ΜΥΥ023-ΠΑΡΑΛΛΗΛΑ ΣΥΣΤΗΜΑΤΑ ΚΑΙ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ
ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2018

1ο ΣΕΤ ΑΣΚΗΣΕΩΝ

Μπουρλή Στυλιανή
ΑΜ: 2774
email: stellabourli96@gmail.com

Πληροφορίες Συστήματος που χρησιμοποιήθηκε στην πρώτη άσκηση:

Όνομα υπολογιστή	opti7020ws01
Επεξεργαστής	Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz
Πλήθος πυρήνων	4 cores
Μεταφραστής	gcc (Ubuntu 4.8.4-2ubuntu1~14.04.3) 4.8.4

Πληροφορίες Συστήματος που χρησιμοποιήθηκε στη δεύτερη άσκηση:

Όνομα υπολογιστή	hp6000ws06
Επεξεργαστής	Intel(R) Core(TM)2 Quad CPU Q8400 @ 2.66GHz
Πλήθος πυρήνων	4 cores
Μεταφραστής	gcc (Ubuntu 4.8.4-2ubuntu1~14.04.3) 4.8.4

1η Άσκηση

“Αρχείο κώδικα σε γλώσσα C: multiply.c”

Στην άσκηση αυτή ζητήθηκε η υλοποίηση και η χρονομέτρηση παράλληλου προγράμματος για τον πολλαπλασιασμό τετραγωνικών πινάκων NxN, με χρήση νημάτων POSIX (PTHREADS) και αυτοδρομολόγησης (self-scheduling).

Λογική Υλοποίησης

Η βασική λογική της υλοποίησης είναι ότι ο τελικός πίνακας χωρίζεται σε μπλοκ, ο υπολογισμός καθενός από τα οποία αποτελεί μία εργασία. Στη συνέχεια, ένας προκαθορισμένος αριθμός νημάτων ζητάει να εκτελέσει μία εργασία από αυτές και επαναληπτικά, μόλις τελειώσει τον υπολογισμό, ξαναζητάει εργασία, έως ότου όλες οι εργασίες εκτελεστούν. Οι εργασίες δίνονται στα νήματα με τη σειρά που έχουν καθοριστεί, δηλαδή με τη σειρά που βρίσκονται τα μπλοκ στον πίνακα.

Υλοποίηση

Συνάρτηση **taskexecute**
Η συνάρτηση για τον υπολογισμό ενός μπλοκ του τελικού πίνακα.
Λειτουργία:
Στην αρχή υπολογίζω τη θέση του κάθε μπλοκ μέσω της διαίρεσης του αριθμού της τρέχουσας εργασίας με το πλήθος των μπλοκ. Πιο συγκεκριμένα, το πηλίκο δίνει τη γραμμή και το υπόλοιπο τη στήλη του μπλοκ. Το πλήθος των μπλοκ M, άρα και το μέγεθος του κάθε μπλοκ S = N/M, είναι παραμετροποιήσιμα. Το μέγεθος N του ολικού πίνακα είναι σταθερό και ίσο με 1024. Στη συνέχεια, υλοποιώ τον πολλαπλασιασμό μεταξύ των αντίστοιχων μπλοκ των 2 πινάκων και υπολογίζω το μπλοκ του τελικού πίνακα.

Συνάρτηση **thrfunc**
Η συνάρτηση την οποία εκτελούν τα νήματα.
Λειτουργία:
Σε αυτήν χρησιμοποιώ μία δομή επανάληψης while, ώστε τα νήματα να ζητούν επαναληπτικά εργασίες. Υπάρχει, επίσης, και μια κλειδαριά mutex έτσι ώστε να επιτυγχάνεται ο αμοιβαίος αποκλεισμός στην τροποποίηση της μεταβλητής t, η οποία δείχνει τον αριθμό της διεργασίας που θα αναλάβει να εκτελέσει το νήμα. Ένα νήμα τη φορά μπορεί να την αυξάνει και αν ολοκληρωθούν οι διαθέσιμες προς εκτέλεση εργασίες εκτελείται η εντολή break με την οποία τα νήματα σταματούν να ζητούν εργασίες.

Συνάρτηση **create_Arrays**
Η συνάρτηση για τη δημιουργία των 2 πινάκων που πρόκειται να πολλαπλασιαστούν, ώστε να προκύψει ο τελικός πίνακας.
Λειτουργία:
Μέσω των αρχείων “Amat1024” και “Bmat1024” λαμβάνω τα δεδομένα και δημιουργώ τον πίνακα A και τον πίνακα B. Σε σχόλια έχω την επιλογή να μπορεί ο χρήστης να εισάγει τα ονόματα των αρχείων που θέλει, αντί για τα αρχεία που έχω ως προεπιλογή. Αφού δημιουργηθούν οι πίνακες κλείνω τα αρχεία.

Συνάρτηση **main**
Η βασική συνάρτηση του προγράμματος.
Λειτουργία:
Αρχικά, καλώ τη συνάρτηση create_Arrays για να δημιουργηθούν οι πίνακες που πρόκειται να πολλαπλασιαστούν. Έπειτα, δημιουργώ τα νήματα. Το πλήθος των νημάτων NUM_OF_THREADS είναι παραμετροποιήσιμο. Στη συνέχεια, εκτελώ join, ώστε η main να περιμένει όλα τα νήματα να τελειώσουν πριν συνεχίσει τη λειτουργία της. Αφού τα νήματα τελειώσουν δημιουργώ ένα αρχείο “Cmat1024” στο οποίο αποθηκεύω τον τελικό πίνακα που προέκυψε από τον υπολογισμό των 2 πινάκων. Τέλος, ενημερώνω το χρήστη και κλείνω το αρχείο.

Χρονομέτρηση – Γραφικές Παραστάσεις - Παρατηρήσεις

Χρονομέτρηση

Για τη χρονομέτρηση χρησιμοποίησα εκτός από το δικό μου παράλληλο πρόγραμμα, το αντίστοιχο σειριακό πρόγραμμα που δινόταν ώστε να κάνω κάποιες συγκρίσεις. Το αρχείο κώδικα του σειριακού προγράμματος είναι το multiply_s.c.
Επίσης, το χρόνο εκτέλεσης του προγράμματος τον υπολόγισα με τη συνάρτηση gettimeofday().

Δοκίμασα 3 διαφορετικά πλήθη εργασιών: 16, 256, 1024. Για κάθε ένα από αυτά εκτέλεσα το πρόγραμμά μου για 1,2,3,4,8 και 16 νήματα. Κάθε εκτέλεση έγινε 4 φόρες και από τους χρόνους που προέκυψαν υπολόγισα τους μέσους όρους. Επίσης χρονομέτρησα και το σειριακό πρόγραμμα.

Όλες οι μετρήσεις φαίνονται στους παρακάτω πίνακες:

Σειριακό πρόγραμμα
N = 1024

1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος Όρος
3.866293	4.778142	3.921331	3.921331	4.12177425

Παράλληλο πρόγραμμα
N = 1024
Εργασίες = 16

Νήματα	1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος Όρος
1	2.831403	2.860099	2.837407	2.836198	2.84127675
2	1.417903	1.429893	1.433456	1.436735	1.42949675
3	0.981692	0.979877	0.978048	0.978698	0.97957875
4	0.764536	0.760123	0.762430	0.765848	0.76323425
8	0.760440	0.769847	0.759144	0.762741	0.763193
16	0.764764	0.765752	0.761286	0.763035	0.76370925

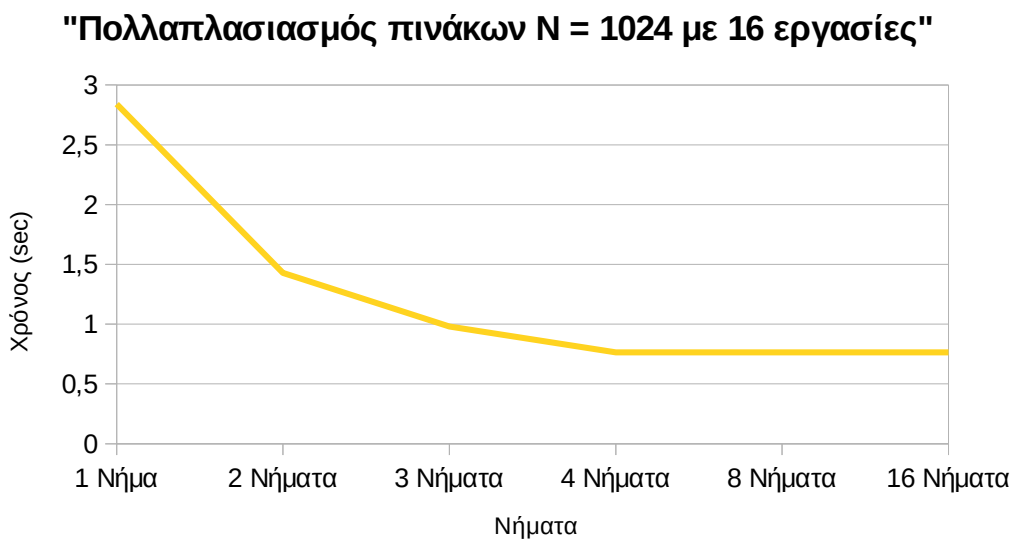
Παράλληλο πρόγραμμα
N = 1024
Εργασίες = 256

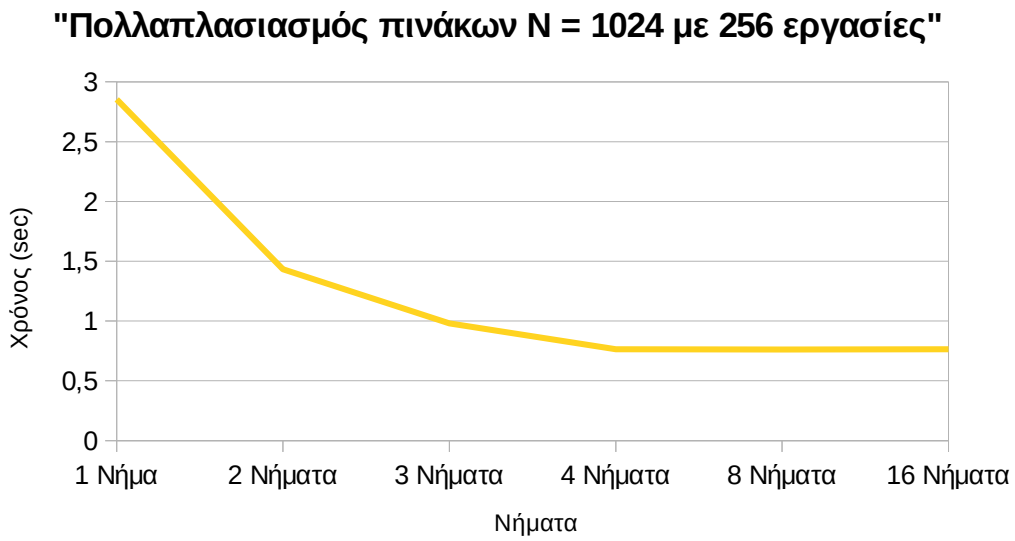
Νήματα	1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος Όρος
1	2.846815	2.860571	2.862892	2.846379	2.85416425
2	1.431294	1.430136	1.433911	1.434974	1.43257875
3	0.981673	0.979046	0.983727	0.979250	0.980924
4	0.759863	0.761409	0.768428	0.762800	0.763125
8	0.760249	0.761182	0.766210	0.759954	0.76189875
16	0.765081	0.770325	0.757944	0.764178	0.764382

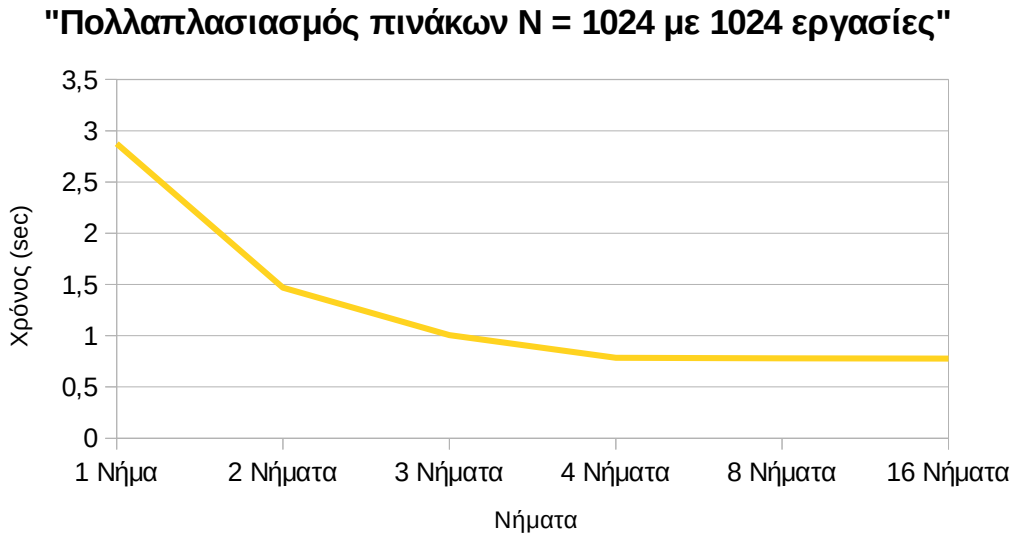
Παράλληλο πρόγραμμα
N = 1024
Εργασίες = 1024

Νήματα	1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος Όρος
1	2.864686	2.882186	2.878864	2.868156	2.873473
2	1.453792	1.475231	1.472465	1.469063	1.46763775
3	1.005659	1.002058	1.010477	1.005184	1.0058445
4	0.787232	0.782145	0.783304	0.781393	0.7835185
8	0.780678	0.776741	0.783791	0.778723	0.77998325
16	0.777074	0.779576	0.777535	0.776989	0.7777935

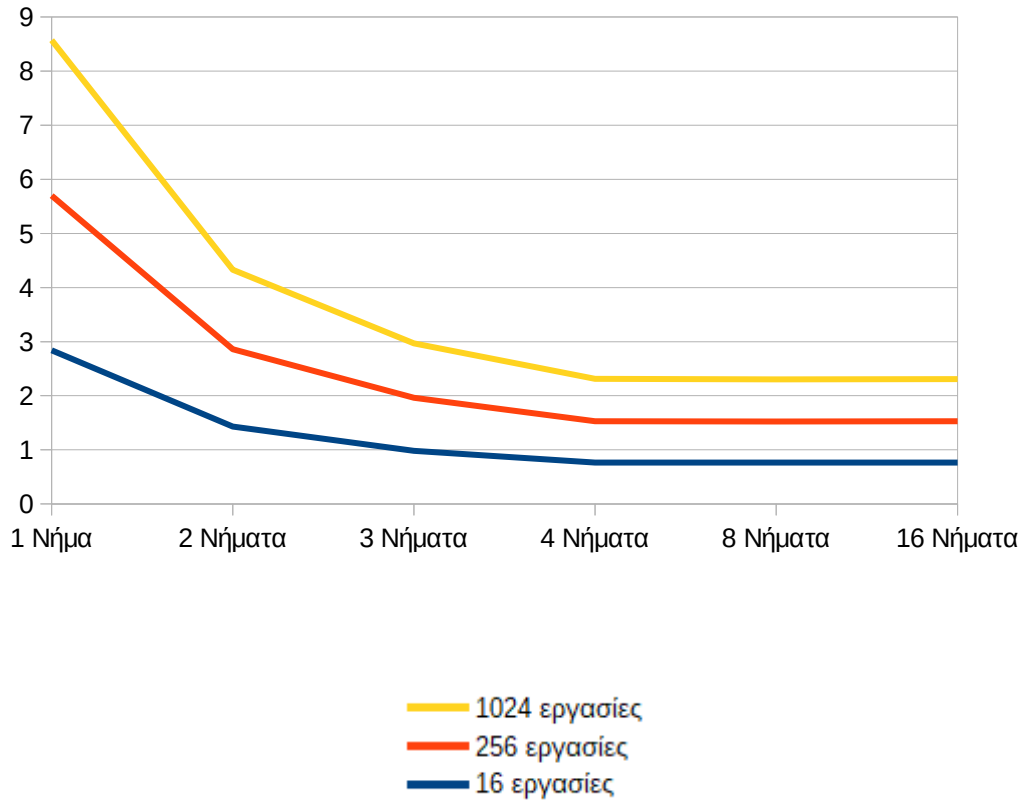
Γραφικές Παραστάσεις







"Πολλαπλασιασμός πινάκων N = 1024 με το διαφορετικό πλήθος διεργασιών"



Παρατηρήσεις

Παρατηρώντας τις παραπάνω γραφικές παραστάσεις κατέληξα στο συμπέρασμα ότι όσο αυξάνεται ο αριθμός των νημάτων, τόσο μειώνεται ο χρόνος εκτέλεσης του προγράμματος. Αυτό είναι απολύτως λογικό, διότι όσα περισσότερα νήματα υπάρχουν,έχοντας ένα λογικό ποσό εργασιών να εκτελέσουν, τόσο πιο γρήγορα τελειώνουν οι εργασίες.
Επίσης, παρατήρησα ότι όσο αυξάνονται οι εργασίες, αν και ισχύει η προηγούμενη παρατήρηση, δηλαδή αυξάνοντας τον αριθμό των νημάτων ο χρόνος εκτέλεσης μειώνεται, παρόλ' αυτά, όσο περισσότερες είναι οι εργασίες, ο χρόνος εκτέλεσης είναι μεγαλύτερος. Αυτό είναι λογικό, διότι όσο περισσότερες εργασίες έχουμε, τόσο μικρότερη είναι κάθε εργασία και έτσι έχουμε τελικά λεπτόκοκκο, καθώς επίσης ένας ακόμη λόγος είναι ότι υπάρχουν μόνο 4 cores διαθέσιμα.

2η Άσκηση

“Αρχεία κώδικα σε γλώσσα C: barrier.c (περιέχει τις συναρτήσεις), barrier_test.c (δοκιμαστικό πρόγραμμα με δικές μου συναρτήσεις), barrier_test_p.c (δοκιμαστικό πρόγραμμα με χρήση PTHREAD)”

Στην άσκηση αυτή ζητήθηκε η υλοποίηση του δικού μας μηχανισμού barrier και η σύγκρισή του με τον αντίστοιχο μηχανισμό των νημάτων POSIX. Πιο συγκεκριμένα, η υλοποίηση αφορούσε τρεις συναρτήσεις, την barrier_init(b,n), την barrier_wait(b) και την barrier_destroy(b), όπου b είναι δείκτης σε μία δομή barrier_t και n είναι το πλήθος των νημάτων που συμμετέχουν.

Υλοποίηση

Δομή **struct barrier_t**
Δημιούργησα μία δομή η οποία περιέχει:

- Ένα μετρητή **counter** ο οποίος μετράει πόσα νήματα εισέρχονται στη συνάρτηση barrier_wait.
- Μια μεταβλητή **state** η οποία χρησιμοποιείται στη συνάρτηση barrier_wait.
- Δύο μεταβλητές **flag** και **flag2** έτσι ώστε να ελέγχεται αν το νήμα που φτάνει είναι το τελευταίο από τα νήματα που πρέπει να συγχρονιστούν.
- Μία μεταβλητή threads που περιέχει τον αριθμό των νημάτων που πρέπει να συγχρονιστούν.
- Μία κλειδαριά **mutex m**, που χρειάζεται για την επίτευξη αμοιβαίου αποκλεισμού στις συναρτήσεις barrier_wait και barrier_destroy.
- Μία μεταβλητή συνθήκης **cond c**, για την επίτευξη συγχρονισμού των νημάτων στις συναρτήσεις barrier_wait και barrier_destroy.

Συνάρτηση **barrier_init**
Σε αυτή τη συνάρτηση αρχικοποίησα τις μεταβλητές που υπάρχουν στη δομή και όρισα τα νήματα που πρέπει να συγχρονιστούν.

Συνάρτηση **barrier_wait**
Τη συνάρτηση αυτή την υλοποίησα με δύο τρόπους.

1ος τρόπος: Χρησιμοποίησα μία δομή if – else, έτσι ώστε να επαναχρησιμοποιείται η συνάρτηση, και να λειτουργεί σωστά σε περίπτωση που δεν έχουν φύγει όλα τα νήματα από τη συνάρτηση και κάποιο φτάσει εκείνη τη στιγμή. Πιο συγκεκριμένα, όταν μπαίνει ένα νήμα, κλειδώνει το mutex, αυξάνει το μετρητή και μπαίνει στο state = 0, δηλαδή στο if και περιμένει. Όλα τα νήματα που έρχονται μέχρι να έρθει και το τελευταίο από αυτά που πρέπει να συγχρονιστούν περιμένουν. Μόλις φτάσει και το τελευταίο αλλάζει τη μεταβλητή συνθήκης flag = 0 σε flag = 1, μηδενίζει το μετρητή, στέλνει σήμα broadcast στα άλλα νήματα και αλλάζει το state =0 σε state = 1, έτσι ώστε αν κάποιο νήμα που έχει φύγει ξαναμπεί στη συνάρτηση, να περιμένει τα υπόλοιπα για να συγχρονιστούν. Αντίστοιχα λειτουργεί ο κώδικας στο else.

2ος τρόπος: Στο δεύτερο τρόπο έκανα μία πολύ απλή υλοποίηση, χωρίς να χρησιμοποιήσω όμως while loop. Όταν ένα νήμα εισέρχεται στη συνάρτηση, κλειδώνει το mutex, αυξάνει το μετρητή και αν δεν είναι το τελευταίο από τα νήματα που πρέπει να συγχρονιστούν περιμένει, αλλιώς μηδενίζει το μετρητή και στέλνει σήμα στα υπόλοιπα νήματα.

Συνάρτηση **barrier_destroy**
Στη συνάρτηση αυτή ελέγχω αρχικά αν κάποια από τα νήματα βρίσκονται στην barrier_wait και στη συνέχεια αν δεν υπάρχει κάποιο καταστρέφω τη μεταβλητή συνθήκης και την κλειδαριά.

Δοκιμαστικό πρόγραμμα
Για τον έλεγχο της λειτουργίας των συναρτήσεων μου, δημιούργησα ένα δοκιμαστικό πρόγραμμα, στο οποίο εκτός από τις συναρτήσεις που έφτιαξα, υπάρχει:

Συνάρτηση **func**
Είναι η συνάρτηση την οποία υλοποιούν τα νήματα επαναληπτικά.
Ένα νήμα τη φορά αυξάνει μία κοινόχρηστη μεταβλητή s, στη συνέχεια περιμένει τα άλλα νήματα έτσι ώστε όλα να συγχρονιστούν με barrier. Αφού συγχρονιστούν αυξάνεται ένας μετρητής που μετράει τις επαναλήψεις και μία μεταβλητή work_done παίρνει την τιμή 1 ώστε να γίνουν κάποιοι έλεγχοι. Αν η μεταβλητή work_done έχει την τιμή 1, τότε αν έχουν γίνει όλες οι επαναλήψεις (REPETITIONS) που έχουμε ορίσει, σταματάει να επαναλαμβάνεται η διαδικασία. Διαφορετικά τα νήματα συγχρονίζονται ξανά με χρήση του barrier, η μεταβλητή work_done μηδενίζεται και τα νήματα επαναλαμβάνουν τη διαδικασία.

Συνάρτηση **main**
Στη βασική συνάρτηση του προγράμματος, δημιουργώ ένα προκαθορισμένο πλήθος νημάτων NUM_OF_THREADS. Στη συνέχεια, εκτελώ join, ώστε η main να περιμένει όλα τα νήματα να τελειώσουν πριν συνεχίσει τη λειτουργία της.

Χρονομέτρηση – Γραφικές Παραστάσεις – Παρατηρήσεις

Χρονομέτρηση
Δημιούργησα δύο αρχεία με το δοκιμαστικό πρόγραμμα που έφτιαξα, όπου στο ένα χρησιμοποίησα τις συναρτήσεις barrier που έφτιαξα εγώ, και στο άλλο τις pthread_barrier του POSIX και χρονομέτρησα το χρόνο εκτέλεσης των προγραμμάτων, ώστε να κάνω τις κατάλληλες συγκρίσεις.

Μεταβάλλοντας το πλήθος των νημάτων σε 1,2,4 και 10 και των αριθμό των επαναλήψεων σε 5,10 και 20 σε κάθε περίπτωση, έλαβα τα δεδομένα που παρουσιάζονται παρακάτω:

Πρόγραμμα με χρήση δικών μου συναρτήσεων
Επαναλήψεις = 5

Νήματα	1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος Όρος
1	0.000233	0.000240	0.000253	0.000229	0.00023875
2	0.000401	0.000368	0.000380	0.000360	0.00037725
4	0.000792	0.000809	0.000901	0.000656	0.0007895
10	0.001705	0.001803	0.001767	0.001578	0.00171325

Πρόγραμμα με χρήση δικών μου συναρτήσεων
Επαναλήψεις = 10

Νήματα	1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος Όρος
1	0.000255	0.000249	0.000245	0.000251	0.00025
2	0.000565	0.000470	0.000836	0.000531	0.0006005
4	0.001622	0.001252	0.001246	0.001362	0.0013705
10	0.003356	0.003279	0.003541	0.003376	0.003388

Πρόγραμμα με χρήση δικών μου συναρτήσεων
Επαναλήψεις = 20

Νήματα	1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος Όρος
1	0.000290	0.000271	0.000286	0.000280	0.00028175
2	0.000688	0.000909	0.000695	0.000743	0.00075875
4	0.002259	0.001993	0.002226	0.002591	0.00226725
10	0.009419	0.005528	0.006285	0.005034	0.0065665

Πρόγραμμα με χρήση συναρτήσεων pthread POSIX
Επαναλήψεις = 5

Νήματα	1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος Όρος
1	0.000242	0.000251	0.000245	0.000232	0.0002425
2	0.000488	0.000594	0.000395	0.000608	0.00052125
4	0.000872	0.000716	0.000877	0.000720	0.00079625
10	0.001443	0.001739	0.001892	0.001887	0.00174025

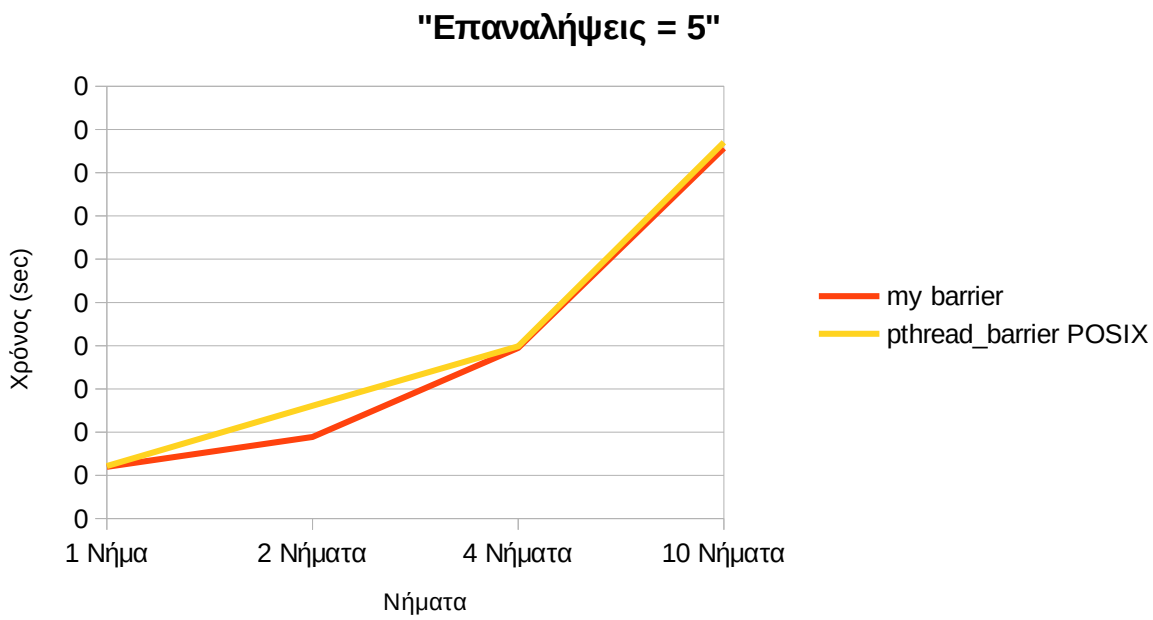
Πρόγραμμα με χρήση συναρτήσεων pthread POSIX
Επαναλήψεις = 10

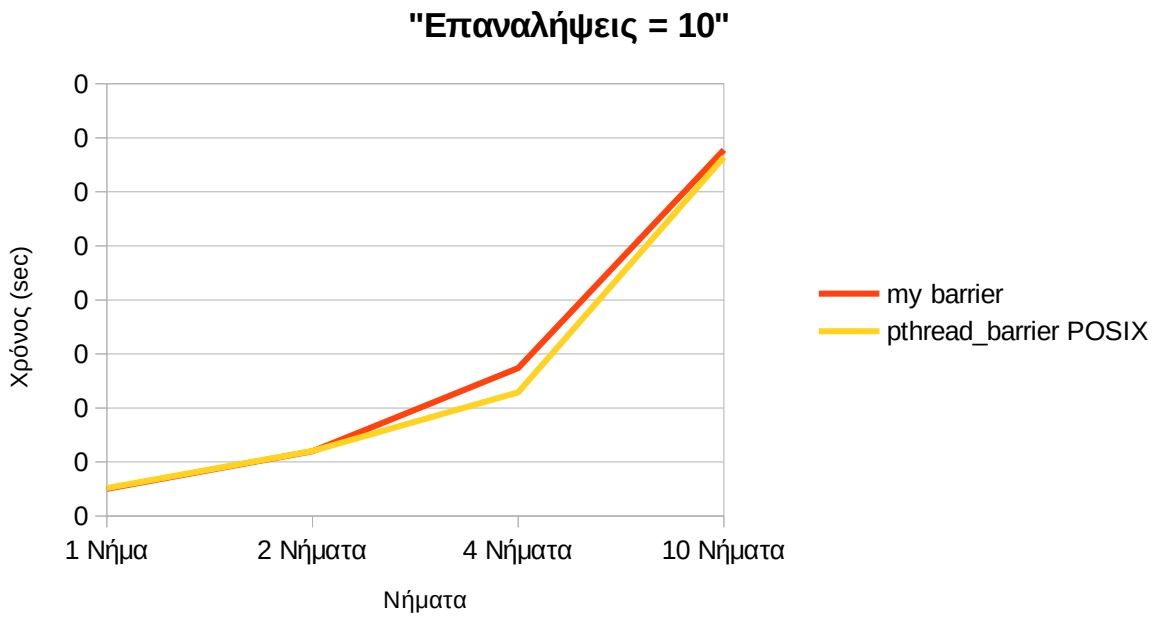
Νήματα	1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος Όρος
1	0.000264	0.000253	0.000263	0.000254	0.0002585
2	0.000692	0.000553	0.000610	0.000551	0.0006015
4	0.001234	0.001115	0.001004	0.001221	0.0011435
10	0.003297	0.002703	0.003536	0.003752	0.003322

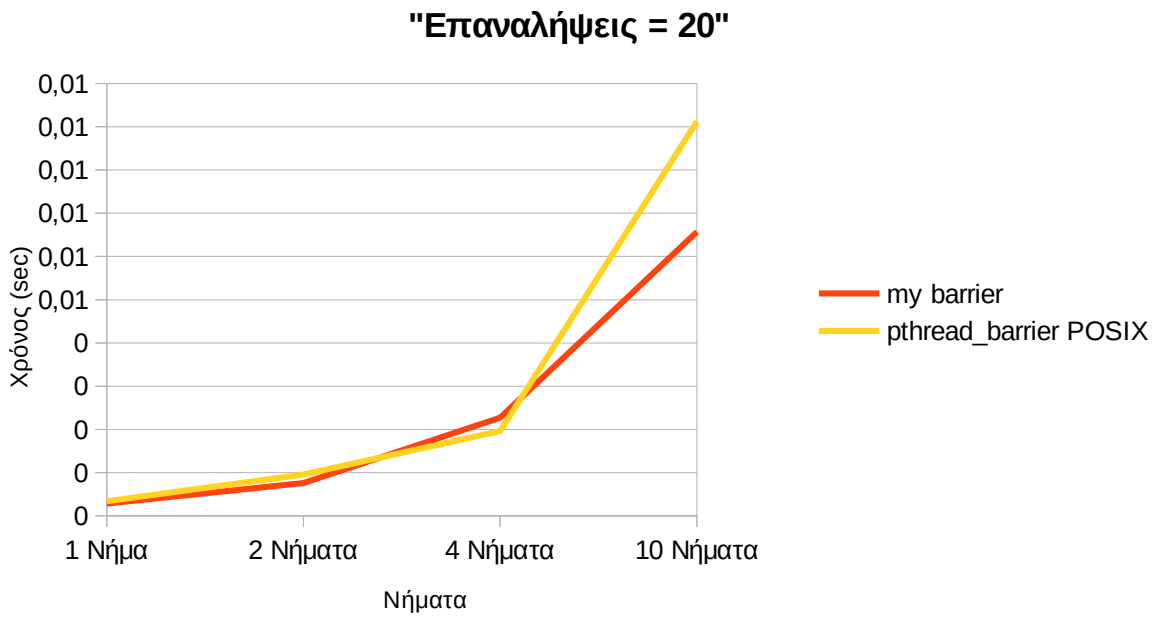
Πρόγραμμα με χρήση συναρτήσεων pthread POSIX
Επαναλήψεις = 20

Νήματα	1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος Όρος
1	0.000447	0.000302	0.000299	0.000305	0.00033825
2	0.001235	0.000851	0.000889	0.000843	0.0009545
4	0.001505	0.002636	0.001757	0.001940	0.0019595
10	0.008952	0.010115	0.007821	0.009598	0.0091215

Γραφικές Παραστάσεις







Παρατηρήσεις

Σύμφωνα με τις παραπάνω γραφικές παραστάσεις, όσο περισσότερα είναι τα νήματα, τόσο περισσότερο καθυστερεί η εκτέλεση του προγράμματος. Αυτό είναι λογικό, διότι περισσότερα νήματα που χρειάζονται συγχρονισμό θα ελέγχουν συνθήκες, θα κλειδώνουν και θα ξεκλειδώνουν την κλειδαριά, θα περιμένουν και όλα αυτά συνεπάγονται καθυστέρηση. Επίσης, όσον αφορά τη σύγκριση των δύο προγραμμάτων, είναι σχεδόν ίδια σε χρόνο, τις περισσότερες φορές όμως το πρόγραμμα με τις δικές μου συναρτήσεις είναι λίγο πιο γρήγορο.