ΜΥΕ023–Παράλληλα Συστήματα και Προγραμματισμός 2019-20

Σετ προγραμμάτων #1 (OpenMP)

Χριστόφορος Καρβέλης
ΑΜ: 12345
E-mail: cse02989@cse.uoi.gr, christoph.karv@yahoo.gr

Το 2ο σετ ασκήσεων αφορά στον παράλληλο προγραμματισμό με το μοντέλο κοινόχρηστου χώρου διευθύνσεων μέσω του προτύπου OpenMP. Ζητείται η παραλληλοποίηση 3 εφαρμογών: 1. πολλαπλασιασμός πινάκων, 2. υπολογιζμός του πλήθους των πρώτων αριθμών καθώς και του μεγαλύτερου πρώτου αριθμού και 3. πολλαπλασιασμός τετραγωνικών πινάκων.

Όλες οι μετρήσεις έγιναν στο παρακάτω σύστημα:

Όνομα υπολογιστή	opti3060ws03
Επεξεργαστής	Intel i3-8300
Πλήθος πυρήνων	4
Μεταφραστής	gcc v7.5.0

Άσκηση 1

Το πρόβλημα

Στην άσκηση αυτή ζητείται να παραλληλοποιηθεί ο πολλαπλασιασμός πινάκων χρησιμοποιώντας OpenMP. Αυτό πρέπει να γίνει παραλληλοποιώντας έναν από τους 3 βρόχους κάθε φορά χρησιμοποιώντας static και dynamic schedules για κάθε for loop. Τέλος ζητείται να χρονομετρηθεί η κάθε περίπτωση.

Μέθοδος παραλληλοποίησης

Χρησιμοποιήθηκε το σειριακό πρόγραμμα από την ιστοσελίδα του μαθήματος. Για την παραλληλοποίηση του πρώτου βρόχου απλά προστέθηκε η οδηγία

#pragma omp parallel private(i, j , k, sum)

πριν τον πρώτο βρόχο του i. Οι μεταβλητές i, j, k και sum πρέπει να είναι ιδιωτικές ώστε κάθε νήμα να κάνει τους υπολογισμούς στο δικό του χώρο χωρίς να επηρεάζει τα υπόλοιπα ενώ οι A , B και C είναι κοινόχρηστες. Δεν απαιτείται αμοιβαίος αποκλεισμός μιας και κάθε νήμα επηρεάζει διαφορετικά στοιχεία του C .

Παρόμοια, για την παραλληλοποίηση του δεύτερου βρόχου.

Για την παραλληλοποίηση του τρίτου βρόχου, όμως, απαιτείται αμοιβαίος αποκλεισμός. Έτσι για την παραλληλοποίηση του τρίτου βρόχου χρησιμοποιήθηκε η οδηγία

#pragma omp parallel private(k) shared(sum)

πριν τον πρώτο βρόχο του k. Οι μεταβλητές i, j, και k πρέπει να είναι ιδιωτικές ώστε κάθε νήμα να κάνει τους υπολογισμούς στο δικό του χώρο χωρίς να επηρεάζει τα υπόλοιπα ενώ

οι A , B, C και sum είναι κοινόχρηστες. Απαιτείται αμοιβαίος αποκλεισμός μιας και κάθε νήμα επιρεάζει το ίδιο sum. Έτσι προστέθηκε η οδηγία

#pragma omp atomic

πριν τον υπολογισμό του sum καθώς μειώνει τον χρόνο εισαγωγής και εξαγωγής από την κρίσιμη περιοχή αφου δεν χρησιμοποιεί κλείδωμα/ξεκλείδωμα απλά κάνει την πράξεις χώρις να μπορεί αλλό νήμα να πειράξει το sum εκείνη την στιγμή.

Πειραματικά αποτελέσματα - μετρήσεις

Τα προγράμματα εκτελέστηκαν στο σύστημα που αναφέρθηκε στην εισαγωγή και η χρονομέτρηση έγινε με τη συνάρτηση omp_get_wtime(). Χρησιμοποιήθηκαν 4 νήματα, και 2 διαφορετικοί τύποι scheduling(static, dynamic). Κάθε πείραμα εκτελέστηκε 4 φορές και υπολογίστηκαν οι μέσοι όροι των χρόνων. Προσοχή δόθηκε οι χρόνοι να μην συμπεριλαμβάνουν την ανάγνωση των αρχείων. Τα αποτελέσματα δίνονται στους παρακάτω πίνακες (οι χρόνοι είναι σε sec):

Για την πρώτη for

Shceduling	1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος όρος
static	1.059807	0.935070	0.887793	0.879482	0.9405380
dynamic	1.011702	0.879493	1.018928	0.879319	0.9473605

Για την δεύτερη for

Shceduling	1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος όρος
static	0.889570	0.884058	0.959979	0.889626	0.90580825
dynamic	0.896783	0.900738	0.892478	0.894780	0.89619475

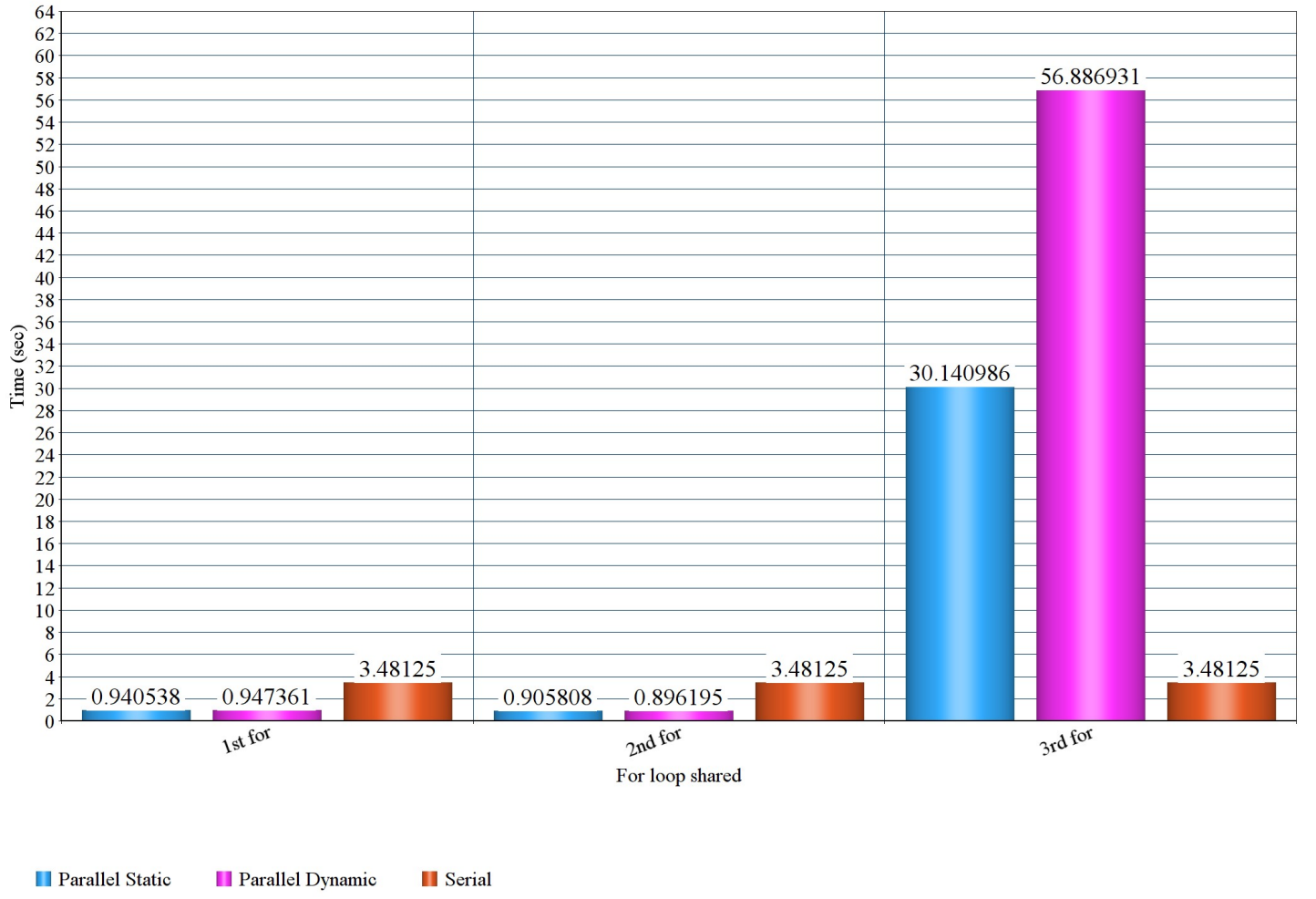
Για την τρίτη for

Shceduling	1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος όρος
static	32,224028	32,349362	32,294513	23,696041	30,140986
dynamic	55,916086	56,653680	59,420521	55,557437	56,886931

Για το σειριακό

1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος όρος
3,480173	3,480621	3,482184	3,482020	3,4812495

Με βάση τους πίνακες των αποτελεσμάτων προκύπτει η παρακάτω γραφική παράσταση



Σχόλια

Με βάση τα αποτελέσματα βλέπουμε ότι ενώ ο διαμοιρασμός εργασίας της πρώτης και δεύτερης for για static και dynamic schedules δίνουν καλούς χρόνους εκτέλεσης ενώ ο διαμοιρασμός εργασίας της τρίτη for προκαλεί ραγδαία αύξηση στους χρόνους εκτέλεσης. Όλα τα παραπάνω είναι μάλλον αναμενόμενα διότι και οι 2 πρώτες for μοιράστηκαν ισόποσα και κάθε μία είχε παρόμοιο φόρτο υπολογισμών με την άλλη σε κάθε περίπρωση. Στην τρίτη όμως, καθώς έχουμε κοινόχρηστη μεταβλητή(sum) η οποία επεξεργάζεται συνεχώς από όλα τα νήματα επιβάλεται η δημιουργία κρίσιμης περιοχής. Έτσι δημιουργείται επιπλέον κόστος στον χρόνο εκτέλεσης λόγο της διαχείρισης των νημάτων και της μνήμης για την πριοχή αυτή.

Άσκηση 2

Το πρόβλημα

Στην άσκηση αυτή ζητείται να παραλληλοποιηθεί ο υπολογιζμός του πλήθους των πρώτων αριθμών καθώς και του μεγαλύτερου πρώτου αριθμού χρησιμοποιώντας OpenMP. Αυτό πρέπει να γίνει χωρίς να να αλλάξουμε τον αλγόριθμο, απλά να μοιράσουμε σωστά τη δουλειά. Τέλος ζητείται να χρονομετρηθεί η κάθε περίπτωση για εναλλακτικούς τρόπους διαμοιρασμού της δουλειάς μεταξύ των νημάτων.

Μέθοδος παραλληλοποίησης

Χρησιμοποιήθηκε το σειριακό πρόγραμμα από την ιστοσελίδα του μαθήματος. Για τον σωστό διαμοιρασμό της δουλειάς πρέπει να παραλληλοποίησουμε τον βρόχο for που υπάρχει στο πρόγραμμα. Έτσι απλά προστέθηκε η οδηγία

#pragma omp parallel private(i, num, divisor, quotient, remainder) shared(count) reduction(max: lastprime)

πριν τον πρώτο βρόχο του i. Οι μεταβλητές i, num, divisor, quotient και remainder πρέπει να είναι ιδιωτικές ώστε κάθε νήμα να κάνει τους υπολογισμούς στο δικό του χώρο χωρίς να επηρεάζει τα υπόλοιπα ενώ η count είναι κοινόχρηστη. Επίσης επειδή θέλουμε η lastprime να περιέχει τον τελευταίο μεγαλύτερο περιττό αριθμό που υπολογίστηκε στο δοθέν διάστημα την δηλώνουμε ως reduction(max: lastprime). Τέλος απαιτείται αμοιβαίος αποκλεισμός μιας και κάθε νήμα επιρεάζει το ίδιο count. Έτσι προστέθηκε η οδηγία

#pragma omp atomic

πριν τον υπολογισμό του count καθώς μειώνει τον χρόνο εισαγωγής και εξαγωγής από την κρίσιμη περιοχή αφου δεν χρησιμοποιεί κλείδωμα/ξεκλείδωμα απλά κάνει την πράξει χώρις να μπορεί αλλό νήμα να πειράξει το sum εκείνη την στιγμή.

Πειραματικά αποτελέσματα - μετρήσεις

Τα προγράμματα εκτελέστηκαν στο σύστημα που αναφέρθηκε στην εισαγωγή και η χρονομέτρηση έγινε με τη συνάρτηση omp_get_wtime(). Χρησιμοποιήθηκαν 4 νήματα, και 3 διαφορετικοί τύποι scheduling(static, dynamic, guided) ενώ χρησιμοποιήθηκε και runtime για να δω κατα πόσο ο χρόνος που θα δώσει θα είναι κοντά στο κλύτερο. Κάθε πείραμα εκτελέστηκε 4 φορές και υπολογίστηκαν οι μέσοι όροι. Τα αποτελέσματα δίνονται στον παρακάτω πίνακα (οι χρόνοι είναι σε sec):

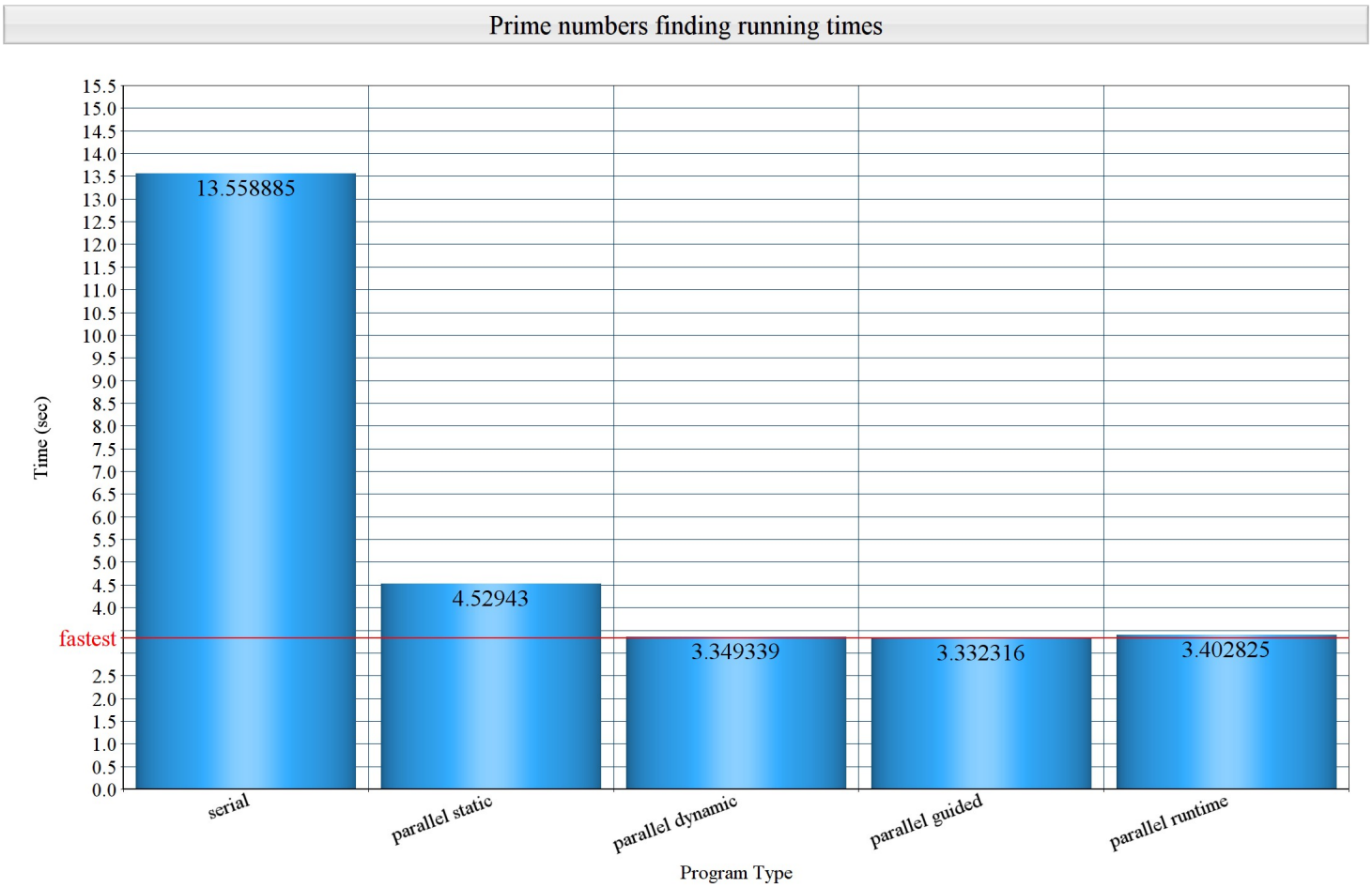
Για το παράλληλο τμήμα

Shceduling	1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος όρος
static	4,529390	4,529305	4,528796	4,530229	4,529430
dynamic	3,349069	3,348891	3,348857	3,350539	3,349339
guided	3,332366	3,332288	3,332114	3,332496	3,332316
runtime	3,402494	3,402465	3,402687	3,403654	3,402825

Για το σειριακό τμήμα

1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος όρος
13,558659	13,561592	13,557027	13,558262	13,558885

Με βάση τους πίνακες των αποτελεσμάτων προκύπτει η παρακάτω γραφική παράσταση



Σχόλια

Με βάση τα αποτελέσματα βλέπουμε ότι ο παραλιλισμός της for δίνει πολύ καλύτερους χρόνους για όλους τους τύπους των scheduling. Από αυτούς βλέπουμε ότι καλύτερα αποτελέσμτα δίνουν οι dynamic και guided(τον καλύτερο χρόνο τον δίνει το guided) ενώ πολύ κοντά είναι και ο runtime. Όλα τα παραπάνω είναι αναμενόμενα καθώς στο dynamic τα νήματα συναγωνίζονται για κάθε επανάληψη έτσι έχουμε συχνό ανταγωνισμό μεταξύ των νημάτων και άρα περισσότερη καθυστέρηση ενώ στο guided δίνονται στα νήματα μεγάλα κομμάτια επαναλήψεων και όσο περνάει η ώρα μικρότερα και έτσι δεν έχουμε τοσο συχνό ανταγωνισμό μεταξύ των νημάτων. Ενώ το static είναι ποίο αργό απο αυτές καθώς μοιράζεται σε κάθε νήμα περίπου το ¼ των συνολικών επαναλήψεων άσχετα με τον

φόρτο που υπάρχει σε κάθε νήμα, αν όμως έχουμε διαφορά στην ταχύτητα των νημάτων αυτό θα έχει ως αποτέλεσμα κάποιο “αργό” νήμα να ασχολείται με αρκετές επαναλήψεις σε αντίθεση με τα guided, dynamic όπου ένα “αργό” νήμα θα πάρει λιγότερες επαναλήψεις καθώς ενα ποιο “γρήγορο” θα έχει προλάβει να πάρει παραπάνω επαναλήψεις καθώς τελείωνει τους υπολογισμούς του πιο γρήγορα.

Άσκηση 3

Το πρόβλημα

Στην άσκηση αυτή ζητείται να παραλληλοποιηθεί ο πολλαπλασιασμός τετραγωνικών πινάκων, χρησιμοποιώντας εργασίες του OpenMP (tasks). Αυτό πρέπει να γίνει με τέτοιον τρόπο ώστε κάθε εργασία να είναι ο υπολογισμός ενός block (υποπίνακα) του αποτελέσματος, μεγέθους S × S. Τέλος ζητείται να χρονομετρηθεί η κάθε περίπτωση για εναλλακτικά μεγέθη υποπινάκων.

Μέθοδος παραλληλοποίησης

Χρησιμοποιήθηκε το σειριακό πρόγραμμα από την ιστοσελίδα του μαθήματος. Για την “διάσπαση” του τετραγωνικού πίνακα C σε υποπίνακες χρησιμοποιήθηκε ο αλγόριθος που βρίσκεται στο σετ διαφανειών 4ου κεφαλαίου, μέρος I: νήματα POSIXΔιεύθυνση URL στην διαφάνεια 43. Έτσι απλά προστέθηκε η οδηγία

#pragma omp parallel

πριν τον πρώτο βρόχο του i ο οποίος δημιουργεί τόσα tasks όσοι και οι υποπίνακες που προκύπτουν εντός της οδηγίας

#pragma omp parallel single

η οποία αναθέτει την δημιουργία των tasks σε ένα μόνο νήμα. ‘Επειτα για την δημιουργία των tasks χρησιμοποιήθηκε η εντολή

#pragma omp task firstprivate(i)

Οι μεταβλητή i πρέπει να είναι ιδιωτική καινα αρχικοποιηθεί στον αριθμό του υποπίνακα ώστε κάθε νήμα να κάνει τους υπολογισμούς στο δικό του χώρο χωρίς να επηρεάζει τα υπόλοιπα ενώ όλες οι άλλες είναι κοινόχρηστες.

Πειραματικά αποτελέσματα - μετρήσεις

Τα προγράμματα εκτελέστηκαν στο σύστημα που αναφέρθηκε στην εισαγωγή και η χρονομέτρηση έγινε με τη συνάρτηση omp_get_wtime(). Χρησιμοποιήθηκαν 4 νήματα, και 3 διαφορετικά μεγέθοι υποπινάκων (16, 256, 1024). Κάθε πείραμα εκτελέστηκε 4 φορές και υπολογίστηκαν οι μέσοι όροι των χρόνων. Τα αποτελέσματα δίνονται στους παρακάτω πίνακες (οι χρόνοι είναι σε sec):

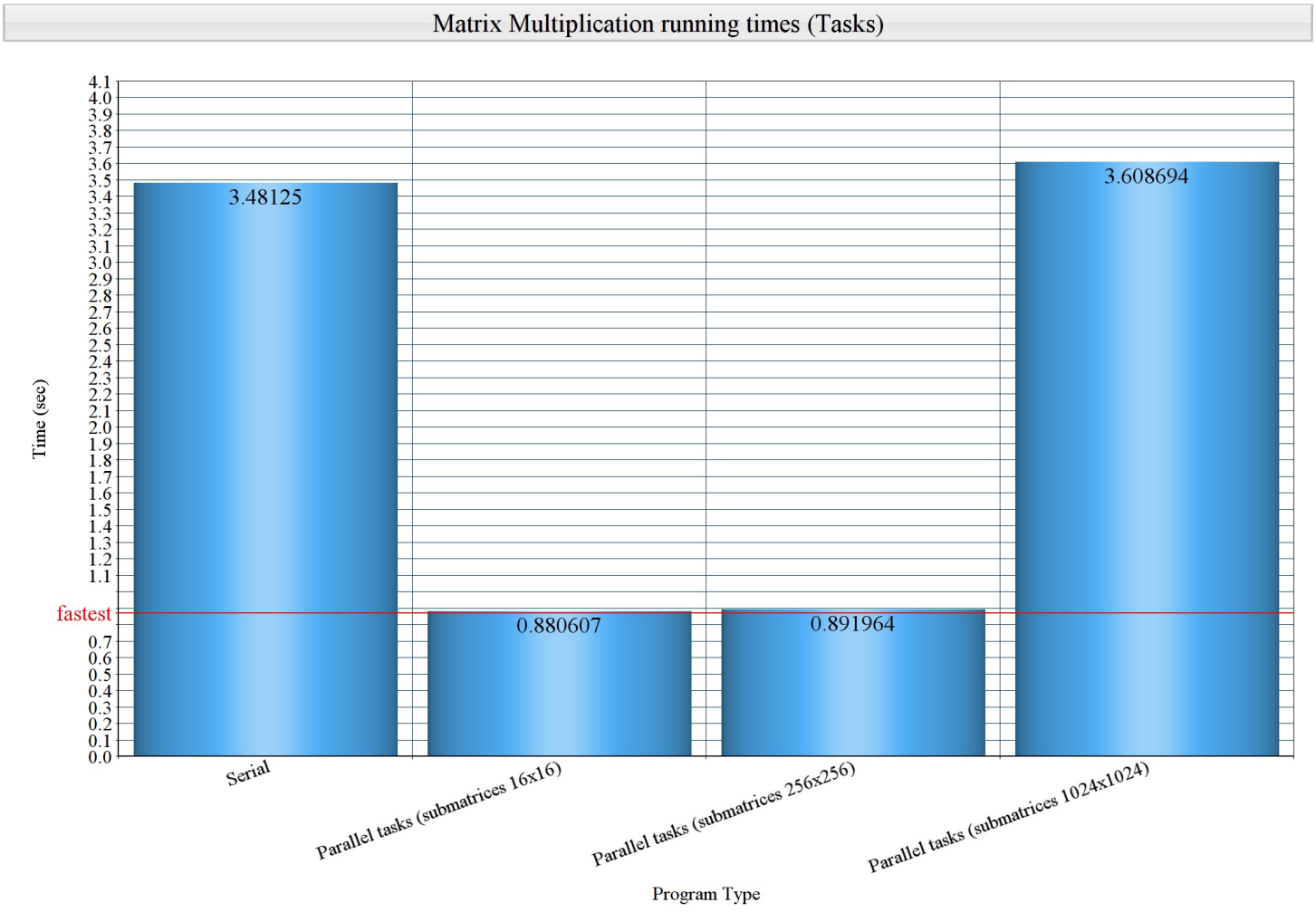
Για το παράλληλο πρόγραμμα

Μέγεθος υποπινάκων	1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος όρος
16x16	0,885251	0,878686	0,876218	0,882274	0,88060725
256x256	0,880399	0,903524	0,893832	0,890100	0,89196375
1024x1024	3,491203	3,491560	3,963554	3,488458	3,60869375

Για το σειριακό προγραμμα

1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος όρος
3,480173	3,480621	3,482184	3,482020	3,4812495

Με βάση τους πίνακες των αποτελεσμάτων προκύπτει η παρακάτω γραφική παράσταση



Σχόλια

Με βάση τα αποτελέσματα βλέπουμε για μεγέθη υποπινάκων που είναι αρκετά μικρτότερα του 1024 έχουμε καλούς χρόνους εκτέλεσης ενώ αυξάνονται όσο αυξάνουν και τα μεγέθη όπου τελικά για μέγεθος 1024x1024 έχουμε χρόνο εκτέλεσης μεγαλύτερο του σειριακού προγράμματος. Όλα τα παραπάνω είναι λογικά αποτελέσματα καθώς για μικρά μεγέθη

υποπινάκων δημιουργούνται περισσότερα tasks τα οποία διαμοιράζονται στα νήματα όπου μπορούν να εκτελεστούν παράλληλα άρα η δουλεία που πρέπει να κάνει το κάθε νήμα μειώνεται. Όσο αυξάνει το μέγεθος των υποπινάκων όμως και πλησιάζει το αρχικό (1024x1024), το πλήθος των tasks μειώνεται καθώς έχουμε μεγαλύτερους υποπίνακες άρα αυξάνεται ο όγκος της εργασίας που πρέπει να εκτελεστεί από κάθε νήμα καθώς έχουμε υπολογισμό μεγάλων υποπινάκων από το κάθε νήμα. Τέλος για υποπίνακα 1024x1024 έχουμε έναν μόνο υποπίνακα άρα ο χρόνος είναι ισός με τον σειριακό συν τους χρόνους για την δημιουργία και την διαχείριση του ενός τάσκ και των 4 νημάτων.