Αναφορά

Παράλληλα και Διανεμημένα Συστήματα

Άσκηση 1

- **Εισαγωγή:** Σκοπός της εργασίας ήταν η παραλληλοποίηση του κώδικα που μας δόθηκε με τρία διαφορετικά πρότυπα παραλληλοποίησης σε Cilk,OpenMP,Pthreads(Ο κώδικας είναι γραμμένος σε γλώσσα C).Ο σειριακός αυτός κώδικας που μας δόθηκε, με την χρήση οκταδικών δέντρων επιτυγχάνει την ιεραρχική ομαδοποίηση N σωματιδίων στον τρισδιάστατο χώρο.Η διαδικασία γίνεται ως εξής:1)αρχικά μέσω της hash code γίνεται κατάτμηση της κάθε συντεταγμένης του κάθε σημείου βάσει ενός τύπου,2) στην συνέχεια χρησιμοποιείται μια μάσκα(Morton code ή αλλιώς Z κωδικοποίηση) μέσω της οποίας γίνεται αλληλένθεση των bits (γνωστό ως bit ιnterleaving) των τριών κωδικών κατάτμησης που παράχθηκαν από πριν για κάθε σημείο. Αυτό γίνεται για να αναπαραστήσουμε τις 3 συντεταγμένες x,y,z, ο τύπος των οποίων ειναι unsigned 32-bits integer σε μια μεταβλητή unsigned 64-bits integer,3) έπειτα γίνεται μερική διάταξη των κωδικών μέσω μιας αναδρομικής διαδικασίας,4) και τέλος γίνεται μια αναδιάταξη των σωματιδίων στην μνήμη, ώστε σωματίδια που ανήκουν στο ίδιο τμήμα να πιάνουν συνεχόμενες θέσεις μνήμης.

  Να σημειώσουμε επίσης ότι και τα τρία πρότυπα παραλληλισμού εφαρμόζονται σε πολυεπεξεργαστές με διαμοιραζόμενη μνημη.

- **OpenMP**: Το μοντέλο προγραμματισμού του OpenMP είναι βασισμένο στο πολυνηματικό μοντέλο παραλληλισμού. Να πούμε ότι δεν αποτελεί αυτοτελή γλώσσα αλλά είναι ένα add-on σε ήδη υπάρχουσες(C,C++,Fortran).Αρχικά, μια εφαρμογή γραμμένη σε OpenMP ξεκινά να τρέχει με ένα μονο νήμα, το master thread. Όταν το πρόγραμμα εισέρχεται σε μία περιοχή που έχει ορίσει ο προγραμματιστής να εκτελεστεί παράλληλα (parallel region,#pragma omp parallel), τότε δημιουργούνται αρκετά νήματα(fork-join μοντέλο) και το μέρος του κώδικα που εκτελείται στην παράλληλη περιοχή εκτελείται παράλληλα. Όταν ολοκληρωθεί ο υπολογισμός της παράλληλης περιοχής όλα τα νήματα

τερματίζουν και συνεχίζει μόνο το master thread.

Παρακάτω θα αναλύσουμε τις αλλαγές που κάναμε στο δοθέν πρόγραμμα:

- test_octree: Οι μόνες αλλαγές που έγιναν στην main, ήταν η προσθήκη μίας επιπλέον μεταβλητής(NUM_THREADS)(η οποία λαμβάνεται όταν καλείται το πρόγραμμα), έτσι ώστε ο user να μας λέει με πόσα threads θέλει να τρέξουμε τις παράλληλες περιοχές του κώδικα και μίας μεταβλητής (active_threads) που μας δείχνει ποσα ενεργά threads έχουμε, η οποία θα χρησιμοποιηθεί και θα αναλυθεί στο κομμάτι της radix.
    - hash_codes: Η παραλληλοποίηση της compute_hash_code έγινε στο κομμάτι της κλήσης της συνάρτησης quantize μιας και τα ύπολοιπα μερη ήταν ή critical ή δεν θα πρόσθεταν καμία επιπλέον βελτίωση στο πρόγραμμα μιας και η δημιουργία threads κόστιζει, γεγονός που μας οδηγεί στο οτι δεν δημιουργούμε threads σε περιοχές με μικρό όγκο δουλειάς και για λίγες επαναλήψεις. Στην guantize λοιπόν αυτό που κάναμε ήταν απλά να προσθέσουμε μία #pragma omp parallel for η οποία ειναι η συντόμευση μιας #pragma omp parallel και μιας #pragma omp for .Η πρώτη ορίζει την περιοχή του κώδικα που θέλουμε να εκτελέσουμε παράλληλα και δημιουργεί τα threads ανάλογα με τα πόσα είναι διαθέσιμα τη δεδομένη στιγμή στο σύστημα μας. Στη δική μας περίπτωση έρχεται να προστεθεί η επιθυμία του user με την εντολή num_threads(), όπου στην παρένθεση βάζουμε την μεταβλητή που έχουμε πάρει σαν είσοδο .Μετά αφήνουμε την δουλειά στην #pragma omp for η οποία ακολουθείται από ένα for loop και το μόνο που κάνει είναι να μοιράζει στα threads που έχουν δημιουργηθεί τις επαναλήψεις της for.Η #pragma omp for κάνει από μόνη της private τον μετρήτη του loop(π.χ i),που σημαίνει ότι κάθε thread έχει στο stack του ένα αντίγραφο του i και με βάση αυτό το i κάνει την δουλειά που του έχει ανατεθεί. Όλες οι άλλες μεταβλητές ειναι shared by default εκτός αν έχει διακυρηχθεί κάτι άλλο.
- morton_codes:Ισχύουν όσα αναφέρθηκαν προηγουμένως, με την διαφορά ότι η #pragma omp parallel for γίνεται απευθείας στην συνάρτηση morton_encoding και όχι σε κάποια άλλη που καλεί αυτή.

- radix_sort:Οι αλλαγές που έχουν γίνει εδώ είναι εμφανώς περισσότερες από τα άλλα κομμάτια, μιας και η συνάρτηση μας είναι αναδρομική. Θα αρχίσουμε με τις μεταβλητές που δηλώσαμε. Καταρχήν κάναμε extern τις μεταβλητές NUM_THREADS, active_threads για να φαίνονται και σε αυτό το αρχείο. Ακόμα, δηλώσαμε τις μεταβλητές flag_nested_thread,create: η πρώτη για να ελέγξουμε αν μπορούμε να δημιουργήσουμε άλλο nested thread (η τιμή 1 αν μπορούμε και 0 αν δεν μπορούμε) και η δεύτερη για να μην δημιουργούμε σε κάθε αναδρομή στην παράλληλη περιοχή threads ίσα με τον αριθμό NUM_threads, γιατί μετά από κάποιες επαναλήψεις θα κράσαρε το σύστημα μας ελλείψει άλλων διαθέσιμων resourses για τη δημιουργία άλλων threads. Μεταβαίνοντας στην υλοποίηση, βλέπουμε ότι η συνάρτηση καλείται σειριακά. Μπαίνει λοιπόν, το master thread στην truncated_radix_sort και εκτελεί τον κώδικα. Η πρώτη αλλαγή είναι η χρησιμοποίηση της #pragma omp flush(active_threads), η οποία παρέχει ένα συνεπές στιγμιότυπο της μνήμης για την μεταβλητή που ζητήσαμε. Χρησιμοποιείται για συγχρονισμό.Στο σημείο που την βάζουμε η τρέχουσα τιμή μίας μοιραζόμενης μεταβλητής γράφεται άμεσα από την cache στην μνήμη (write back).Αυτό γίνεται για να έχουν όλα τα threads την ίδια εικόνα της μνήμης για όλες τις shared μεταβλητές. Στην συνέχεια κάνουμε κάποιους ελέγχους που σχετίζονται με το πόσα ενεργά threads έχουμε και ανάλογα με την περίπτωση μεταβάλλουμε το flag και το create. Έπειτα, κάνουμε πάλι flush την flag για να την βάλουμε στην συνάρτηση omp_set_nested(), η οποία αν παίρνει τιμή 0 δεν μας αφήνει να δημιουργήσουμε άλλα εμφωλευμένα threads, ενώ αν παίρνει οποιαδήποτε άλλη τιμή μας αφήνει να δημιουργήσουμε εμφωλευμένα threads. Μετά μπαίνουμε στην παράλληλη περιοχή - εδώ να πούμε ότι η #pragma omp parallel κάνει flush όλες τις shared μεταβλητές και στην είσοδο και στην έξοδο. Στην συνέχεια συναντάμε μια #pragma omp for η οποία έχουμε εξηγήσει τί κάνει, ας διευκρινίσουμε όμως ότι βάζουμε επιπλέον τις οδηγίες nowait και schedule(static). Η πρώτη βγάζει το barrier (το οποίο περιμένει ολα τα threads που έχουν δημιουργηθεί να φτάσουν σε αυτό το σημείο, ώστε σε δεύτερη φάση να συνεχίσει το πρόγραμμα) που έχει η for by default, γιατί θέλουμε τα threads να δρούν ανεξάρτητα και να μην τα περιμένουμε από τη στιγμή που μεταβάλλουν διαφορετικές θέσεις μνήμης. Η δεύτερη είναι μία οδηγία για το πώς θα γίνει ο διαμοιρασμός

των επαναλήψεων. Τέλος, καλούμε αναδρομικά τη συνάρτηση και αυξάνουμε ή μειώνουμε τα active_threads αναλόγως με το αν έχουμε το nested on ή off(το παίρνουμε με το omp_get_nested()), δηλαδή αν μπορούμε να δημιουργήσουμε άλλα threads ή όχι.

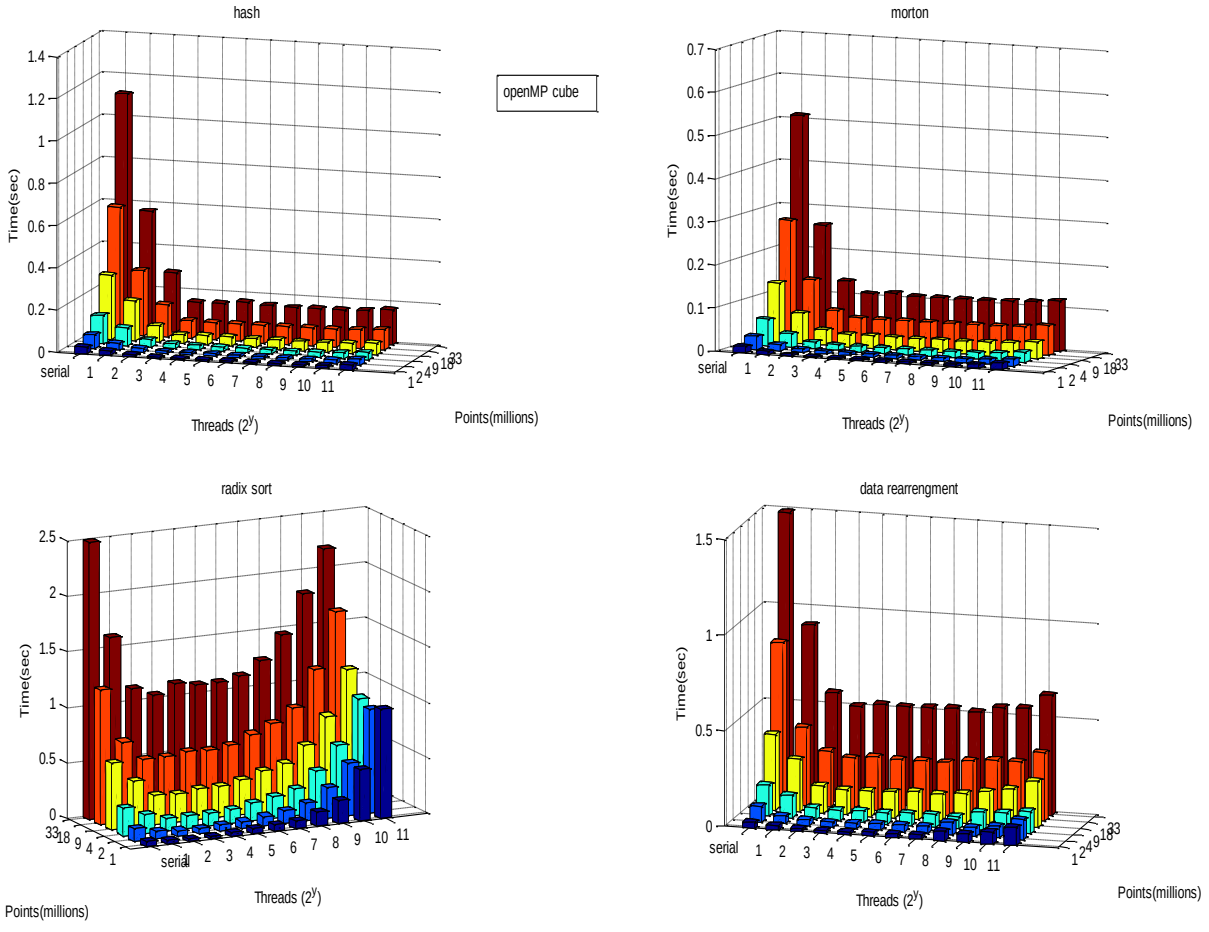
- data_rearrangement: Ισχύει ακριβώς ότι ισχύει στην morton_encoding.

Διαγράμματα/σχόλια:

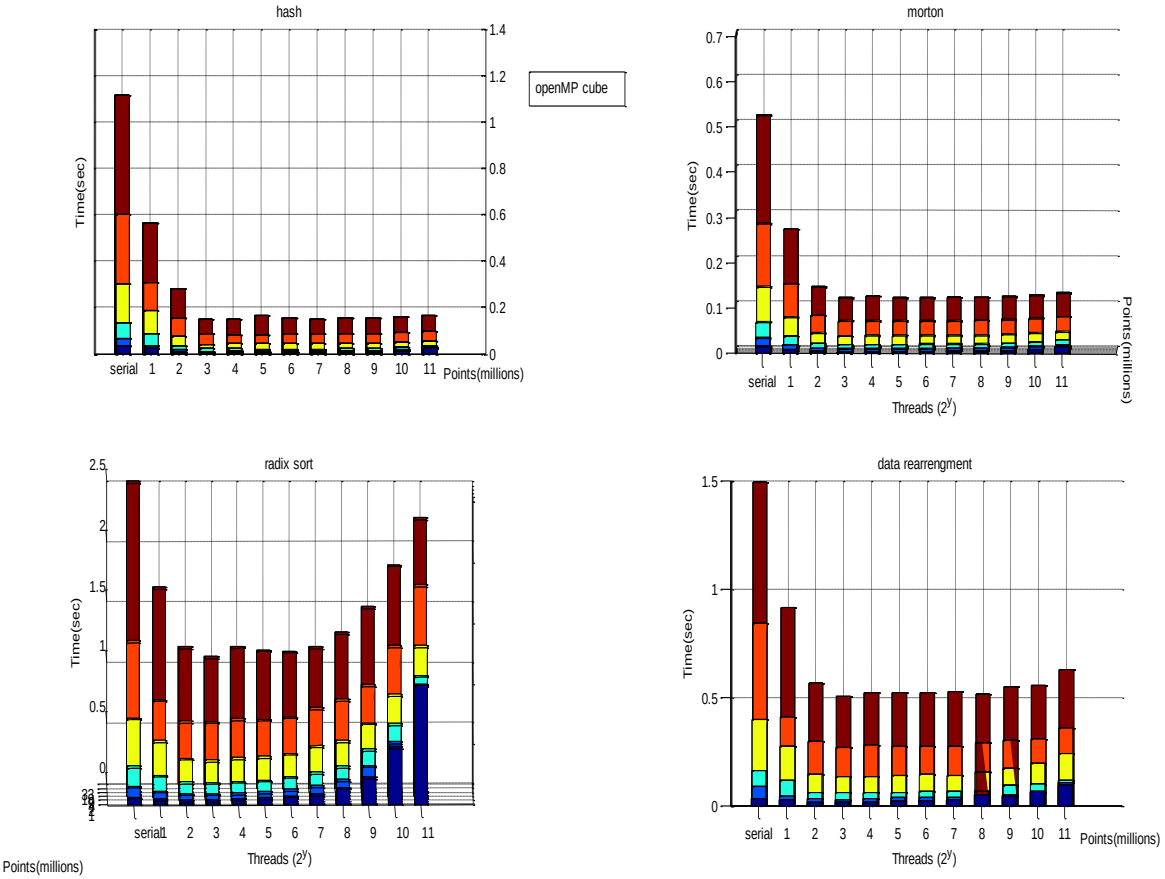
Παρακάτω φαίνονται τα διαγράμματα χρόνου του παραλληλοποιημένου προγράμματος συναρτήσει των threads και σημειών, με σταθερό το βάθος του δέντρου L=18 και το όριο πληθυσμού S=128, όπως μας ζητήθηκε και για δύο κατανομές κύβου και σφαίρας. Στην πρώτη γραμμή φαίνεται ο σειριακός και στις υπόλοιπες ο παράλληλος συναρτήσει των threads.

Για κύβο:

από διάφορες οπτικές:



και σε κάτοψη:



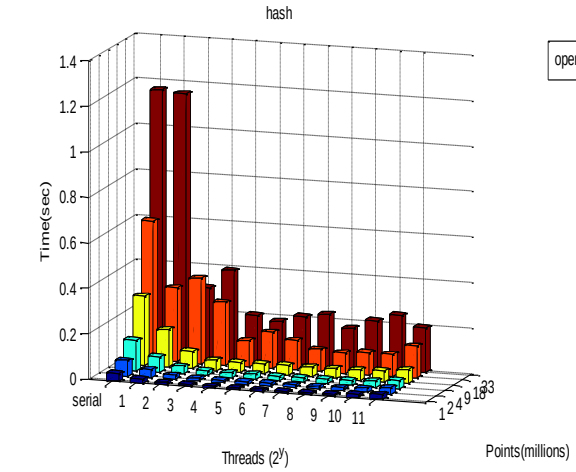
Σχολιασμός: Παρατηρούμε οτι όσο αυξάνονται τα threads σχεδόν γραμμικά μειώνεται και ο χρόνος(για μικρό αριθμό threads,μετά η βελτίωση δεν είναι τόσο εμφανής) μέχρι τα resourses που έχουμε στον υπολογιστή που κάναμε τις μετρήσεις να ειναι μικρότερα του ζητούμενου αριθμού όπου και σταθεροποιείται ο χρόνος. Οι μετρήσεις έγιναν σε ένα 8πύρινο σύστημα και αυτό φαίνεται στα διαγράμματα, αφού πέρα των 8threads ο χρόνος αυξάνεται λίγο και μετά σταθεροποιείται και αυτό συμβαίνει γιατί η OpenMP αναγνωρίζει πόσα threads έχει το σύστημα και ρίχνει τον αριθμό των threads που δημιουργούμε στο default στην περίπτωση μας 8(συνήθως όσο και ο αριθμός των πυρήνων).Αυτα συμβαίνουν σε όλες τις συναρτήσεις εκτός

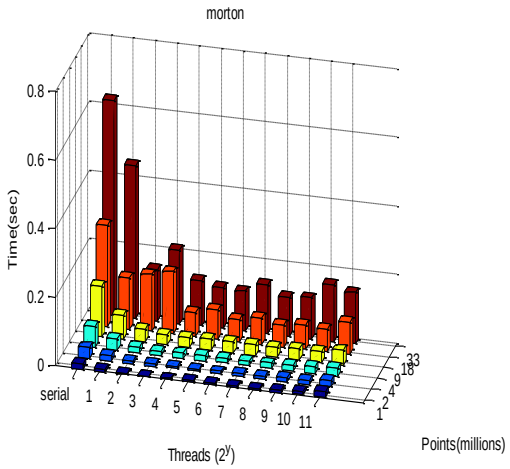
απο την radix_sort η οποία λόγω αναδρομής μας χαλάει τα σχέδια. Μιας και δεν μπορεί να ελέγξει απο μόνη της η OpenMP πόσα threads θα δημιουργηθούν. Παρατηρούμε λοιπόν, ότι μετά τα 8 threads ο χρόνος χειροτερεύει και αυτό συμβαίνει λόγω του overhead που προσθέτει στο σύστημα η δημιουργία νέων threads. Όταν έχουμε παραπάνω threads από τα cpus τότε κάθε πυρήνας διαχειρίζεται πολλά threads και να εναλλάσει συνέχεια τον έλεγχο του προγράμματος σε αυτά τα threads, το οποίο εκτός από την καθυστέρηση που προκαλεί επιβαρύνει και την μνήμη(Ram,cache). Αυτά ακριβώς που προηγήθηκαν γίνονται αντιληπτά στα παραπάνω διαγράμματα και ιδιαίτερα σε μικρό αριθμό N που οι χρόνοι και το φόρτο εργασίας ειναι πολύ μικρότερα.

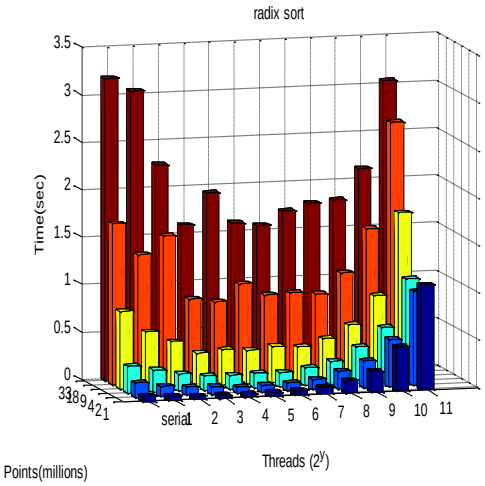
Παρόμοια είναι τα σχόλια για την περίπτωση της σφαίρας. Για να δείτε καλύτερα τους χρόνους και τις διαφορές μπορείτε να ανατρέξετε στα .fig αρχεία που παρατίθενται στην εργασία.

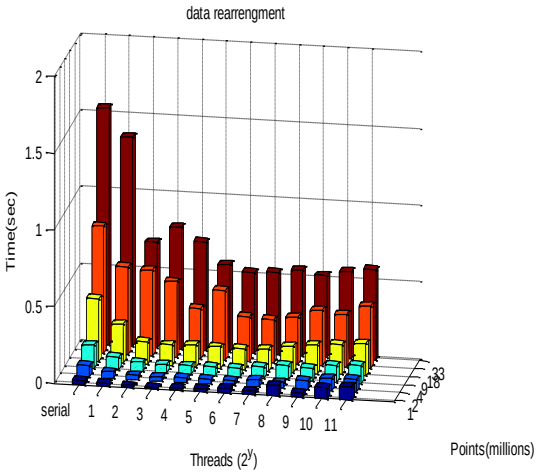
Σε σφαίρα:

από διάφορες οπτικές:

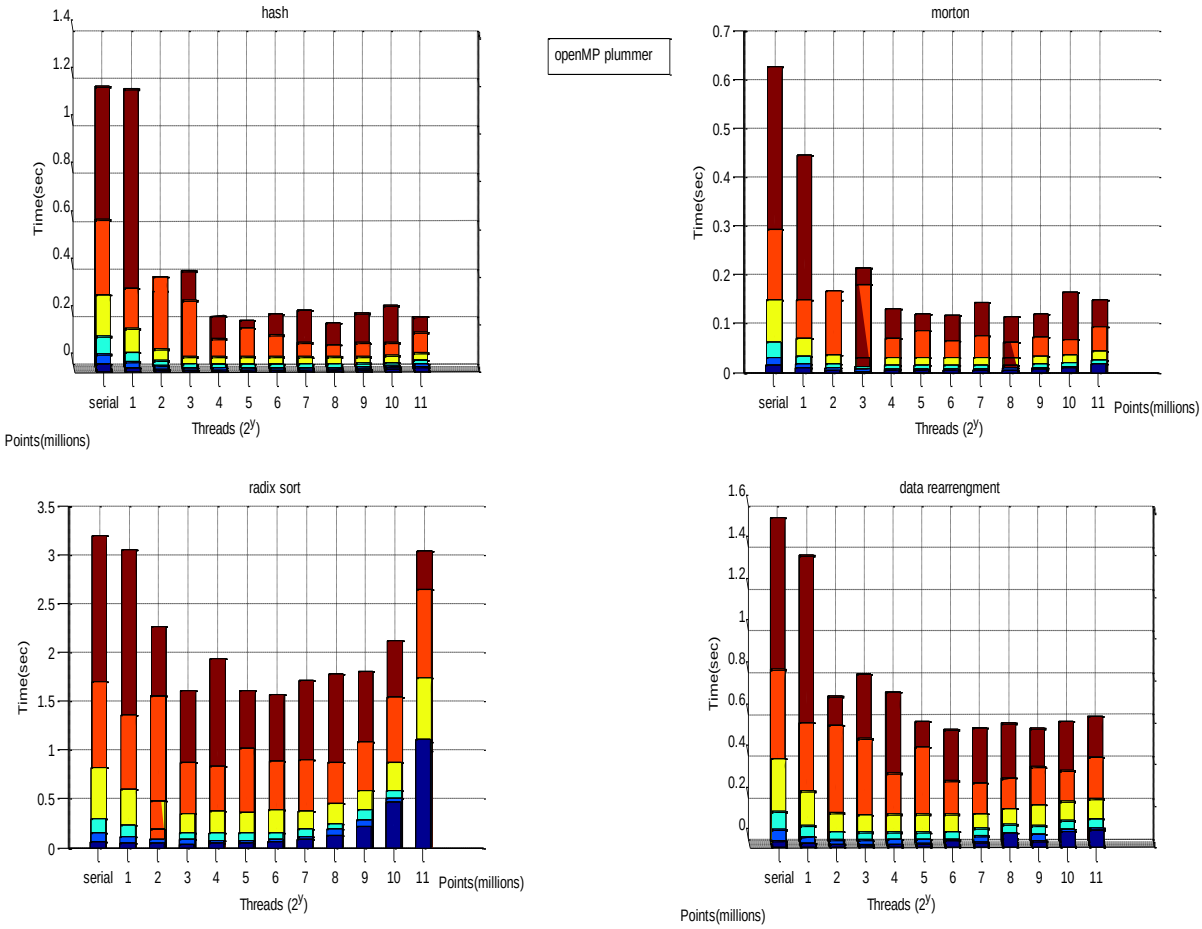








και σε κάτοψη:



- **Cilk:** Η cilk είναι και αυτή ένα add-on στις ήδη υπάρχουσες γλώσσες C,C++,Fortran. Σε αντίθεση με την OpenMP δεν βασίζεται στο fork-join μοντέλο αλλά στο work-stealing το οποίο ουσιαστικά σημαίνει ότι όταν ένας επεξεργαστής τελειώσει την δουλεία του θα πάει να κλέψει το πρώτο task από την ουρά των tasks ενός άλλου επεξεργαστή. Task στην ουρά του κάθε επεξεργαστή προστίθεται με την cilk_spawn, η οποία εξαναγκάζει τον compiler να δημιουργήσει έναν κώδικα για το σκοπό αυτό. Ένα επιπλέον και πολύ ενδιαφέρον χαρακτηριστικό είναι ότι τα statements της Cilk δεν εξαναγκάζoun τον compiler να παραλληλοποιήσει το πρόγραμμα, απλά του λέει ότι το συγκεκριμένο κομμάτι μπορεί να παραλληλοποιηθεί (serial semantics). Η λογική εδώ είναι παρόμοια με την OpenMP. Και για αυτό δεν χρειάστηκε να κάνουμε πολλές αλλαγές στον κώδικα.

  Παρακάτω θα αναλύσουμε τις αλλαγές που κάναμε στο δοθέν πρόγραμμα:

  - test_octree: Οι αλλαγές που έγιναν εδώ αφορούν όπως και στην OpenMP να παίρνουμε σαν είσοδο τα threads(workers η ονομασία στην

περίπτωση της Cilk) που ζητάει ο user. Και στην συνέχεια με την εντολή *__cilkrts_set_param() να τα δημιουργούμε σε συνδυασμό πάντα και με τα διαθέσιμα του συστήματος όπως περιγράψαμε στην OpenMP.*

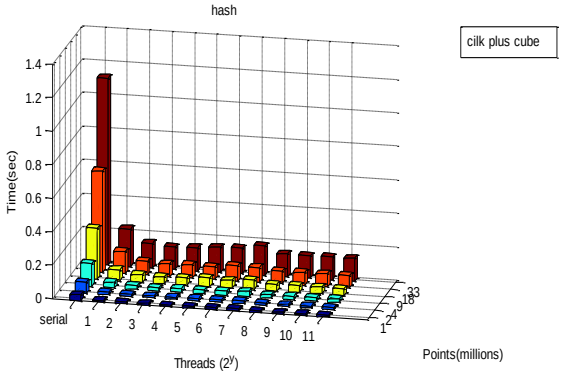
- *hash_codes: Εδώ με την ίδια λογική με την OpenMP παραλληλοποιήσαμε την quantize χρησιμοποιώντας την cilk_for, η οποία δηλώνει ότι το for loop μπορεί να παραλληλοποιηθεί.*
- *morton_codes: Ισχύουν τα ίδια με την hash_codes απλά παραλληλοποιήθηκε κατευθείαν η αρχική συνάρτηση morton_encoding.*
- *radix_sort: Στην περίπτωση της cilk σε αντίθεση με την OpenMP δεν χρειάστηκε να περιπλέξουμε τα πράγματα τόσο πολύ μιας και θέτοντας τους workers από την test_octree μας εξασφαλίζει ότι οι μέγιστοι workers που πρέπει να χρησιμοποιηθούν είναι συγκεκριμένοι και δεν πέφτει στην παγίδα της αναδρομής,δηλαδή κάθε φορά που καλείται αναδρομικά η συνάρτηση να δημιουργούνται workers ίσοι με τον αριθμό της εισόδου των workers που πήραμε.*
- *data_rearrangement: Ισχύουν τα ίδια με την morton_codes.*

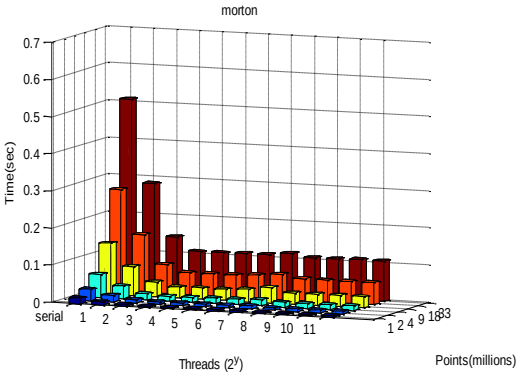
Διαγράμματα/σχόλια:

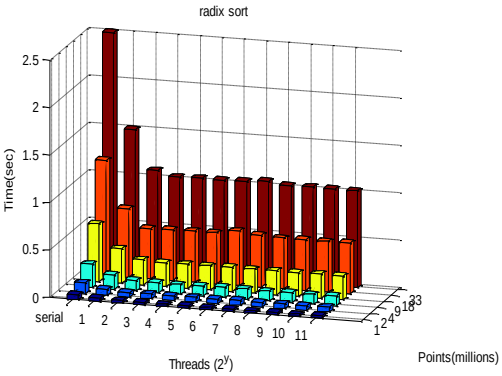
Παρακάτω φαίνονται τα διαγράμματα χρόνου του παραλληλοποιημένου προγράμματος συναρτήσει των threads και σημειών, με σταθερό το βάθος του δέντρου L=18 και το όριο πληθυσμού S=128, όπως μας ζητήθηκε και για δύο κατανομές κύβου και σφαίρας. Στην πρώτη γραμμή φαίνεται ο σειριακός και στις υπόλοιπες ο παράλληλος συναρτήσει των threads.

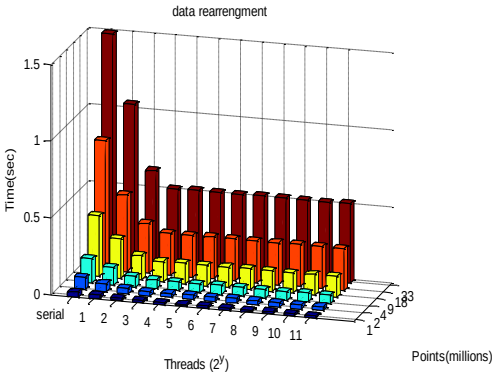
Για κύβο:

απο διάφορες οπτικές:

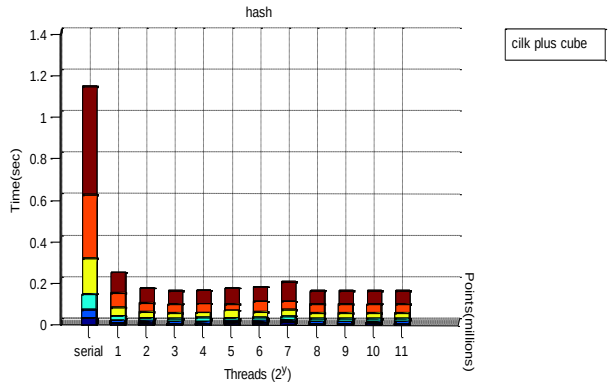

hash
cilk plus cube
Time(sec)
Threads (2^y)
Points(millions)

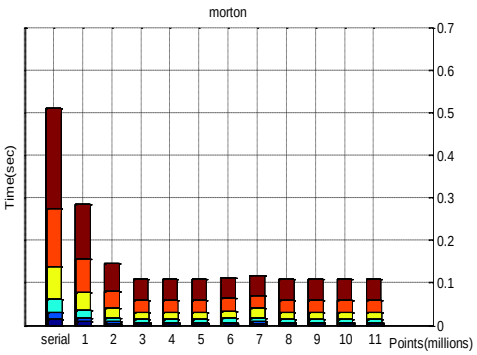

morton
Time(sec)
Threads (2^y)
Points(millions)

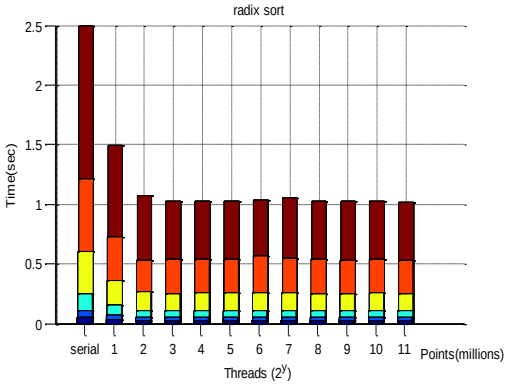

radix sort
Time(sec)
Threads (2^y)
Points(millions)

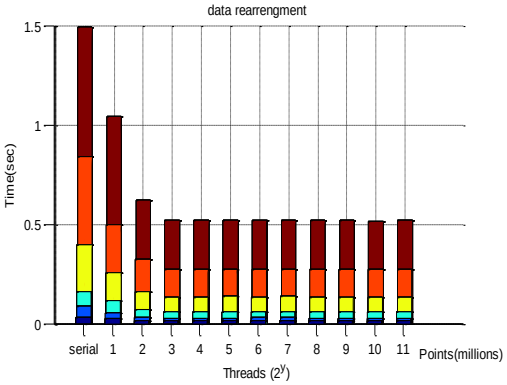

data rearrengment
Time(sec)
Threads (2^y)
Points(millions)

και η κάτοψη:


hash
cilk plus cube
Time(sec)
Threads (2^y)
Points(millions)


morton
Time(sec)
Threads (2^y)
Points(millions)


radix sort
Time(sec)
Threads (2^y)
Points(millions)

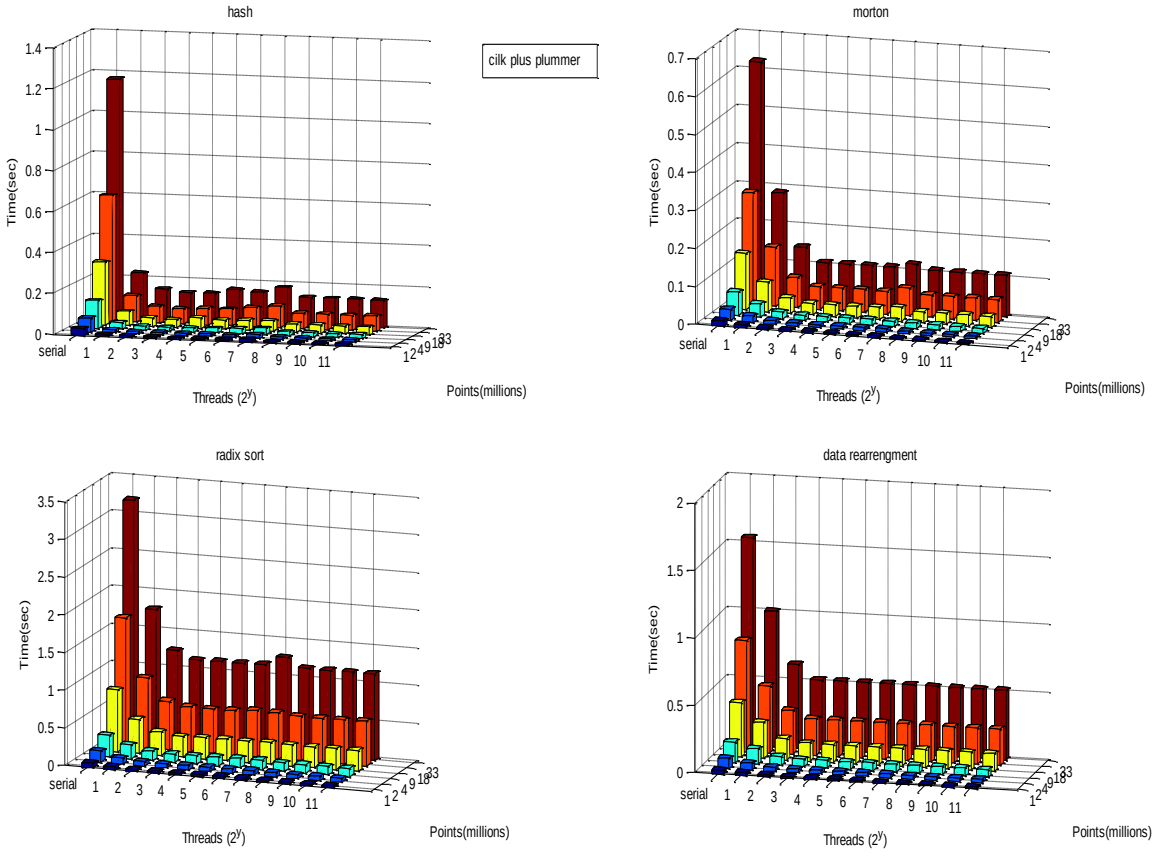

data rearrengment
Time(sec)
Threads (2^y)
Points(millions)

Σχολιασμός: Ο σχολιασμός της Cilk δεν διαφέρει σχεδόν σε τίποτα με την OpenMP όπως μπορούμε να αντιληφθούμε και από τα διαγράμματα.Η μόνη διαφορά έγκειται στην radix_sort. Όπως βλέπουμε λοιπόν υπάρχει και εδώ μια σχεδόν γραμμική μείωση του χρόνου μέχρι τα 8 threads που έχει το σύστημα μας και μετά μια μικρή αύξηση και σταθεροποίηση για τους λόγους που αναφέραμε στην OpenMP. Στην περίπτωση της radix_sort τώρα γίνεται το ίδιο με τις υπόλοιπες συναρτήσεις παρόλη την παρουσία την αναδρομής και αυτο συμβαίνει για το λόγο που αναφέραμε στο σχολιασμό του κώδικα. Μία Τελευταία παρατήρηση είναι η θεαματική μείωση του χρόνου(super linear) στην hash_codes με δύο threads, που είναι ένα δείγμα για το πόσο καλά διαχειρίζεται τα threads η Cilk.

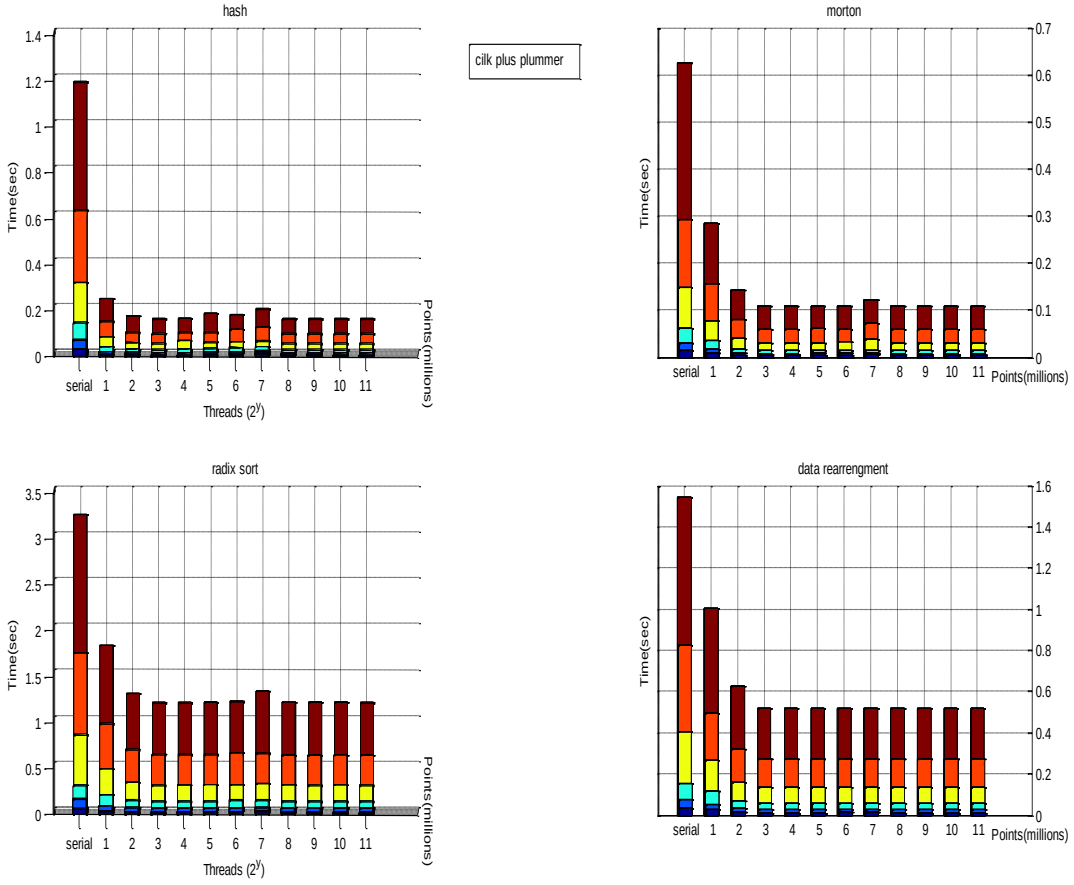
Παρόμοια είναι τα σχόλια για την περίπτωση της σφαίρας.

Σε σφαίρα:

από διάφορες οπτικές:



και η κάτοψη:



- ***Pthreads:*** *Τα Pthreads είναι ένα πρότυπο χαμηλότερου επιπέδου σε σχέση τα πρότυπα που αναφέραμε προηγουμένως. Και αυτός είναι και ο λόγος που το καθιστά και πιο δύσκολο/περίπλοκο σε σχέση με τα προηγούμενα.*

Παρακάτω θα αναλύσουμε τις αλλαγές που κάναμε στο δοθέν πρόγραμμα:

- test_octree: Οι αλλαγές που έγιναν εδώ ήταν όπως στις άλλες δυο περιπτώσεις να περνάει σαν είσοδο ο αριθμός των threads που θέλει ο user να χρησιμοποίησει το σύστημα(επιπλέον του master thread εξηγείται παρακάτω η διαφορά). Ακόμα φτιάχτηκαν δυο struct τα οποία

περνάν ως όρισμα στις κλήσεις των συναρτήσεων morton_encoding,data_rearrangement. Δημιουργήθηκε πίνακας τύπου pthread_t με μέγεθος όσο τα threads που μας ζητάει ο user. Επίσης δημιουργήσαμε μία μεταβλητή pthread_attr_t, την αρχικοποιήσαμε και της δώσαμε την ιδιότητα να κάνει τα thread joinable. Στην συνέχεια δημιουργήσαμε δύο for loops στα οποία περάσαμε τιμές στους πίνακες τύπου struct που δηλώσαμε για να έχει κάθε thread τα δικά του δεδομένα. Έπειτα μέσα σε αυτά τα loops,καλέσαμε την pthread_create , για να δημιουργήσουμε τόσα threads όσα ο αριθμός που μας δόθηκε, τα οποία εκτελούν την ρουτίνα που δείχνει η διεύθυνση που περάσαμε σαν όρισμα. Τα άλλα ορίσματα ήταν οι ιδιότητες του νήματος, το struct και ένα μοναδικό αναγνωριστικό (ID) που επιστρέφει η υπορουτίνα για κάθε καινούριο thread. Αυτό έγινε για τις περιπτώσεις της morton_enconding και της data_rearrangement, οι άλλες δύο καλούνται σειριακά και παραλληλοποιούνται στο εσωτερικό τους. Τέλος, μετά τα for loops που αναφέραμε παραπάνω καλούμε την συνάρτηση pthread_join, η οποία αναστέλει το thread συντονιστή μέχρι να τερματιστούν όλα τα thread εργαζόμενοι. Εδώ να προσθέσουμε ότι σε αντίθεση με την OpenMP το master thread δεν κάνει δουλειά μαζί με τα άλλα thread απλά τα δημιουργεί.

- hash_codes: Στην συνάρτηση αυτή κάνουμε ότι κάναμε για τις morton_encoding και data_rearengement στην test_octree. Δηλαδή δημιουργούμε ένα struct και καλούμε την pthread_create με τον ίδιο τρόπο την quantize αφού έχει εκτελεστεί σειριακά το προηγούμενο μέρος της hash_code.

- Morton_codes,data_rearragement: Αναφέραμε την διαδικασία στην test_octree. Απλά να πρόσθεσουμε ότι σπάμε το loop με τέτοιο τρόπο ώστε κάθε thread με ένα συγκεκριμένο id να κάνει συγκεκριμένη δουλεία.(Το id του κάθε thread περνάει μέσω του struct)

- radix_sort: Για ακόμα μία φορά τα πράγματα περιπλέκονται αρκετά για αυτή τη συνάρτηση. Αρχικά καλείται η σειριακή συνάρτηση που το μόνο που κάνει είναι να περνάει τα ορίσματα της μέσω ενός structure στην παραλληλοποιήμενη συνάρτηση και νατην περιμένει να τελειώσει.(η συνθήκη είναι τα active_threads να είναι 0 ή μικρότερα). Καλείται λοιπόν η παράλληλη συνάρτηση την πρώτη φορά καλείται σειριακά και μπαίνει στην τελευταία for στην οποία βρίσκεται και η

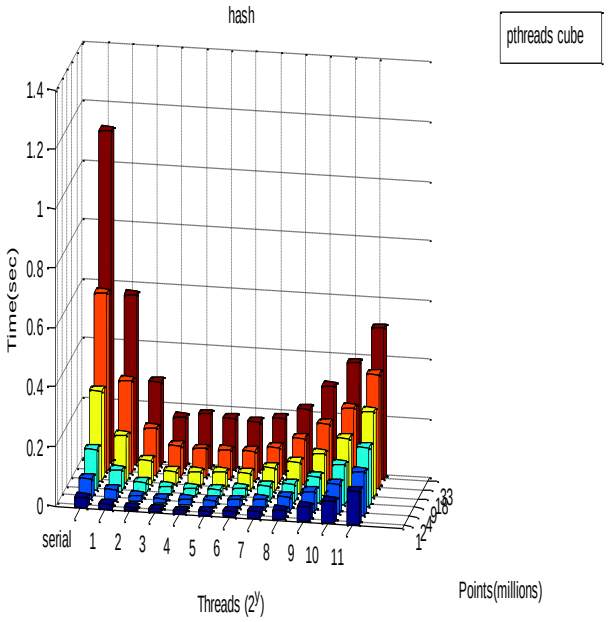
αναδρομή. Εκεί ελέγχοντας μία συνθήκη(αν τα active threads είναι μεγαλύτερα από τον αριθμό των threads που θέλουμε να δημιουργήσουμε) καλούμαι αναδρομικά τη συνάρτηση με νέο thread με την pthread_create και αυξάνουμε τα active threads ή συνεχίζουμε την αναδρομική της κλήση με το ήδη υπάρχον.Τα thread που δημιουργούμαι τα κάνουμε detached όπως έγινε και στην OpenMP μιας και το καθένα επεξεργάζεται διαφορετική θέση μνήμης και δεν χρειάζεται να το περιμένουμε. Το struct που περνάμε σε κάθε thread περιέχει μία μεταβλητή η οποία μας λέει αν το thread έχει πατέρα και αυτή η μεταβλητή είναι που μας βοηθάει να μειώνουμε τα active thread, ελέγχοντας αν έχει πατέρα ή όχι. Για την αυξομείωση της μεταβλητής active_threads χρησιμοποιείται mutex (pthread_mutex_lock, pthread_mutex_unlock) για να μην υπάρχουν race conditions μεταξύ των threads μίας και η μεταβλητή είναι global.

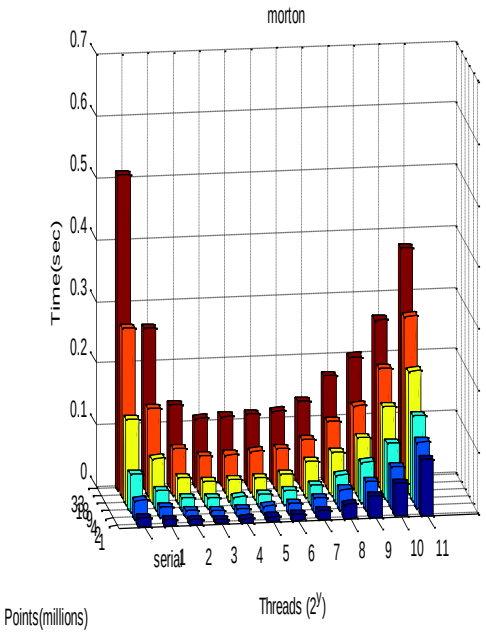
Διαγράμματα/σχόλια:

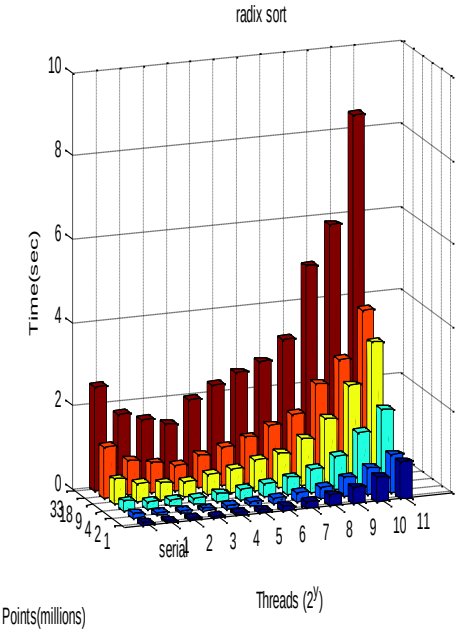
Παρακάτω φαίνονται τα διαγράμματα χρόνου του παραλληλοποιημένου προγράμματος συναρτήσει των threads και σημειών, με σταθερό το βάθος του δέντρου L=18 και το όριο πληθυσμού S=128, όπως μας ζητήθηκε και για δύο κατανομές κύβου και σφαίρας. Στην πρώτη γραμμή φαίνεται ο σειριακός και στις υπόλοιπες ο παράλληλος συναρτήσει των threads.

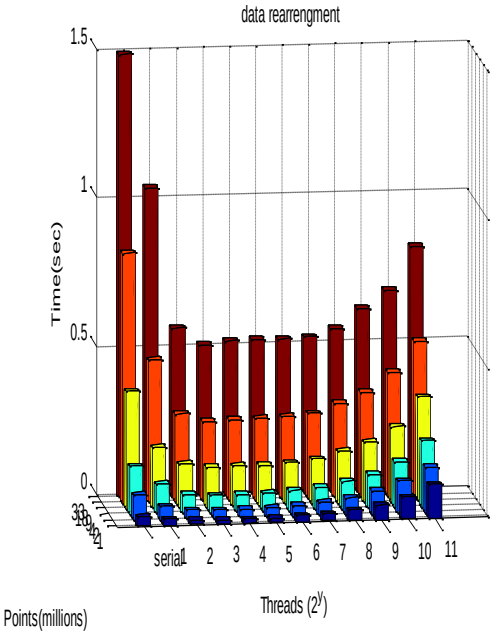
Για κύβο:

από διάφορες οπτικές:

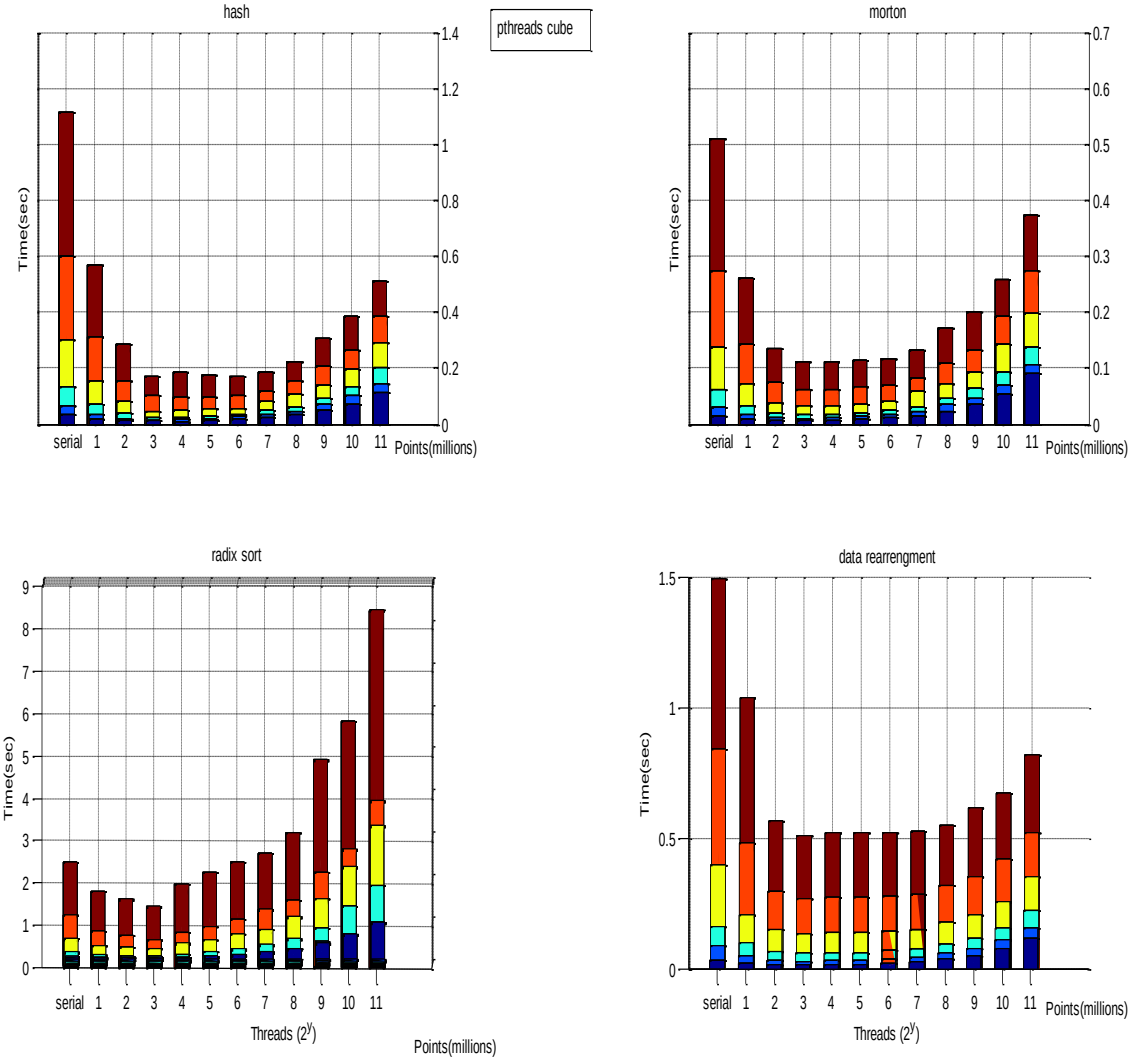








και η κάτοψη:

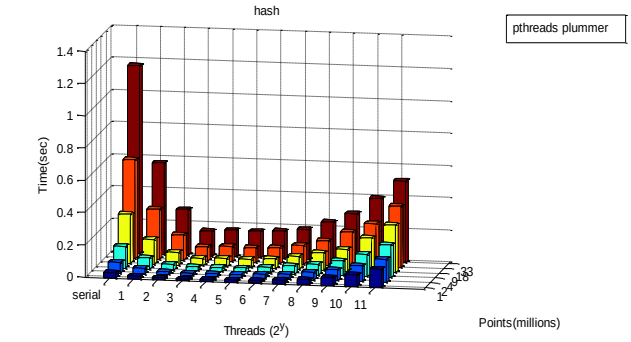


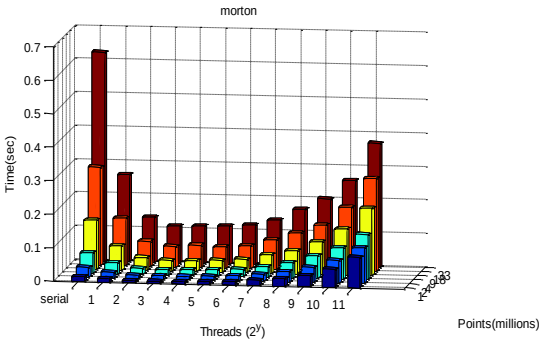
Σχολιασμός: Σε αντίθεση με τα άλλα δυο πρότυπα βλέπουμε ότι η Pthreads δεν αλλάζει τα threads που θα χρησιμοποιήσουμε σε default και για αυτό βλέπουμε μία δραματική αύξηση του χρόνου για πολλά threads και γίνεται ακόμα πιο έντονη στην radix_sort λόγω και των mutex και του μεγαλύτερου φόρτου εργασίας. Εξηγήσαμε γιάτι συμβαίνει αυτή η αύξηση στο χρόνο στην OpenMp στην radix_sort είναι ο ίδιος λόγος που συμβαίνει και εδώ όταν ξεπερνάμε τα threads του συστήματος που χρησιμοποιούμε. Κατά τα άλλα βλέπουμε και εδώ μια σχεδόν γραμμική μείωση του χρόνου μέχρι τα 8 Threads.

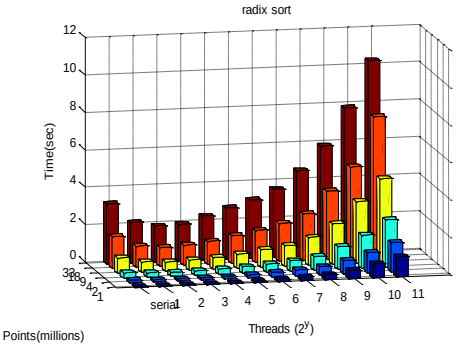
Τα παραπάνω ισχύουν και στην περίπτωση της σφαίρας.

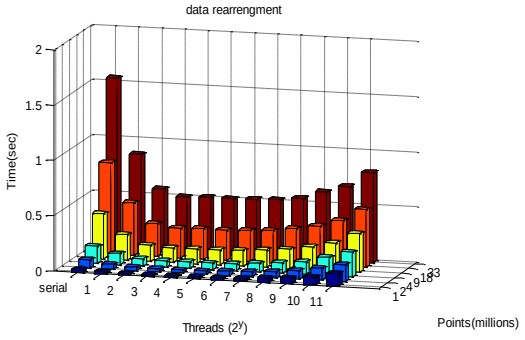
Σε σφαίρα:

από διάφορες οπτικές:









και σε κάτοψη: