Παράλληλα και Διανεμημένα Συστήματα

Εργασια 1

Pthreads & OpenMP

Ζαμπόκας Γεώργιος
ΑΕΜ: 7173
Ημερομηνία: 26/12/2013

Για την υλοποίηση της άσκησης χρησιμοποίησα αναδρομικό κώδικα σε γλώσσα C. Ξεκίνησα με σειριακή υλοποίση και έπειτα προχώρησα στις παράλληλες για διευκόλυνση αλλά και για να υπάρχει ενα μέτρο σύγκρισης στην ταχύτητα εκτέλεσης.

Για την υλοποίηση σε OPEN MP

Χρησιμοποίησα τις παρακάτω συναρτήσεις:

- void cube(Box* boxParent)

Η βασική συνάρτηση υπολογισμου των κύβων. Παίρνει ως όρισμα εναν δείκτη τύπου Box (πατέρας) ώστε να έχουμε όλες τις απαραίτητες μεταβλητές για την δημιουργία νέων Box.

Δημιουργεί ένα νέο Box και τοποθετεί τα στοιχεία του που υπολογίζονται με απλούς τύπους.

Στη συνέχεια ψάχνει για σημεία που βρίσκονται εντός του νέου κύβου ξεκινώντας από το πρώτο στοιχείο του πατέρα κύβου γλιτώνοντας έτσι μερικές συκγρίσεις.

Αν τα στοιχεία είναι 0, δημιουργεί φύλλο με boxid=0.

Αν τα στοιχεια είναι λιγότερα από S, δημιουργεί φύλλο και περνάει τα στοιχεία που ανήκουν στον κύβο στον πίνακα B.

Αν τα στοιχεία είναι περισσότερα απο S, καλεί αναδρομικά τον εαυτό της.

- void findColleagues(Box* b)

Η συνάρτηση υπολογισμού των colleagues. Δέχεται ως όρισμα κάθε Box που υπάρχει στον πίνακα box[ ] και υπολογίζει τους γειτονικούς κύβους. Αυτό πραγματοποιείται με την μέτρηση των αποστάσεων τον κέντρων ανάμεσα στους κύβους.Για να είναι γείτονες πρέπει αυτή η απόσταση να ειναι ίση με την ακμή του κύβου (οριζόντιοι και κάθετοι γείτονες) η με 1,41 επί την ακμή (διαγώνιοι γείτονες).

Για την υλοποίηση σε pthreads

Χρησιμοποιήθηκαν οι ίδιες συναρτήσεις με την διαφορά ότι πλέον δέχονται void* ως ορίσματα και μέσα σε αύτες κάθε αναφορά σε στοιχείο της δομής γίνεται με typecast του εκάστοτε pointer σε τύπου Box*.

Παρ'ολα αυτά προστέθηκαν και 2 νέες:

- void* rootCube(void* root)

  Καλείται από το πρώτο thread και δημιουργείται μέσα στη main. Με τη σειρά της καλεί τα επόμενα 8 threads που αποτελούν τα 8 παιδία της ρίζας. Κάθε φορά καλεί και την pthtread_join για να περιμένουμε την δημιουργία των παιδιών πρώτα πριν συνεχιστεί το πρόγραμμα.

- void* startColleagues(void)

  Καλείται μέσω του πρώτου thread για την διαδικασία εύρεσης των γειτόνων. Για κάθε κουτί δημιουργεί ενα thread που καλεί την findColleagues. Χρησιμοποιούμε και πάλι την pthread_join για να πετύχουμε συγχρονισμένη συνέχεια του προγράμματος.

Επιπλέον στην υλοποίηση με pthreads ορίζουμε έναν αριθμό από 2000 threads για να χρησιμοποιήσουμε (αρκετός σε όλες τις περιπτώσεις).

Τέλος, όταν πρέπει να γίνει μεταβολή μιας global μεταβλητής χρησιμοποιείται η μέθοδος του αμοιβαίου αποκλεισμού (σε αυτή την υλοποίηση ορίζουμε 4 mutex keys).

Στα κοινά των υλοποιήσεων:

Ξεκινώ ορίζω τα απαραίτητα headers κάθε φορά, τις global μεταβλητές που χρειάζονται και την δομή Box για την αναπαράσταση των κύβων.

Στην main παίρνω από την κονσόλα τον αριθμό των στοιχείων και τον φυσικό αριθμό S.

Ορίζω μαζί τους πίνακες A,B στο κατάληλο μέγεθος. Τα στοιχεία τους αποτελούν σημεία της επιφάνειας της μοναδιαίας σφαίρας στο πρώτο ογδοημόριο και ακολουθουν unitary κατανομή πάνω σε αυτή.

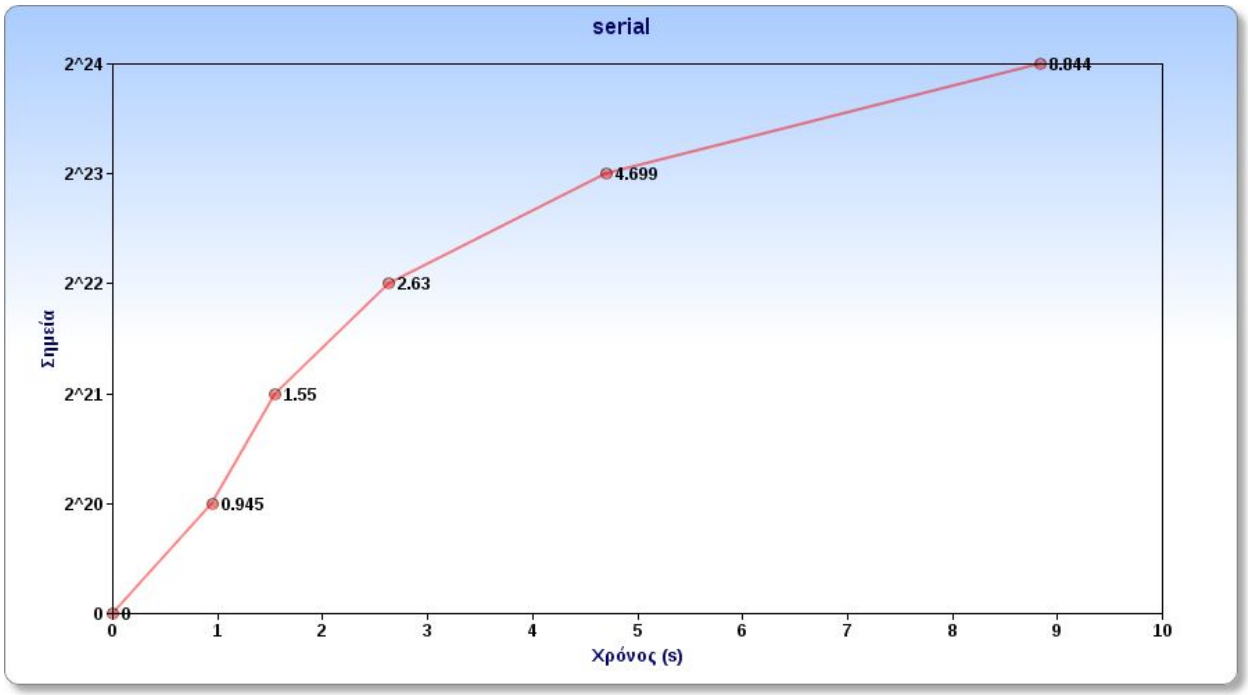
Δεσμεύω και μνήμη για τους pointers box (όλα τα κουτιά) και leaf (τα φύλλα μόνο).

Ακολουθεί το setup της ρίζας. Από κει και πέρα η κάθε μέθοδος ακολουθει διαφορετική διαδρομή όπως εξηγήθηκε παραπάνω.

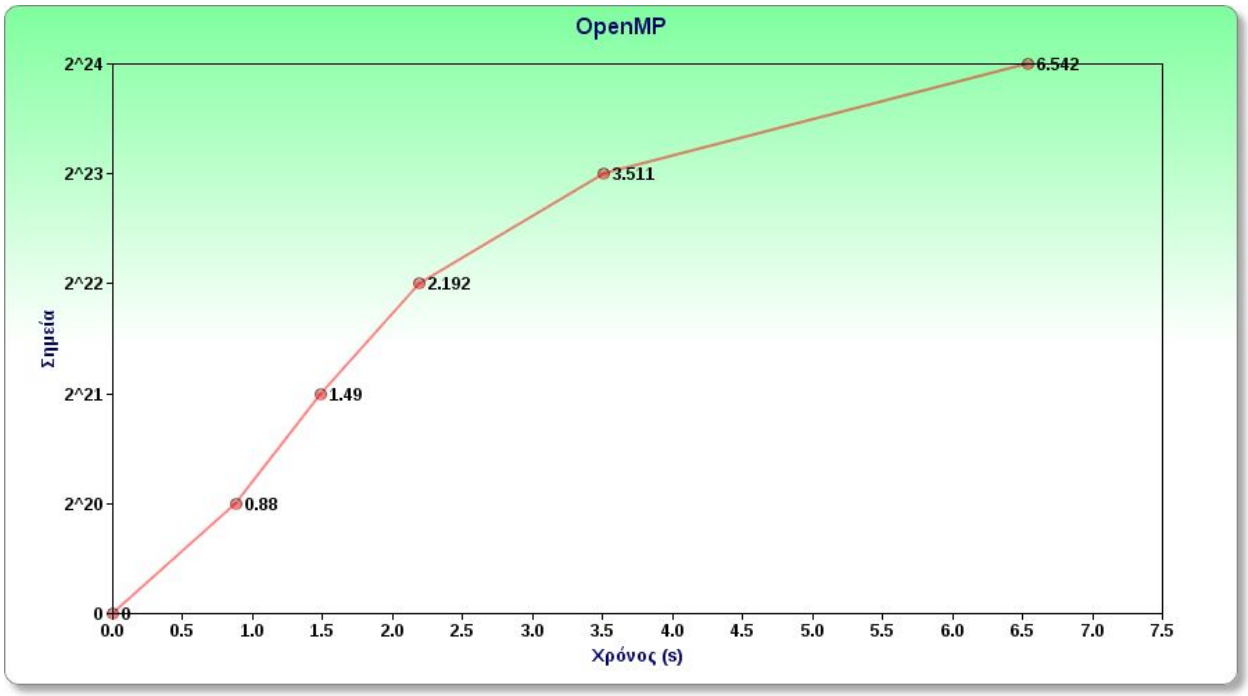
Τέλος, ο έλεγχος των αποτελεσμάτων έγινε με το κομμάτι κώδικα στο τέλος των αρχείων του πηγαίου, και τον έχω επισημάνει με σχόλιο για να μην εμποδίζει.

Τα διαγράμματα που προέκυψαν από τους χρόνους εκτέλεσης για τα δοσμένα Ν:

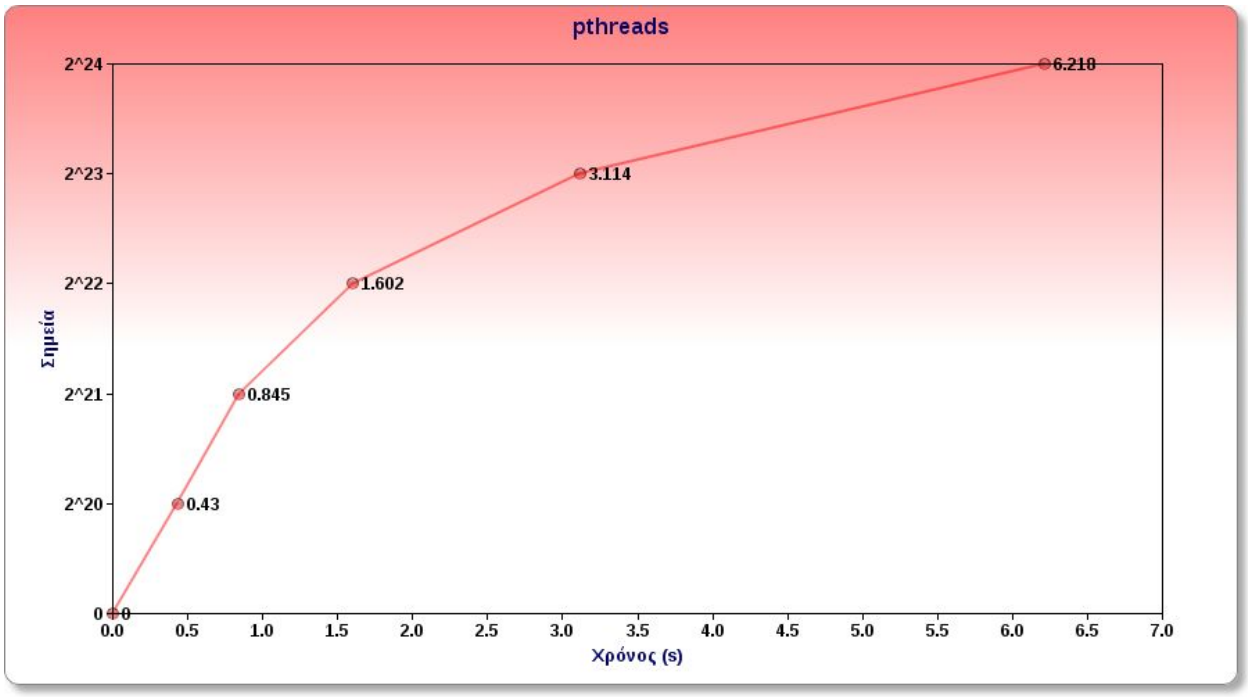
σειριακά:



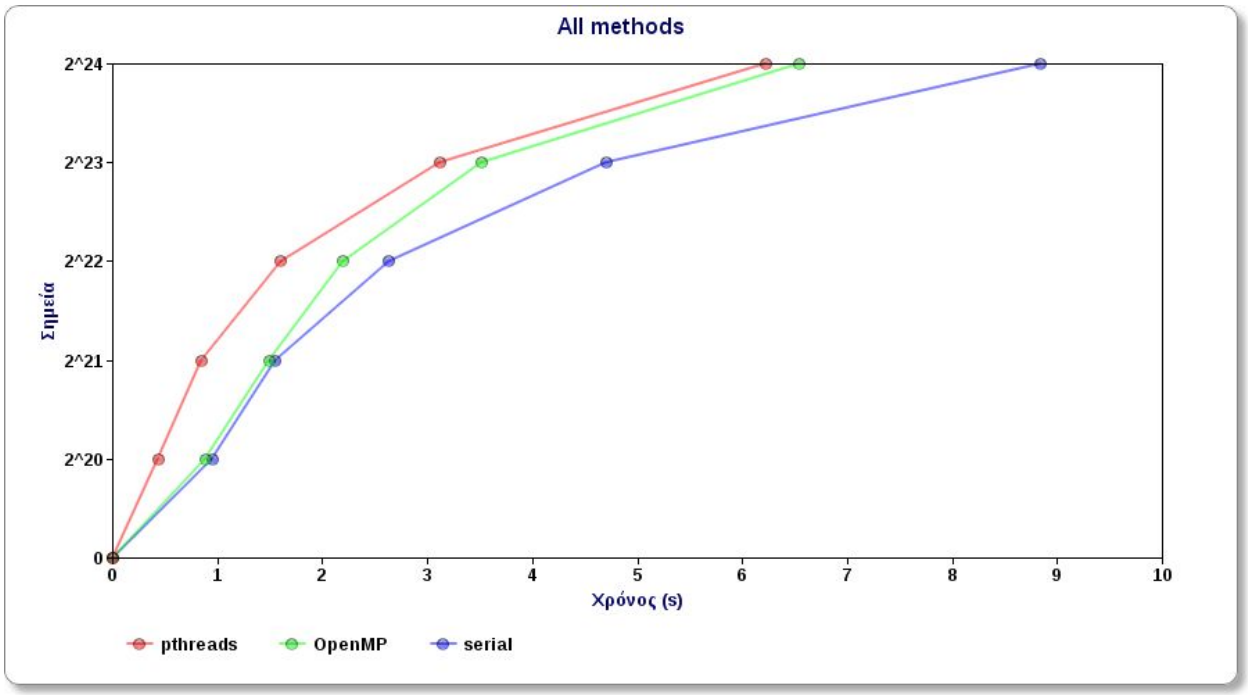
OpenMP:



pthreads:



και ένα συγκεντρωτικό:



Συμπέρασμα:

Από τα διαγράμματα είναι φανερό ότι επιτύχαμε αρκέτη μείωση του χρόνου εκτέλεσης και στις 2 παράλληλες υλοποιήσεις σε σχέση με την σειριακή.

Πιο συγκεκριμένα για N=2^24 είχαμε μείωση του χρόνου εκτέλεσης κατά 2,3 δευτ σε OpenMP και 2.65 δευτ σε Pthreads.

Ακόμη, είναι αξιοσημείωτο ότι για N=2^20 σε Pthreads είχαμε μείωση του χρόνου εκτέλεσης κατά 0.5 δευτ στα 0.95 δευτ σειριακού χρόνου εκτέλεσης (σχεδόν 2-πλάσια ταχύτητα!). Για ίδιο N σε OpenMP είχαμε πολύ μικρότερη μείωση (περίπου 0.06 δευτ).