ΜΥΕ023 – Παράλληλα Συστήματα και Προγραμματισμός

Σετ Ασκήσεων #1

Σπυρίδωνος Βασίλειος

Το πρώτο σετ ασκήσεων αφορά τον παράλληλο προγραμματισμό με το μοντέλο κοινόχρηστου χώρου διευθύνσεων μέσω του προτύπου OpenMP. Ζητείται η παραλληλοποίηση 3 εφαρμογών, δύο με πολλαπλασιασμό πινάκων και μία με εύρεση πρώτων αριθμών.

Όλες οι μετρήσεις έγιναν στο παρακάτω σύστημα:

Όνομα υπολογιστή	opti3060ws04
Επεξεργαστής	Intel i3-8300
Πλήθος πυρήνων	4
Μεταφραστής	gcc 7.5.0

Άσκηση 1

Το πρόβλημα

Στην άσκηση αυτή ζητείται να παραλληλοποιηθεί και να χρονομετρηθεί ένα σειριακό πρόγραμμα που υλοποιεί πολλαπλασιασμό τετραγωνικών πινάκων, παραλληλοποιώντας (έναν κάθε φορά) τους βρόχους του και δοκιμάζοντας static και dynamic schedules. Οι πίνακες έχουν 1024 x 1024 ακέραια στοιχεία και δίνονται στην ιστοσελίδα του μαθήματος.

Μέθοδος παραλληλοποίησης

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

#pragma omp parallel for schedule(runtime) private(j , k, sum) num_threads(4)

πριν τον πρώτο βρόχο του i. Οι μεταβλητές 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. Το schedule(runtime) υπάρχει για να μπορούμε να αλλάξουμε το schedule σε static ή dynamic από τη γραμμή εντολών χρησιμοποιώντας τις εντολές :

export OMP_SCHEDULE=“STATIC” ή

export OMP_SCHEDULE=“DYNAMIC”

Το num_threads(4) δηλώνει ότι ο αριθμός νημάτων που θα δημιουργηθούν θα είναι 4.

Παρόμοια, για την παραλληλοποίηση του δεύτερου βρόχου προστέθηκε η οδηγία :

#pragma omp parallel for schedule(runtime) private(k, sum) num_threads(4)

πριν το βρόχο του j. Η μόνη διαφορά από την οδηγία που προστέθηκε στον πρώτο βρόχο, είναι η απουσία της μεταβλητής j μέσα στο private(). Δε χρειάζεται γιατί το σύστημα την κάνει αυτόματα ιδιωτική.

Για την παραλληλοποίηση του τρίτου βρόχου προστέθηκε η οδηγία :

#pragma omp parallel for schedule(runtime) num_threads(4) reduction(+:sum)

πριν το βρόχο του k. Το reduction(+:sum) προστέθηκε γιατί αλλιώς θα είχαμε πρόβλημα με την εντολή προσαύξησης της μεταβλητής sum, εφόσον έχουμε 4 νήματα τα οποία πρόκειται να μεταβάλουν ταυτόχρονα τη μεταβλητή. Το reduction(+:sum) δουλεύει ως εξής : οι μεταβλητές sum θα είναι ιδιωτικές σε κάθε νήμα, τα νήματα θα τις υπολογίζουν και στο τέλος θα συνδυαστούν με πρόσθεση αφού μέσα στο reduction έχουμε τον τελεστή ‘+’. Επίσης, ακριβώς πριν από την οδηγία #pragma…. προστέθηκε το: sum = 0, ώστε κάθε φορά που πρόκειται να υπολογίσουμε ένα στοιχείο του C, το άθροισμα να είναι ίσο με μηδέν.

Πειραματικά αποτελέσματα – μετρήσεις

Τα προγράμματα εκτελέστηκαν στο σύστημα που αναφέραμε στην εισαγωγή και η χρονομέτρηση έγινε με τη συνάρτηση omp_get_wtime(). Χρησιμοποιήθηκαν 4 νήματα σε όλα τα προγράμματα. Για να ελέγξω αν τα αποτελέσματα είναι ορθά, χρησιμοποίησα στο τερματικό την εντολή :

diff x y

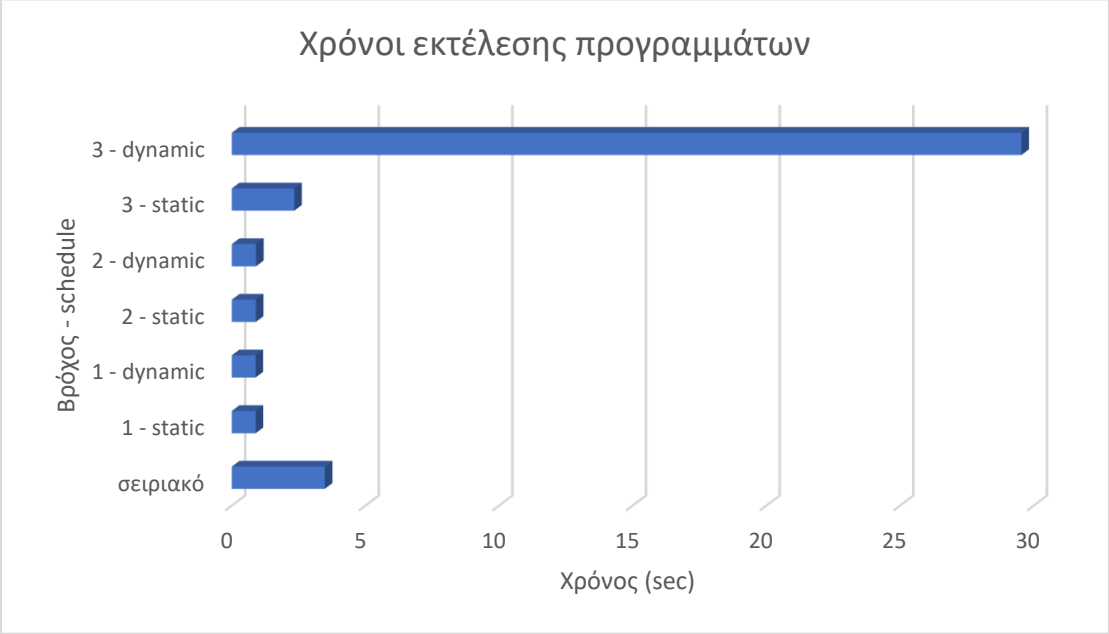
, όπου x : το αρχείο με τα σωστά αποτελέσματα του σειριακού προγράμματος

και y : το αρχείο με τα αποτελέσματα του κάθε προγράμματος που υλοποίησα.

Το πρόγραμμα 1 είναι αυτό που έχει παραλληλοποιημένο τον πρώτο βρόχο. Ομοίως, τα προγράμματα 2 και 3 έχουν παραλληλοποιημένο το δεύτερο και τρίτο βρόχο αντίστοιχα. Κάθε πείραμα εκτελέστηκε 4 φορές και υπολογίστηκαν οι μέσοι όροι. Προσοχή δόθηκε οι χρόνοι να μη συμπεριλαμβάνουν την ανάγνωση των αρχείων. Τα αποτελέσματα δίνονται στον παρακάτω πίνακα (οι χρόνοι είναι σε δευτερόλεπτα) :

Πρόγραμμα	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	**4η εκτέλεση**	**Μέσος Όρος**
Σειριακό	3,489074	3,489569	3,490294	3,447243	3,4790450
1 – static	0,889096	0,889710	0,889197	0,894779	0,8906955
1 – dynamic	0,885758	0,885767	0,886001	0,886021	0,8858868
2 – static	0,897789	0,895892	0,919508	0,890194	0,9008458
2 – dynamic	0,903112	0,903414	0,912548	0,909364	0,9071095
3 – static	2,412357	2,270674	2,484156	2,183316	2,3376258
3 – dynamic	29,231236	29,783149	29,736892	29,416741	29,542005

Με βάση τους πίνακες των αποτελεσμάτων προκύπτει η παρακάτω γραφική παράσταση.



Σχόλια

Με βάση τα αποτελέσματα κάνουμε τις εξής παρατηρήσεις :

- Τα παραλληλοποιημένα προγράμματα είναι πιο γρήγορα από το σειριακό, εκτός από το τρίτο πρόγραμμα με dynamic schedule. Λογικό, εφόσον η αρχική δουλειά μοιράζεται στα 4 νήματα. Επίσης, παρατηρούμε ότι οι χρόνοι των παράλληλων προγραμμάτων είναι περίπου το ¼ του χρόνου του σειριακού προγράμματος.
- Τα προγράμματα 1 και 2 έχουν πολύ μικρές διαφορές μεταξύ τους όσον αφορά το χρόνο εκτέλεσης. Από τα δύο, φαίνεται ότι το πρόγραμμα 1 με dynamic schedule είναι ελάχιστα πιο γρήγορο.

- Το τρίτο πρόγραμμα με static schedule είναι πιο αργό από τα προγράμματα 1 και 2 και πάρα πολύ πιο αργό με dynamic schedule. Οπότε, η παραλληλοποίηση του τρίτου βρόχου δεν είναι η πιο αποδοτική. Ο λόγος που αυξάνεται τόσο πολύ ο χρόνος εκτέλεσης του τρίτου προγράμματος με dynamic schedule είναι ο κακός διαμοιρασμός του φορτίου στα νήματα. Θα υπάρχουν πολλές εναλλαγές μεταξύ των νημάτων το οποίο ευθύνεται για τη μεγάλη καθυστέρηση. Επίσης, το dynamic schedule παίρνει πιο πολύ χρόνο καθώς «αποφασίζει» την ώρα της εκτέλεσης για το ποιο τμήμα θα πάρει το κάθε νήμα.

Άσκηση 2

Το πρόβλημα

Στην άσκηση αυτή ζητείται να παραλληλοποιηθεί και να χρονομετρηθεί ένα σειριακό πρόγραμμα που υπολογίζει το πλήθος των πρώτων αριθμών καθώς και το μεγαλύτερο πρώτο αριθμό μέχρι και το N, για οποιοδήποτε N. Συγκεκριμένα, ζητήθηκε να συμπληρώσουμε τη συνάρτηση openmp_primes() ώστε να κάνει τους ίδιους υπολογισμούς με τη σειριακή

συνάρτηση serial_primes(), αλλά παράλληλα και χωρίς να αλλάξουμε τον αλγόριθμο.

Μέθοδος παραλληλοποίησης

Χρησιμοποιήθηκε το πρόγραμμα από την ιστοσελίδα του μαθήματος. Για την υλοποίηση της συνάρτησης openmp_primes(), χρησιμοποιήθηκε ο αλγόριθμος της συνάρτησης serial_primes(). Για την παραλληλοποίηση της συνάρτησης προστέθηκε η οδηγία :

#pragma omp parallel for schedule(guided) private(num, divisor, quotient, remainder) num_threads(4) reduction(+:count) reduction(max:lastprime)

πριν τον πρώτο βρόχο του i. Επέλεξα guided schedule καθώς έτσι πέτυχα το μικρότερο χρόνο εκτέλεσης (περισσότερα σε λίγο). Οι μεταβλητές 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. Το num_threads(4) δηλώνει ότι ο αριθμός νημάτων που θα δημιουργηθούν θα είναι 4.

Το reduction(+:count) προστέθηκε γιατί αλλιώς θα είχαμε πρόβλημα με την εντολή προσαύξησης της μεταβλητής count, εφόσον έχουμε 4 νήματα τα οποία πρόκειται να μεταβάλουν ταυτόχρονα τη μεταβλητή. Το reduction(+:count) δουλεύει ως εξής : οι μεταβλητές count θα είναι ιδιωτικές σε κάθε νήμα, τα νήματα θα τις υπολογίζουν και στο τέλος θα συνδυαστούν με πρόσθεση αφού μέσα στο reduction έχουμε τον τελεστή ‘+’. Με αυτόν τον τρόπο θα έχουμε στο αποτέλεσμα το σωστό πλήθος των πρώτων αριθμών. Το reduction(max:lastprime)

προστέθηκε ώστε τελικά στη μεταβλητή lastprime που είναι ο μεγαλύτερος πρώτος αριθμός μέχρι και το N, να αποθηκευτεί όντως ο μεγαλύτερος πρώτος αριθμός. Αν δεν υπήρχε το reduction(max:lastprime), η τιμή της μεταβλητής lastprime δε θα ήταν απαραίτητα ο μεγαλύτερος πρώτος αριθμός, αλλά οποιοσδήποτε αριθμός ανατέθηκε στη μεταβλητή lastprime από το τελευταίο νήμα που εκτελέστηκε. Το reduction(max:lastprime) δουλεύει ως εξής : οι μεταβλητές lastprime θα είναι ιδιωτικές σε κάθε νήμα, τα νήματα θα τις υπολογίζουν και στο τέλος θα συνδυαστούν με τον τελεστή 'max', αναθέτοντας έτσι στη μεταβλητή lastprime το μέγιστο πρώτο αριθμό μέχρι το N.

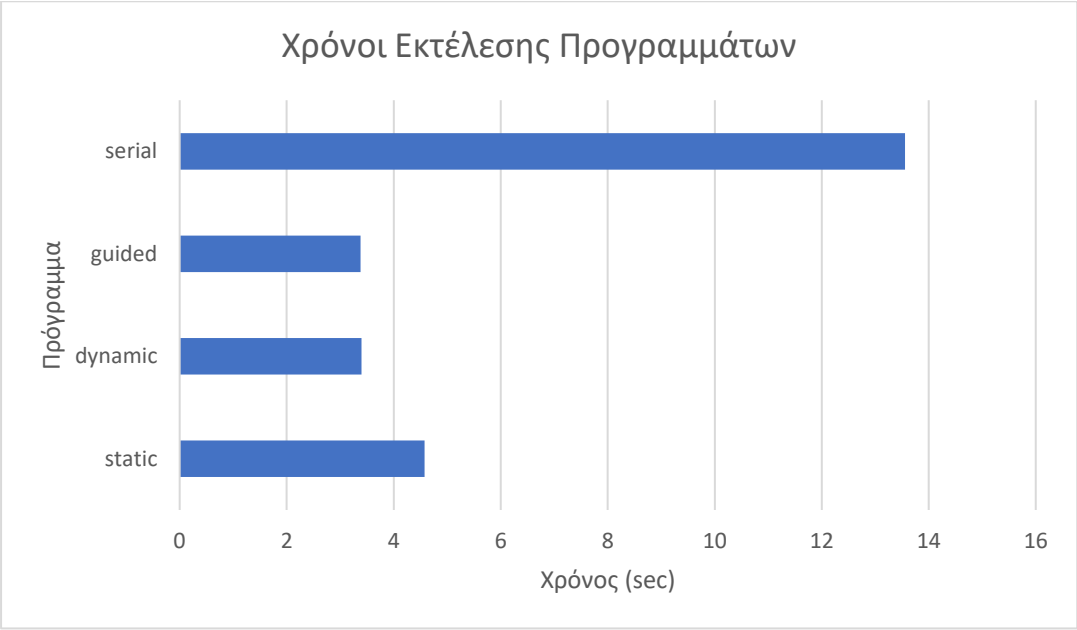
Πειραματικά αποτελέσματα – μετρήσεις

Δοκιμάστηκαν 3 διαφορετικά προγράμματα. Η μόνη διαφορά τους είναι στο schedule που χρησιμοποιήθηκε. Τα προγράμματα 1,2 και 3 χρησιμοποιούν static, dynamic και guided schedules αντίστοιχα. Τελικά, χρησιμοποιήθηκε το guided schedule, με το οποίο πέτυχα το μικρότερο χρόνο εκτέλεσης. Τα προγράμματα εκτελέστηκαν στο σύστημα που αναφέραμε στην εισαγωγή και η χρονομέτρηση έγινε με τη συνάρτηση omp_get_wtime(). Χρησιμοποιήθηκαν 4 νήματα σε όλα τα προγράμματα.

Τα αποτελέσματα της χρονομέτρησης δίνονται στον παρακάτω πίνακα (οι χρόνοι είναι σε δευτερόλεπτα) :

Πρόγραμμα	1η εκτέλεση	2η εκτέλεση	3η εκτέλεση	4η εκτέλεση	Μέσος Όρος
static	4,576730	4,576688	4,576545	4,576937	4,576725
dynamic	3,399851	3,398304	3,398403	3,398505	3,398766
guided	3,379722	3,379892	3,380864	3,379793	3,380068
serial	13,557247	13,556848	13,556703	13,556694	13,556873

Με βάση τους πίνακες των αποτελεσμάτων προκύπτει η παρακάτω γραφική παράσταση.



Σχόλια

Με βάση τα αποτελέσματα κάνουμε τις εξής παρατηρήσεις :

- Τα παραλληλοποιημένα προγράμματα είναι πιο γρήγορα από το σειριακό. Λογικό, εφόσον η αρχική δουλειά

μοιράζεται στα 4 νήματα. Επίσης, παρατηρούμε ότι οι χρόνοι των προγραμμάτων με dynamic και guided schedule είναι περίπου το ¼ του χρόνου του σειριακού προγράμματος, ενώ ο χρόνος του προγράμματος με static schedule είναι περίπου το 1/3 του σειριακού προγράμματος. Ένας πιθανός λόγος είναι η άνιση κατανομή του φορτίου στα νήματα. Συγκεκριμένα, με static schedule, το νήμα που θα αναλάβει να εκτελέσει τις τελευταίες επαναλήψεις του for θα κάνει περισσότερο χρόνο από π.χ. το νήμα που ανέλαβε τις πρώτες επαναλήψεις, γιατί στις τελικές επαναλήψεις οι υπολογισμοί είναι πιο μεγάλοι. Αντίθετα, με το dynamic και το guided schedule, θα έχουμε πιο αποτελεσματική κατανομή των μεγάλων υπολογισμών στα νήματα.

- Τα προγράμματα με dynamic και guided schedule έχουν πολύ μικρές διαφορές μεταξύ τους όσον αφορά το χρόνο εκτέλεσης. Από τα δύο, φαίνεται ότι το πρόγραμμα με guided schedule είναι ελάχιστα πιο γρήγορο.

Άσκηση 3

Το πρόβλημα

Στην άσκηση αυτή ζητείται να παραλληλοποιηθεί και να χρονομετρηθεί ένα σειριακό πρόγραμμα που υλοποιεί πολλαπλασιασμό τετραγωνικών πινάκων, χρησιμοποιώντας

εργασίες του OpenMP (tasks). Η κάθε εργασία θα πρέπει να είναι ο υπολογισμός ενός block (υποπίνακα) του αποτελέσματος, μεγέθους S x S. Οι πίνακες έχουν 1024 x 1024 ακέραια στοιχεία και δίνονται στην ιστοσελίδα του μαθήματος.

Δεν κατάφερα να υλοποιήσω το ζητούμενο αυτής της άσκησης. Ωστόσο, ακόμη και αν δεν έκανα την υλοποίηση με τους υποπίνακες, παραλληλοποίησα το πρόγραμμα χρησιμοποιώντας tasks και αποφάσισα να το ανεβάσω, να το χρονομετρήσω και να φτιάξω και τη σχετική αναφορά. Προφανώς, βέβαια, γνωρίζω ότι δεν είναι αυτό που ζητήθηκε…!

Μέθοδος παραλληλοποίησης

Χρησιμοποιήθηκε το σειριακό πρόγραμμα από την ιστοσελίδα του μαθήματος. Αρχικά, προστέθηκε η οδηγία :

#pragma omp parallel num_threads(4)

που δηλώνει την αρχή της παράλληλης περιοχής και τον αριθμό των νημάτων που θα χρησιμοποιηθούν (4).

Ακολουθεί η οδηγία :

#pragma omp single

που δηλώνει ότι το παρακάτω μπλοκ κώδικα που είναι μέσα στις αγκύλες θα εκτελεστεί από μόνο ένα νήμα.

Έπειτα η οδηγία :

#pragma omp task firstprivate(i, j, k, sum)

η οποία όπως αναφέρθηκε παραπάνω θα εκτελεστεί μόνο από ένα νήμα, θα αναθέσει εργασίες (tasks) στα υπόλοιπα νήματα. 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. Επίσης, την ώρα που θα εκτελεστούν, θα πρέπει να έχουν τις ίδιες τιμές με τη στιγμή που ανατέθηκαν στα νήματα. Ο κώδικας που ακολουθεί, θα διαβάζεται από το νήμα που μπήκε πρώτο στο #pragma omp single και αυτό θα αναθέτει tasks στα υπόλοιπα (3) νήματα. Τα tasks αυτά θα έχουν ολοκληρωθεί στο τέλος ενός barrier. Επιπλέον, έχει προστεθεί η οδηγία :

#pragma omp taskwait

πάνω από την εντολή :

C[i][j] = sum;

Καθώς, πρέπει να έχουν ολοκληρωθεί τα tasks που υπολογίζουν τη μεταβλητή sum κάθε φορά, ώστε να ανατεθεί η τιμή της στο αντίστοιχο στοιχείο του C.

Πειραματικά αποτελέσματα – μετρήσεις

Τα προγράμματα εκτελέστηκαν στο σύστημα που αναφέραμε στην εισαγωγή και η χρονομέτρηση έγινε με τη συνάρτηση omp_get_wtime(). Χρησιμοποιήθηκαν 4 νήματα. Για να ελέγξω αν τα αποτελέσματα είναι ορθά, χρησιμοποίησα στο τερματικό την εντολή :

diff x y

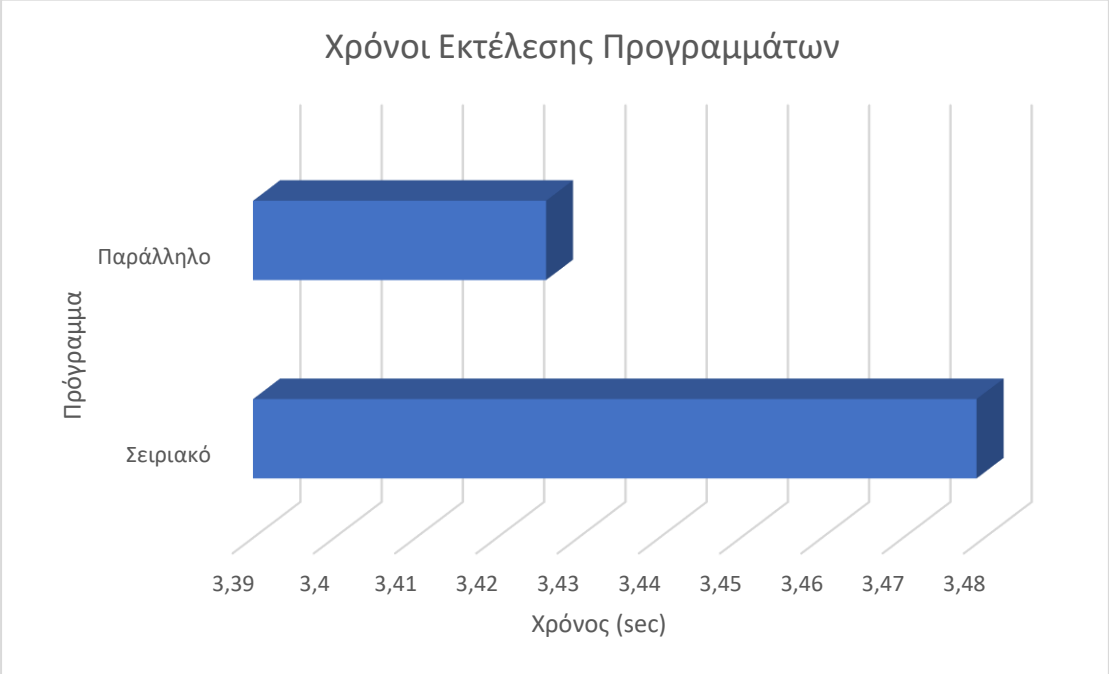
, όπου x : το αρχείο με τα σωστά αποτελέσματα του σειριακού προγράμματος

και y : το αρχείο με τα αποτελέσματα του προγράμματος που υλοποίησα.

Το σειριακό και το παράλληλο πρόγραμμα εκτελέστηκαν 4 φορές και υπολογίστηκαν οι μέσοι όροι. Προσοχή δόθηκε οι χρόνοι να μη συμπεριλαμβάνουν την ανάγνωση των αρχείων. Τα αποτελέσματα δίνονται στον παρακάτω πίνακα (οι χρόνοι είναι σε δευτερόλεπτα) :

Πρόγραμμα	**1η εκτέλεση**	**2η εκτέλεση**	**3η εκτέλεση**	**4η εκτέλεση**	**Μέσος Όρος**
Σειριακό	3,489074	3,489569	3,490294	3,447243	3,4790450
Παράλληλο	3,419277	3,422039	3,421050	3,441870	3,4260590

Με βάση τους πίνακες των αποτελεσμάτων προκύπτει η παρακάτω γραφική παράσταση.



Σχόλια

Με βάση τα αποτελέσματα κάνουμε τις εξής παρατηρήσεις :

- Το παραλληλοποιημένο πρόγραμμα είναι ελάχιστα πιο γρήγορο από το σειριακό.