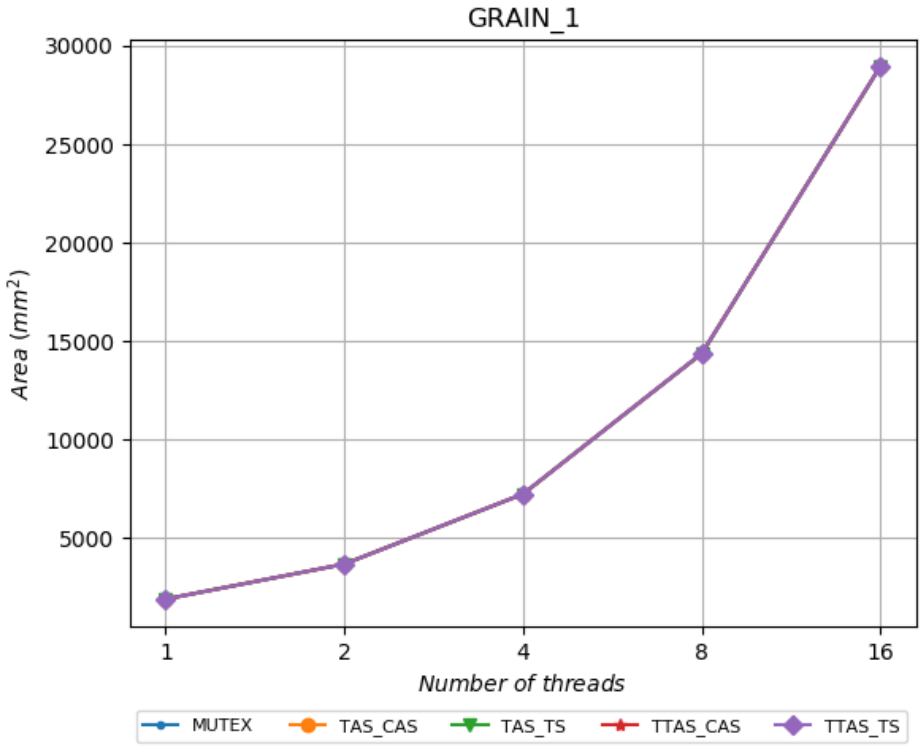


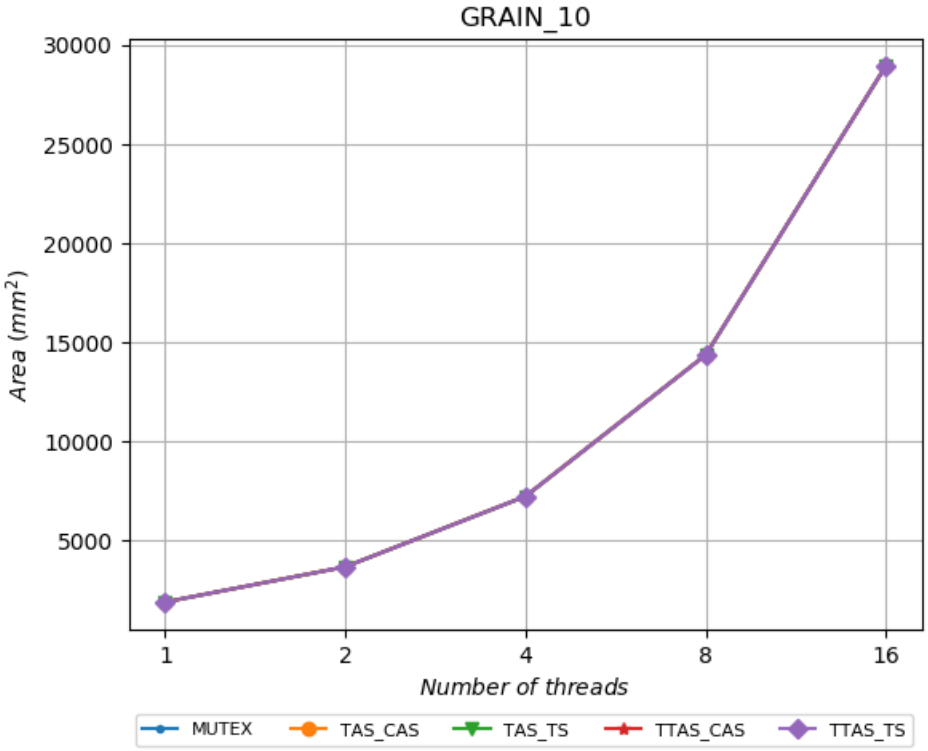


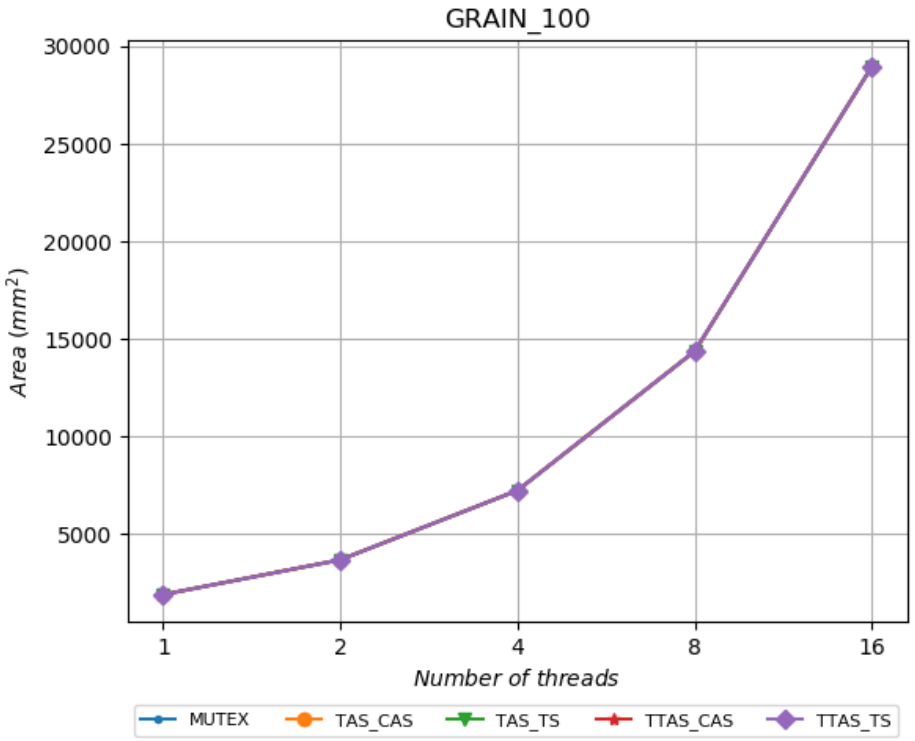


Μελετώντας το EDP που συσχετίζει τη κατανάλωση ενέργειας με τον συνολικό χρόνο εκτέλεσης, καταλήγουμε στα ίδια συμπεράσματα όπως και στην απλή κατανάλωση ενέργειας. Η μόνη διαφορά που παρατηρούμε είναι πως πλέον ο μηχανισμός Mutex δεν είναι ο πιο “οικονομικός”, καθώς όπως είδαμε στο πρώτο ερώτημα απαιτεί με διαφορά τον μεγαλύτερο χρόνο εκτέλεσης. Έτσι, καταλήγει να είναι το ίδιο μη αποδοτικός με τον TTAS, τουλάχιστον για grain size = 1. Ακόμη, βλέπουμε πως δεν υπάρχει μεγάλη αλληλουχία μεταξύ των διαφορετικών τιμών του grain size, όπως προηγουμένως, καθώς για κάθε τιμήν του οι μηχανισμοί TTAS και Mutex δεν εμφανίζουν ανάλογη απόδοση. Το μόνο σίγουρο είναι πως οι μηχανισμοί TAS είναι σταθερά οι πιο αποδοτικοί.

AREA

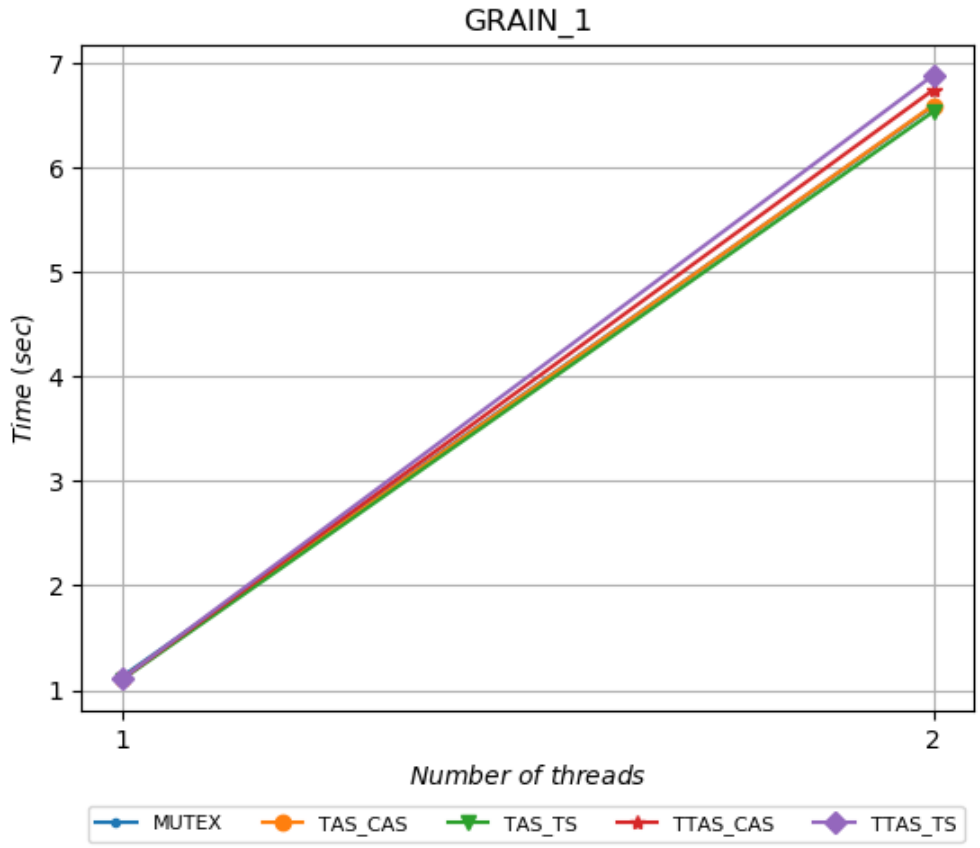


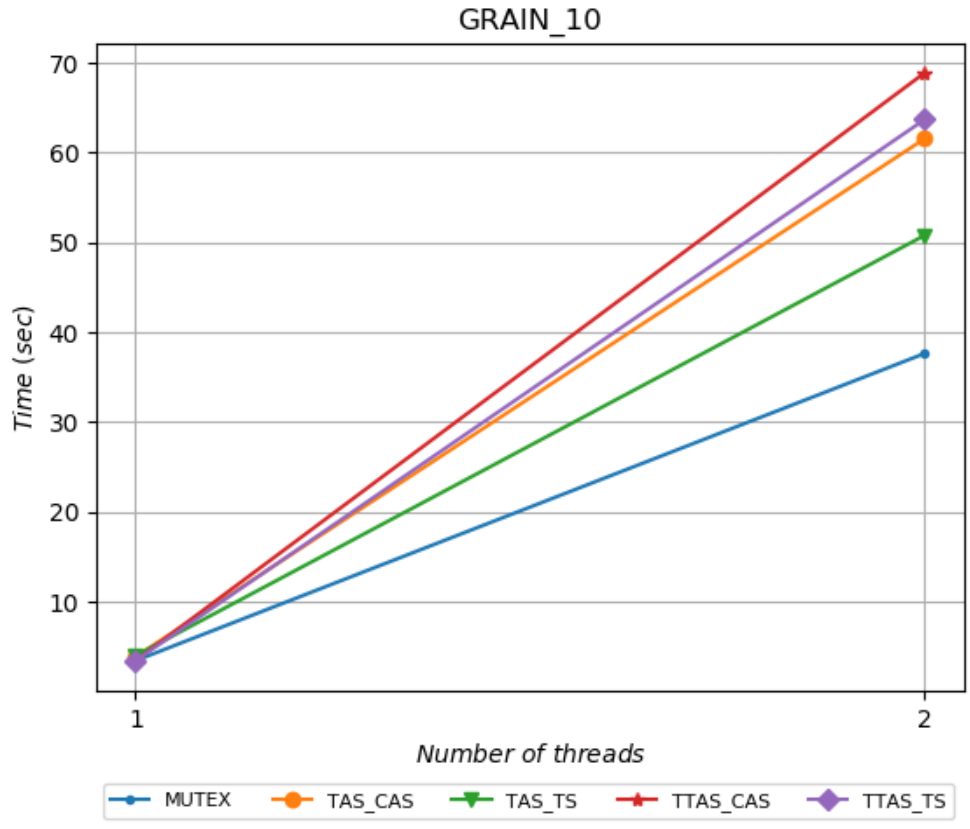


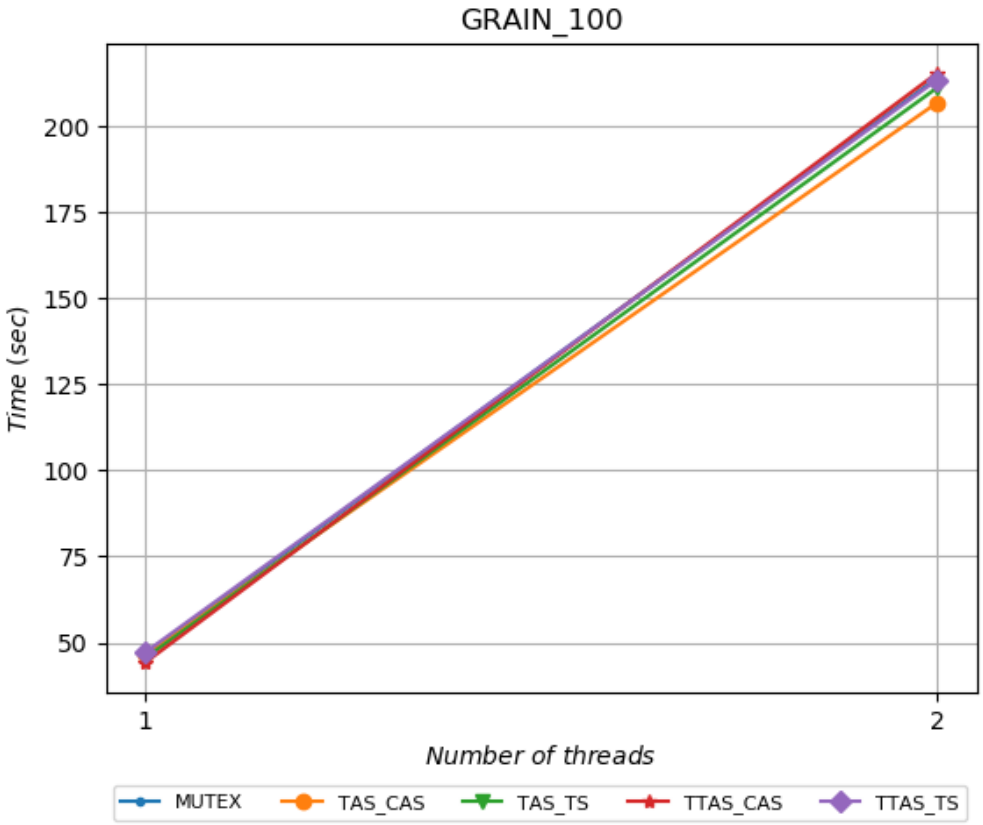


Όσον αφορά το μέγεθος του chip, παρατηρούμε πως είναι σταθερό για κάθε έναν μηχανισμό αλλά και για τις διάφορες τιμές του grain size. Επίσης, είναι λογικό όσο αυξάνεται ο αριθμός των νημάτων να αυξάνεται το μέγεθος του chip, ωστόσο οι τιμές που παρατηρούμε είναι αρκετά μεγαλύτερες από το αναμενόμενο. Για 1 πυρήνα το μέγεθος καθορίζεται στα 1869mm^2. Αυτό οφείλεται στη τιμή dispatch_width = 64 που χρησιμοποιούμε στην προσομοίωση, καθώς όπως είδαμε στη προηγούμενη σειρά ασκήσεων, ελάχιστο μέγεθος επιτυγχάνουμε για τιμές 2-4. Επομένως, δεν έχει να κάνει με λανθασμένα αποτελέσματα της προσομοίωσης, αλλά με μη ιδανική αρχιτεκτονική των πυρήνων.

Ερώτημα 3.1.4

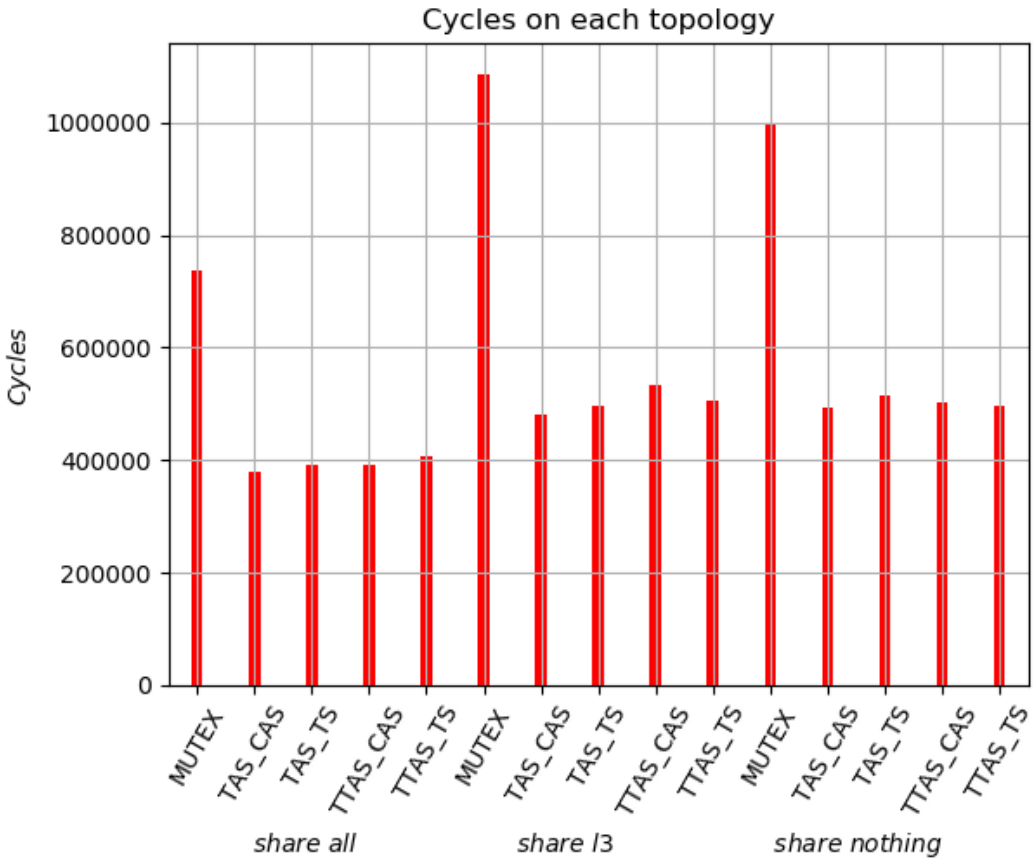

GRAIN_1
Time (sec)
Number of threads
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS

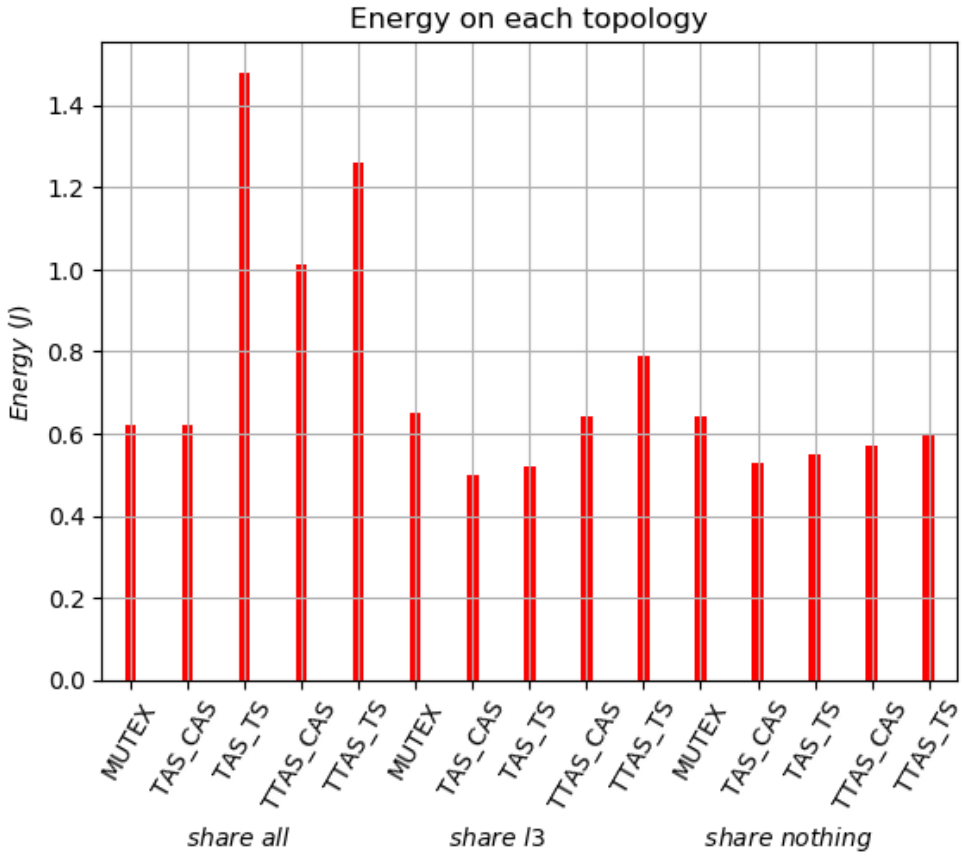

GRAIN_10
Time (sec)
Number of threads
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS

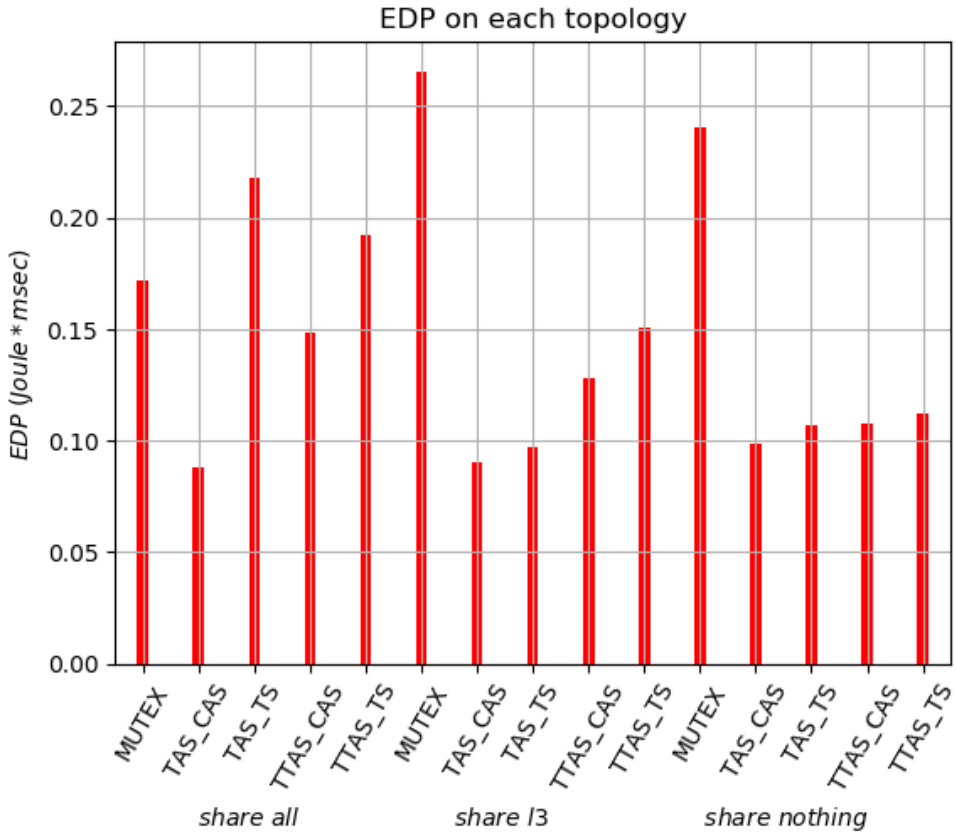


Σε αυτό το ερώτημα πραγματοποιήσαμε τις ίδιες μετρήσεις με το πρώτο ερώτημα, αλλά αυτή τη φορά σε πραγματικό σύστημα δύο φυσικών πυρήνων. Παρατηρούμε, λοιπόν, ότι όσο αυξάνεται το grain size αυξάνεται σημαντικά και ο χρόνος εκτέλεσης, όπως δηλαδή συνέβη και στην προσομοίωση. Επίσης, έχοντας μόνο δύο φυσικούς πυρήνες, μπορούμε εύκολα να διαπιστώσουμε την απότομη αύξηση του χρόνου εκτέλεσης όταν αυξάνεται ο αριθμός των νημάτων. Παράλληλα, βλέπουμε πως δεν υπάρχει κάποια ορατή διαφορά μεταξύ των διαφορετικών μηχανισμών συγχρονισμού για grain size = 1 και 100. Ωστόσο, μια διαφορά σε σχέση με τα αποτελέσματα της προσομοίωσης είναι πως για grain size = 10 ο μηχανισμός Mutex φαίνεται να είναι γρηγορότερος από τους TAS και TTAS.

Ερώτημα 3.2.1


Cycles on each topology
Cycles
1000000
800000
600000
400000
200000
0
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS
share all
share l3
share nothing


Energy on each topology
Energy (J)
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS
share all
share l3
share nothing



Παρατηρούμε ότι η τοπολογία share-all είναι αρκετά πιο γρήγορη από τις άλλες δύο, καθώς σε αυτές πρέπει κάθε φορά να γίνει μεταφορά του block του lock μεταξύ των μνημών cache των επεξεργαστών. Στην τοπολογία share-nothing, πρέπει να υπάρχει επικοινωνία μεταξύ της κύριας μνήμης κάτι που εισάγει περισσότερες καθυστερήσεις από ότι η επικοινωνία εντός κάποιου επιπέδου κρυφής μνήμης.

Επίσης, μεταξύ των μηχανισμών συγχρονισμού, είναι φανερό πως ο mutex είναι μακράν ο πιο αργός από τους υπόλοιπους, ένα συμπέρασμα στο οποίο καταλήξαμε και στο ερώτημα 3.1.1 .

Όσον αφορά την αυτή καθ’ αυτή κατανάλωση ενέργειας αλλά και το EDP, παρατηρούμε πως η τοπολογία share-all εμφανίζει την μεγαλύτερη κατανάλωση ενέργειας, η οποία σε κάποιους μηχανισμούς είναι έως και διπλάσια σε σχέση με τις άλλες δύο τοπολογίες. Ωστόσο, φαίνεται πως ο παράγοντας χρόνος μεταβάλλει τα αποτελέσματα όταν παρατηρούμε το διάγραμμα του EDP, καθώς οι διάφορες τοπολογίες συγκλίνουν λίγο περισσότερο μεταξύ τους, με τον μηχανισμό mutex να δείχνει πόσο πολύ επηρεάζει ο μεγάλος χρόνος εκτέλεσης το EDP. Πάντως, και σε αυτό το διάγραμμα, οι μηχανισμοί συγχρονισμού καταλαμβάνουν την ίδια σειρά όπως και στο διάγραμμα του χρόνου εκτέλεσης.

Μέρος Β

	INST	IS	EX	WR	CMT	Σχόλια
0	LOOP: LD F0, 0(R1)	1	2-5	6	7	MISS, fetch A[0]-A[1], LRU=1
1	ADDD F4, F4, F0	1	7-9	10	11	RAW F0
2	LD F1,0(R2)	2	3-6	7	11	MISS, fetch B[0]-B[1], LRU=0
3	MULD F4,F4,F1	2	11-15	16	17	RAW F4, F1
4	ANDI R9,R8,0x2	3	4-5	8	17	CDB conflict
5	BNEZ R9,NEXT	3	9-10	11	18	pred = T, act = NT, RAW R9
9	NEXT: LD F5,8(R1)	7	8	9	--	LOAD queue FULL, HIT
10	ADDD F4, F4, F5	7	--	--	--	FU busy, RAW F4,F5
11	ADDI R1,R1,0x8	8	9-10	--	--	
12	SUBI R8,R8,0X1	9	10-11	--	--	RS FULL, flush @11
6	IF: LD F2,16(R2)	12	13-16	17	18	LOAD queue FULL, MISS, fetch B[2]-B[3]
7	MULD F2,F2,F5	12	18-22	23	24	FU busy, RAW F2
8	ADDD F4,F4,F2	13	24-26	27	28	RAW F2,F4
9	NEXT: LD F5,8(R1)	13	14	15	28	HIT
10	ADDD F4, F4, F5	14	28-30	31	32	FU busy, RAW F4,F5
11	ADDI R1,R1,0x8	14	15-16	18	32	CDB conflict
12	SUBI R8,R8,0X1	18	19-20	21	33	ROB FULL
13	BNEZ R8,LOOP	18	22-23	24	33	pred = T, act = NT, RAW R8
0	LOOP: LD F0, 0(R1)	19	20	22	--	HIT, CDB conflict
1	ADDD F4, F4, F0	19	--	--	--	FU busy, RAW F0,F4 , flush @24
14	SD F4, 8(R2)	25	32-35	36	37	MISS, fetch B[0]-B[1], LRU=0,RAW F4

Τελικά περιεχόμενα της cache: **A[0]-[1], B[0]-[1]**