Προηγμένα Αρχιτεκτονικής Υπολογιστών

4η Εργαστηριακή άσκηση
Μάνος Αραπίδης
ΑΜ: 03116071

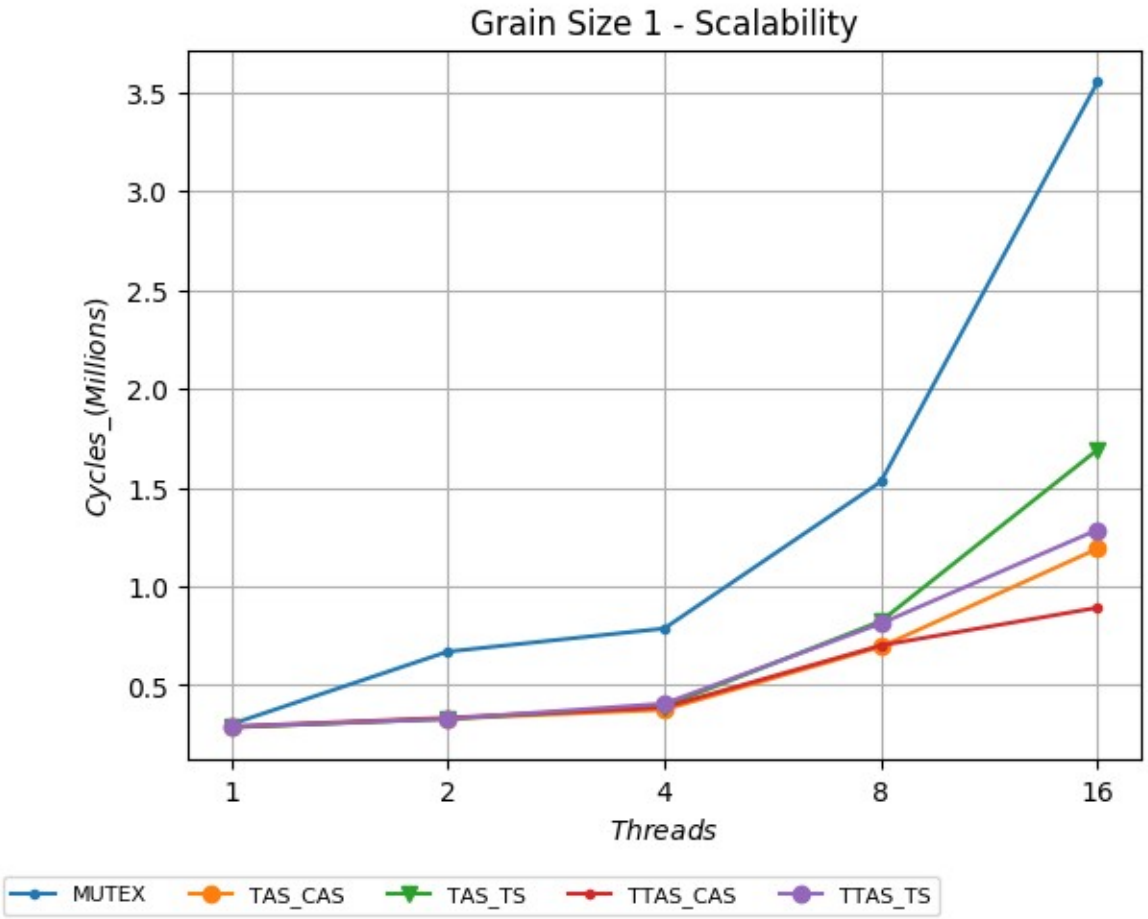
Μέρος Α

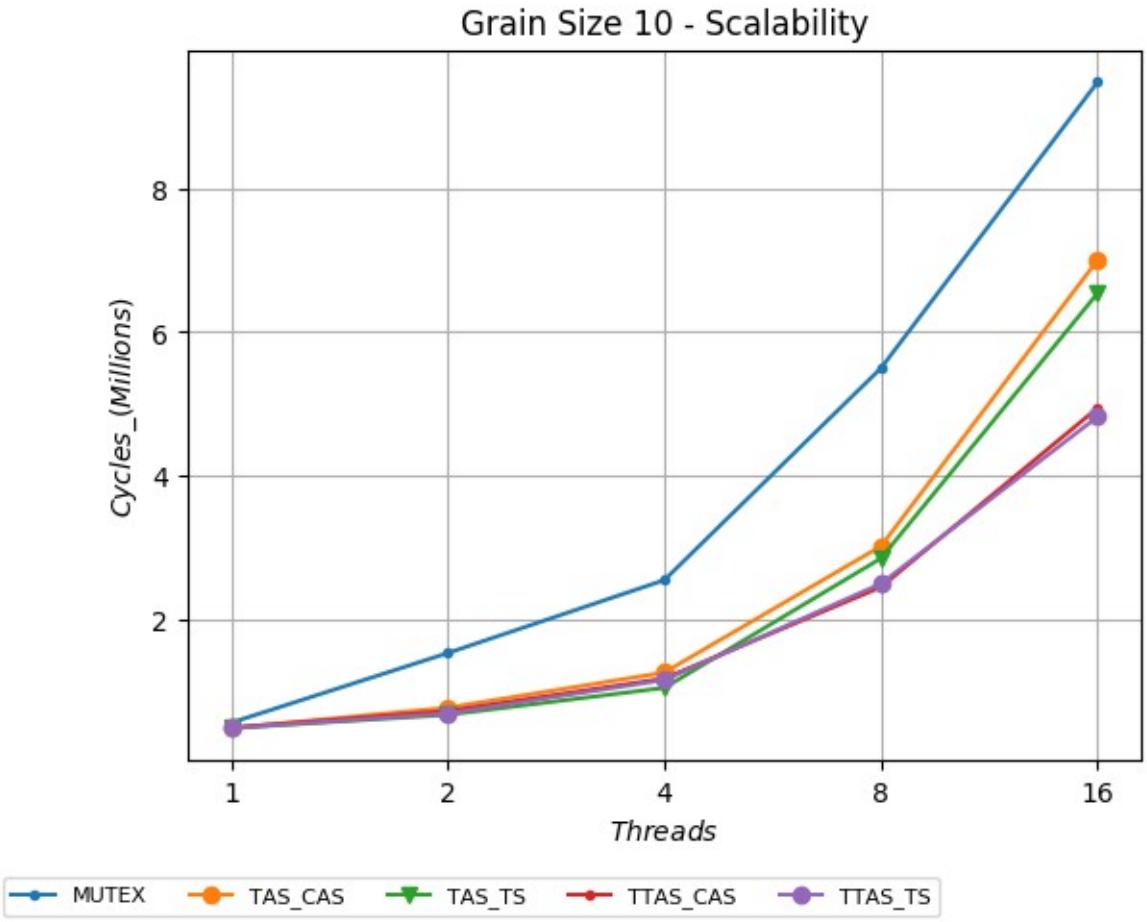
i.

Σκοπός αυτού του ερωτήματος είναι να δώσουμε το διάγραμμα κλιμάκωσης της περιοχής ενδιαφέροντος σε σχέση με τον αριθμό νημάτων . Πιο συγκεκριμένα, θα εξετάσουμε τις εκδόσεις του προγράμματος για : TAS_CAS, TAS_TS, TTAS_CAS, TTAS_TS, MUTEX και για κάθε grain size : 1, 10 και 100 .

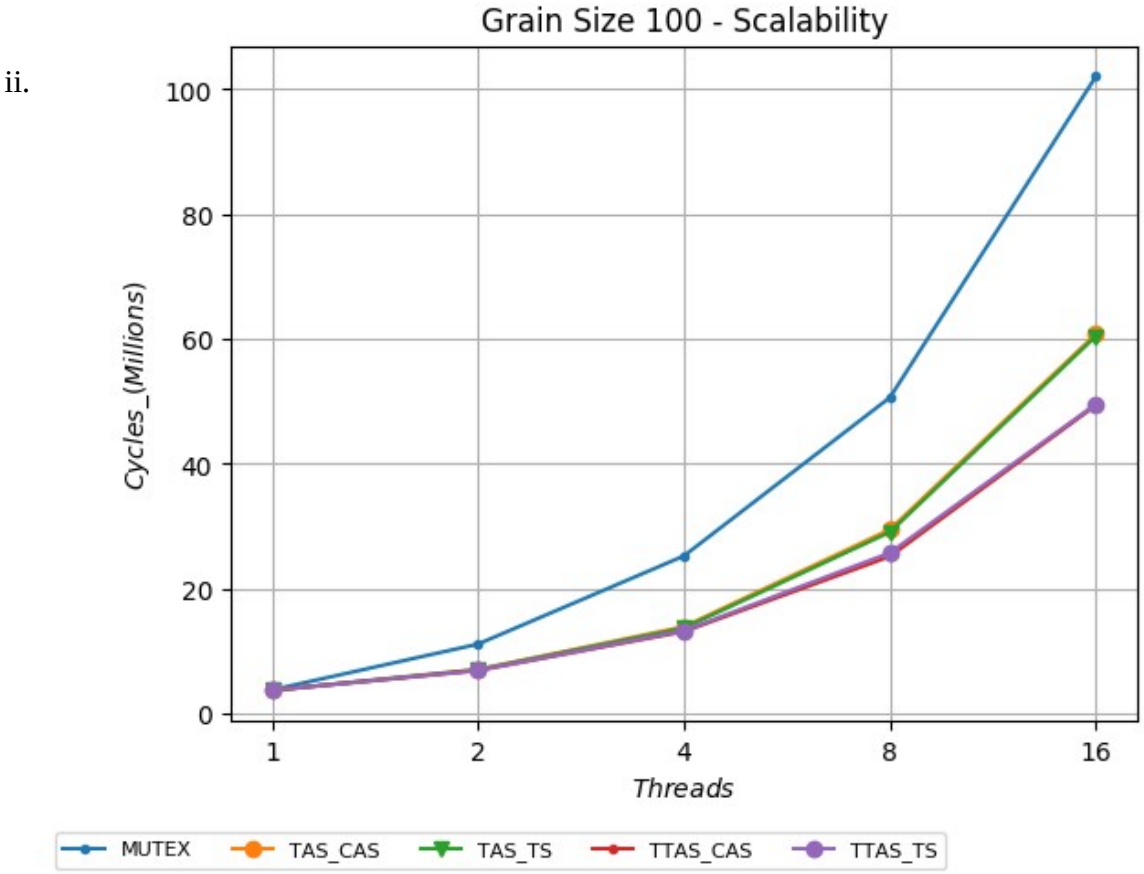
Ως μετρική θα χρησιμοποιήσουμε τον χρόνο εκτέλεσης σε αριθμό κύκλων.

Ακολουθούν διαγράμματα για κάθε grain size .




Grain Size 10 - Scalability
Cycles_(Millions)
8
6
4
2
1
2
4
8
16
Threads
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS

ii.


Grain Size 100 - Scalability
Cycles_(Millions)
100
80
60
40
20
0
1
2
4
8
16
Threads
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS

Σκοπός αυτού του ερωτήματος είναι να αναλύσουμε την κλιμάκωση του χρόνου εκτέλεσης σε σχέση με την φύση της κάθε υλοποίησης καθώς και με το grain size .

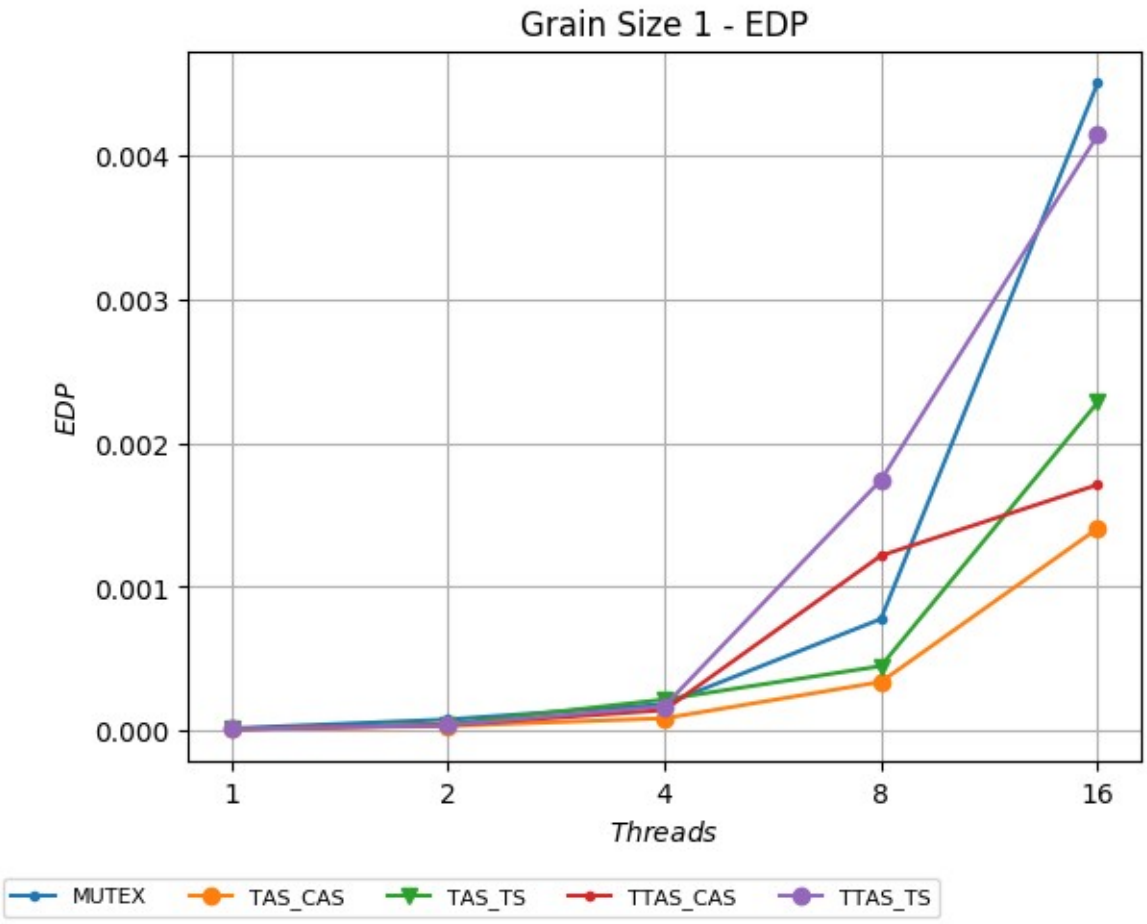
Από τις τρείς γραφικές παρατηρούμε ότι την χειρότερη απόδοση την παρουσιάζει το κλείδωμα MUTEX . Όταν ένα νήμα δεν μπορεί να έχει πρόσβαση στην κρίσιμη περιοχή ο χρονοπρογραμματιστής το βάζει σε κατάσταση sleep και το επιστρέφει σε running, όταν είναι διαθέσιμη . Ωστόσο είναι πιθανό η εναλλαγή αυτή να απαιτεί περισσότερη χρόνο από ότι βρίσκεται το νήμα στην κρίσιμη και έτσι ένα μεγάλο του διαθέσιμου χρόνου του το “σπαταλάει” σε contex switch . Σε αντίθεση οι υλοποιήσεις TAS και TTAS προσπαθούν συνέχεια να εισέλθουν στην κρίσιμη περιοχή και ειδικά σε εφαρμογές σαν την δική μας ,όπου είναι cpu intensive, πετυχαίνουμε καλύτερο επεξεργαστικό throughput .

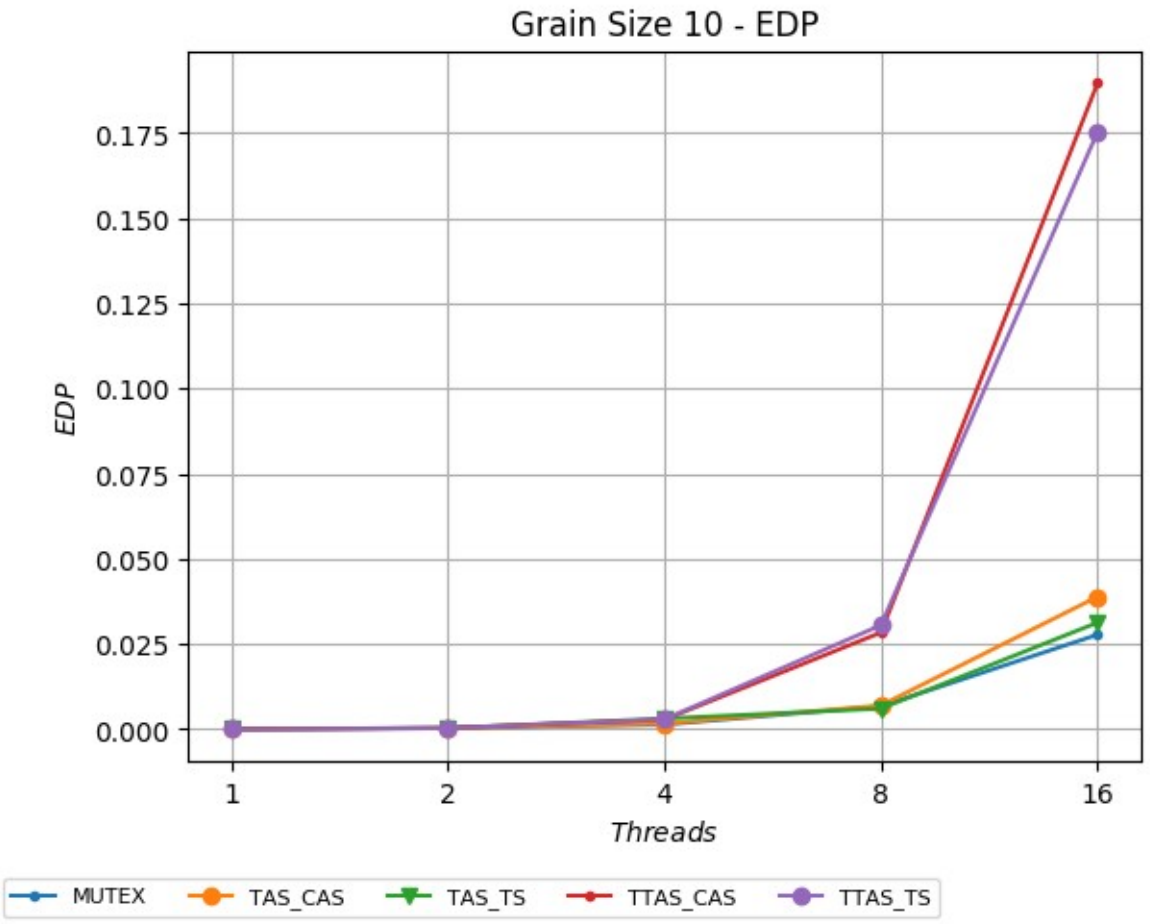
Στη συνέχεια παρατηρούμε ότι η TTAS παρουσιάζει καλύτερη απόδοση σε σχέση με την TAS όσο αυξάνεται ο αριθμός των νημάτων. Αυτό είναι αναμενόμενο καθώς στην TAS τα νήματα κάνουν invalidate την cache line και γράφουν σε αυτήν ανεξαρτήτως αν έχουν αποκτήσει πρόσβαση στην κρίσιμη περιοχή . Αντίθετα, η TTAS διαβάζει το κλειδί και επιχειρεί να γράψει σε αυτό μόνο όταν μπορεί να έχει πρόσβαση . Έτσι, αποφεύγουμε τις καθυστερήσεις που προκαλούνται από την διαδικασία συνάφειας της cache στους πυρήνες . Τέλος, δεν παρατηρούμε σημαντικές διαφορές μεταξύ των μεθόδων TS και CAS .

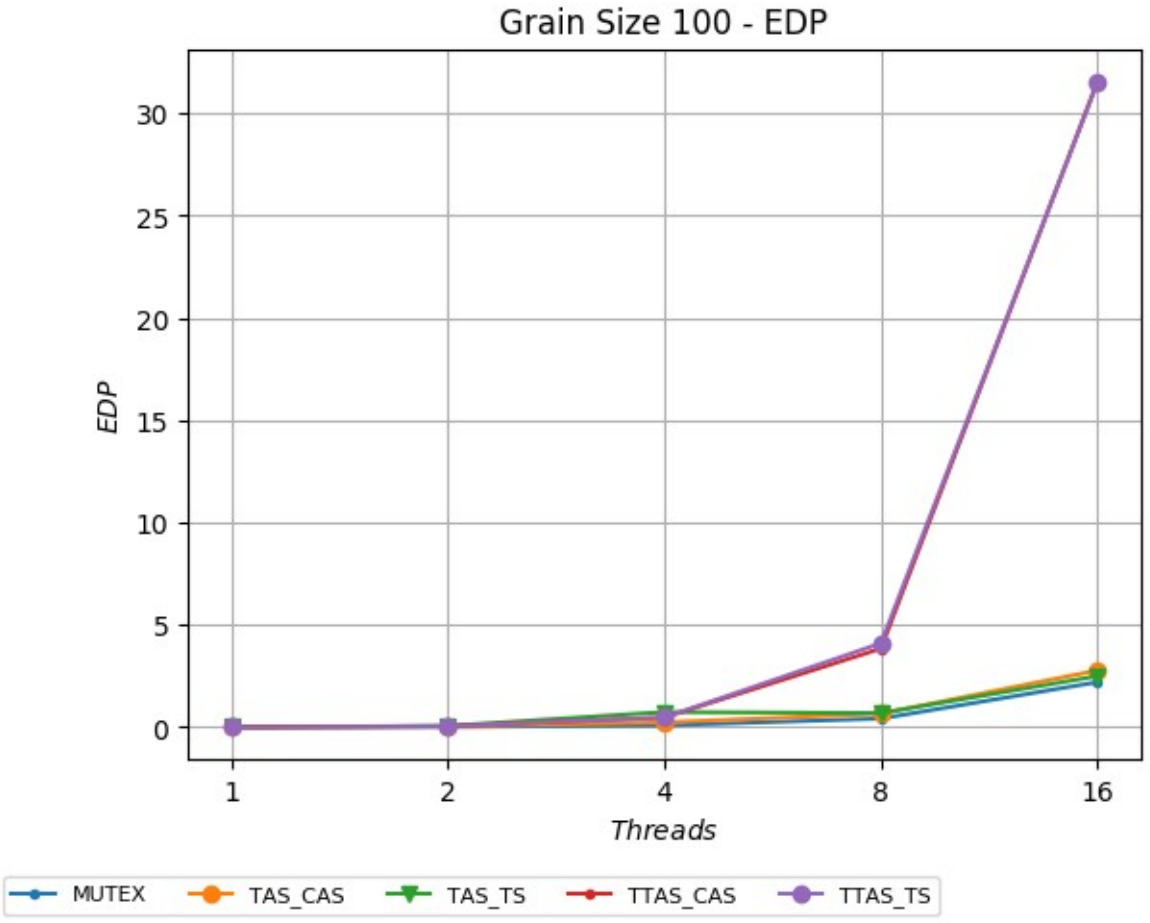
Σχετικά με το grain size, όσο αυξάνεται τόσο αυξάνονται και οι κύκλοι, καθώς κάθε κρίσιμη περιοχή έχει μεγαλύτερο όγκο “εργασίας” . Ακόμη παρατηρούμε, ότι καθώς αυξάνεται ο αριθμός των νημάτων-πυρήνων έχουμε την ίδια αύξηση και στους κύκλους, καθώς έχουμε μεγαλύτερες καθυστερήσεις, λόγω της ανάγκης συγχρονισμού περισσότερων νημάτων και της πολυπλοκότητας των μεθόδων συνάφειας μνήμης cache .

iii.

Σ’ αυτό το ερώτημα μας ζητήθηκε να συμπεριλάβουμε στην ανάλυση μας την κατανάλωση ενέργειας σχετικά με την κλιμάκωση. Ακολουθούν διαγράμματα για κάθε υλοποίηση και grain size και χρησιμοποιούμε ως μετρική την EDP .


Grain Size 1 - EDP
EDP
0.004
0.003
0.002
0.001
0.000
1
2
4
8
16
Threads
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS


Grain Size 10 - EDP
EDP
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
1
2
4
8
16
Threads
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS


Grain Size 100 - EDP
EDP
30
25
20
15
10
5
0
1
2
4
8
16
Threads
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS

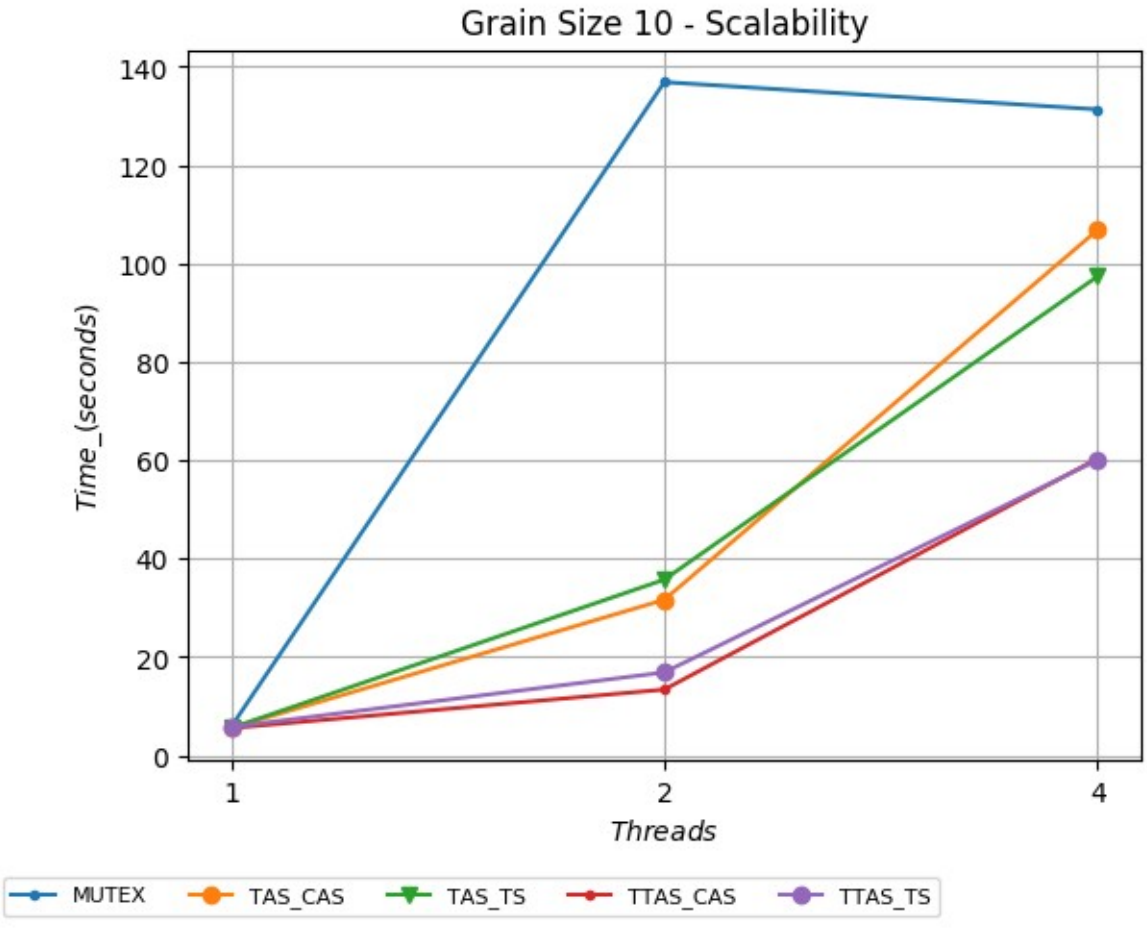
Αρχικά παρατηρούμε ότι η κατανάλωση ενέργειας για αριθμό νημάτων <= 4 παραμένει σταθερή για όλες τις υλοποιήσεις . Μετά από αυτό το σημείο οι γραφικές αρχίζουν να διαφοροποιούνται αλλά παρουσιάζουν κοινή συμπεριφορά, με την αύξηση των νημάτων-πυρήνων να αυξάνεται και η κατανάλωση.

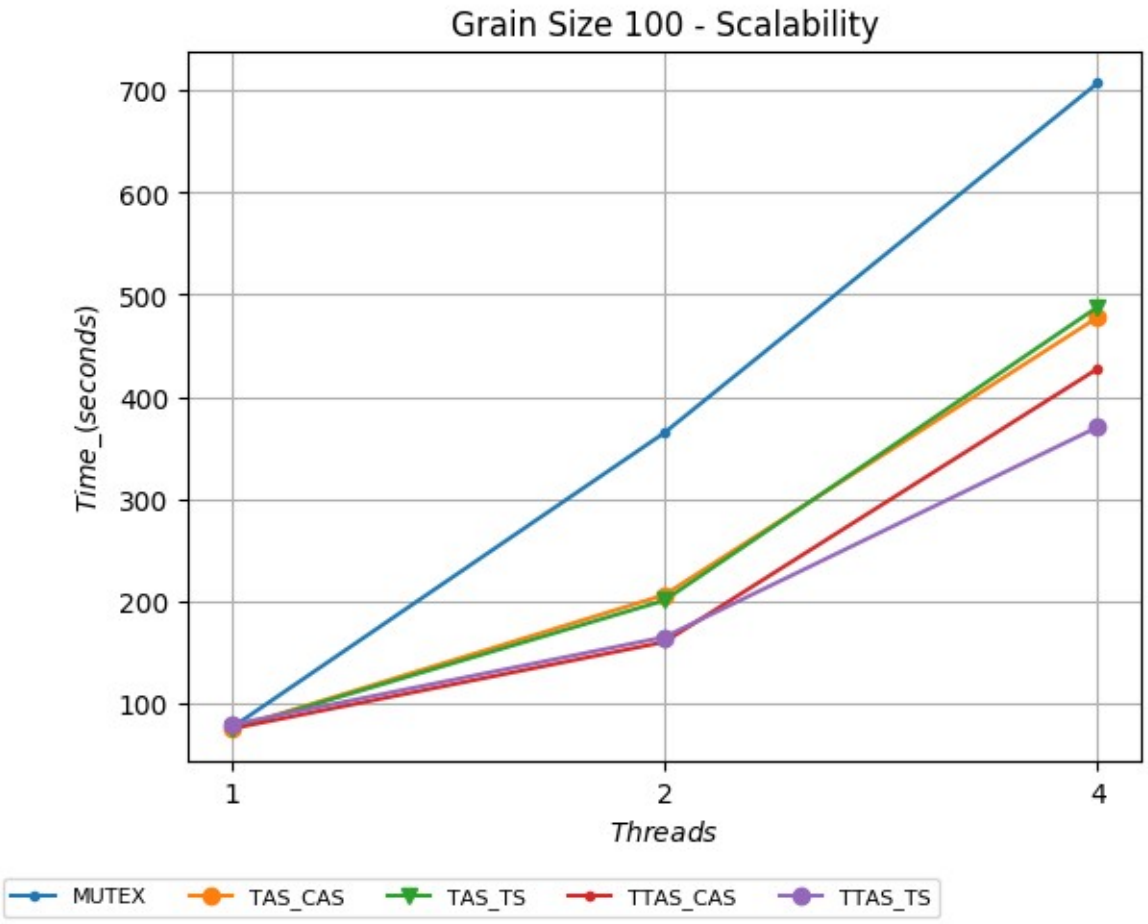
Σε σχέση με τις μεθόδους συγχρονισμού, παρατηρούμε ότι οι “παραλλαγές” της TTAS έχουν την μεγαλύτερη κατανάλωση, με τιμή πολλαπλάσια των υπολοίπων κάτι που γίνεται καλύτερα αντιληπτό για grain size : 10 και 100. Την χαμηλότερη τιμή φαίνεται να έχει το κλείδωμα MUTEX, το οποίο είναι λογικό καθώς δεν απασχολεί συνέχεια τους πόρους του συστήματος αλλά περιμένει μέχρι να ελευθερωθεί η κρίσιμη περιοχή . Έπειτα, ακολουθεί με πρακτικά ίδιες τιμές η μέθοδος TAS. Τέλος, παρατηρούμε ότι για TS και CAS έχουμε την ίδια κατανάλωση ενέργειας, πέρα του grain size :1 όπου οι τιμές είναι τόσο μικρές και οι διαφορές τους θα μπορούσαν να θεωρηθούν αμελητέες .

iv.

Σ’ αυτό το ερώτημα μας ζητήθηκε να εκτελέσουμε το ερώτημα 1 στο δικό μας σύστημα. Στην προκειμένη περίπτωση, ο προσωπικός μου υπολογιστής διαθέτει 4 πυρήνες χωρίς Hyper Threading, οπότε ο μέγιστος αριθμός νημάτων που μπορεί να υποστηρίξει είναι 4 . Ακολουθούν διαγράμματα για τις υλοποιήσεις μέχρι 4 πυρήνες και με μετρική τον χρόνο εκτέλεσης.


Grain Size 1 - Scalability
Time_(seconds)
Threads
0
10
20
30
40
50
60
1
2
4
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS

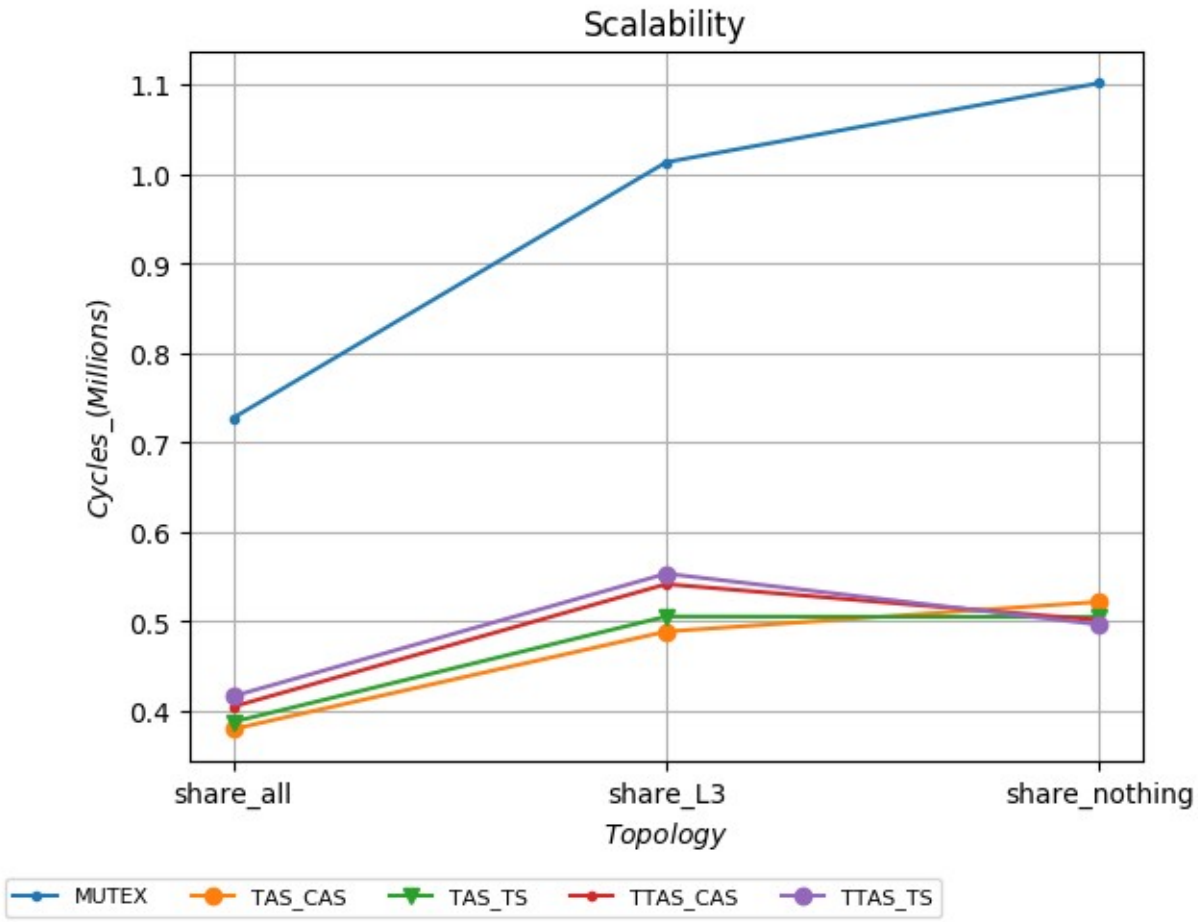

Grain Size 10 - Scalability
Time_(seconds)
Threads
0
20
40
60
80
100
120
140
1
2
4
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS

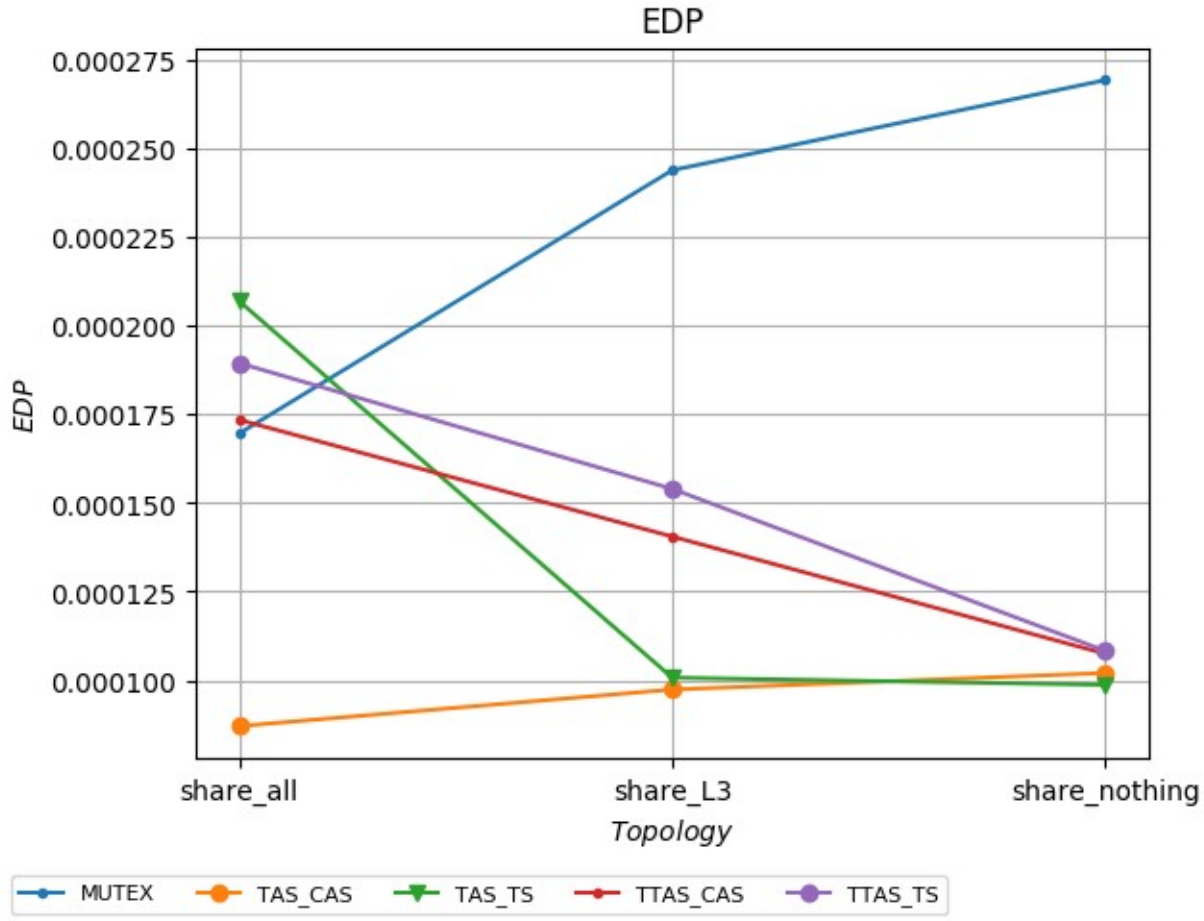


Συνοψίζοντας, μπορούμε να καταλήξουμε στο συμπέρασμα, ότι τα αποτελέσματα που πήραμε από το πραγματικό σύστημα, δεν απέχουν και πολύ από τα αποτελέσματα που πήραμε από τις προσομοιώσεις. Εξαίρεση σε αυτό, είναι, το MUTEX, που για Grain size: 1 δεν έχει την υψηλότερη καθυστέρηση.

v.

Σε αυτό το ερώτημα, αναλύουμε την επίδραση όπου έχει ο βαθμός και ο τρόπος που μοιράζεται η cache στην κλιμάκωση του χρόνου εκτέλεσης και στην κατανάλωση ενέργειας . Συγκεκριμένα, θα δούμε τις τοπολογίες : share all , share L3 και share nothing. Ακολουθούν διαγράμματα για τις παραπάνω περιπτώσεις, με μετρικές τον συνολικό πλήθος κύκλων εκτέλεσης της κρίσιμης περιοχής και την EDP .


Scalability
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
Cycles_(Millions)
share_all
share_L3
share_nothing
Topology
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS


EDP
0.000275
0.000250
0.000225
0.000200
0.000175
0.000150
0.000125
0.000100
EDP
share_all
share_L3
share_nothing
Topology
MUTEX
TAS_CAS
TAS_TS
TTAS_CAS
TTAS_TS

Αρχικά, παρατηρούμε ότι πάλι το MUTEX έχει τον χειρότερο χρόνο εκτέλεσης . Οι μέθοδοι TAS και TTAS παρουσιάζουν παρόμοια αποτελέσματα, με την πρώτη να αποδίδει ελαφρώς καλύτερα . Συγκριτικά, με τις τοπολογίες βλέπουμε ότι στην γενική περίπτωση ο μεγαλύτερος βαθμός διαμοιρασμού φέρνει και καλύτερο χρόνο εκτέλεσης. Πιο συγκεκριμένα, η share all μείωνει τις καθυστερήσεις από τις μεθόδους συνάφειας της cache. Αυτή η ανάγκη υπάρχει μόνο για το επίπεδο L1, καθώς τα υπόλοιπα L2 και L3 διαμοιράζονται μεταξύ των πυρήνων. Από την αντίθετη μεριά , στην share nothing κάθε αλλαγή οφείλει να ενημερώνει την διαφορετική cache όλων των πυρήνων, στην περίπτωση που διαθέτει παλιά δεδομένα . Παρόλο αυτά, παρατηρούμε ότι για την μέθοδο TAS η απόδοση παραμένει σταθερή μεταξύ των share L3, share nothing και στην περίπτωση της TTAS έχει ελαφρώς καλύτερη η share nothing . Τέλος, οι TS και CAS δεν παρουσιάζουν κάποια ουσιώδεις διαφοροποίηση .

Σχετικά με το δεύτερο διάγραμμα μας, είναι δύσκολο να αντλήσουμε συμπεράσματα καθώς παρατηρούμε ότι ή μείωση του βαθμού διαμερισμού της cahe μπορεί να οδηγήσει σε αύξηση ή μείωση της κατανάλωσης ή να μην έχει καμία επίδραση . Ειδικότερα, παρατηρούμε ότι για το MUTEX μειώνεται η κατανάλωση για μεγαλύτερο διαμερισμό ενώ έχουμε την αντίθετη συμπεριφορά για την TTAS . Ακόμα για την TAS παρατηρούμε ότι οι TS και CAS φέρνουν διαφορετικά αποτελέσματα. Η TAS_CAS παραμένει πρακτικά σταθερή για όλες τις τοπολογίες ενώ η TAS_TS έχει αυξημένη κατανάλωση για την share all αλλά μετά συγκλίνει με την TAS_CAS.

Συνοψίζοντας τα παραπάνω , συμπεραίνουμε ότι από πλευράς χρόνου εκτέλεσης και κατανάλωσης ενέργειας η βέλτιστη επιλογή είναι τοπολογία share all με υλοποίηση TAS_CAS.

Μέρος Β

Δίνεται ο παρακάτω κώδικας :

```
0x00448408 LOOP:  LD F0, 0(R1)
0x0044840C        ADDD F4, F4, F0
0x00448410        LD F1, 0(R2)
0x00448414        MULD F4, F4, F1
0x00448418        ANDI R9, R8, 0x2
0x0044841C        BNEZ R9, NEXT
0x00448420 IF:    LD F2, 16(R2)
0x00448424        MULD F2, F2, F5
0x00448428        ADDD F4, F4, F2
0x0044842C NEXT:  LD F5, 8(R1)
0x00448430        ADDD F4, F4, F5
0x00448434        ADDI R1, R1, 0x8
0x00448438        SUBI R8, R8, 0x1
0x0044843C        BNEZ R8, LOOP
0x00448440        SD F4, 8(R2)
```

Αρχικά R8= 1

Σε αυτό το ερώτημα μας ζητάτε να παρουσιάσουμε σε ένα πίνακα τους χρόνους δρομολόγησης, εκτέλεσης και ολοκλήρωσης των καθώς και σχόλια για κάθε εντολή .

OP	IS	EX	WR	CMT	Σχόλια
LD F0, 0(R1)	1	2-5	6	7	Cache miss
ADDD F4, F4, F0	1	7-9	10	11	Raw F0

LD F1, 0(R2)	2	3-6	7	11	Cache miss
MULD F4, F4, F1	2	11-15	16	17	Raw F1, F4
ANDI R9, R8, 0x2	3	4-5	8	17	CDB conflict, R9=0
BNEZ R9, NEXT	3	9-10	11	18	Raw R9, Prediction: T, Actual: NT
LD F5, 8(R1)	7	8-8	9		Cache hit, Full Load Queue
ADDD F4, F4, F5	7				Fu Busy, Raw F4, Raw F5
ADDI R1, R1, 0x8	8	9-10			CBD conflict
SUBI R8, R8, 0x1	9	10-11			Full Integer RS, R8=0,
LD F2, 16(R2)	12	13-16	17	18	Cache miss
MULD F2, F2, F5	12	18-22	23	24	FU Busy, Raw F2
ADDD F4, F4, F2	13	24-26	27	28	Raw F4, F2
LD F5, 8(R1)	13	14-14	15	28	Cache hit
ADDD F4, F4, F5	14	28-30	31	32	Fu Busy, Raw F4, F5
ADDI R1, R1, 0x8	14	15-16	18	32	CBD conflict
SUBI R8, R8, 0x1	18	19-20	21	33	ROB Full, R8=0
BNEZ R8, LOOP	18	22-23	24	33	Prediction: T, Actual :NT
LD F0, 0(R1)	19	20	22		CBD conflict
ADDD F4, F4, F0	19				Raw F4,F0
SD F4, 8(R2)	25	32-35	36	37	Cache miss, Raw F4

Τέλος τα περιεχόμενα της cache ειναι:

A[0]	A[1]
B[0]	B[1]