Παράλληλα και Διανεμημένα – Εργασία 1
Σιώης Πολυζώης 8535

Χαρακτηριστικό του αλγορίθμου bitonic είναι ότι σπάει το πρόβλημα της διάταξης ενός πλήθους στοιχείων σε ανεξάρτητα κομμάτια του προβλήματος. Μπορούμε λοιπόν να εκμεταλλευτούμε το χαρακτηριστικό αυτό ώστε να μετατρέψουμε την σειριακή υλοποίηση του bitonic σε παράλληλη, βελτιώνοντας κατά πολύ την ταχύτητά του.

Στη συνέχεια θα παρουσιαστούν οι εξής παράλληλες εκδοχές του αλγορίθμου :

- pthreads, επαναληπτική
- CilkPlus, αναδρομική και επαναληπτική
- openMP, αναδρομική και επαναληπτική

Προσπάθεια κατανόησης αναδρομικού και επαναληπτικού bitonic

Επαναληπτικός

Για μια εποπτική εξέταση του αλγορίθμου χρησιμοποίησα printf() σε κάθε ένα από τα 3 εμφολευμένα for loops(της impBitonicSort()) και τον έτρεξα για $2^4 = 16$ στοιχεία.
Πήρα την εξής έξοδο :

```
-- k=2
---- j=1
------ i=0 / ij=1 / ij>i so comparing a[0] with a[1] / i&k=0
------ i=1 / ij=0
------ i=2 / ij=3 / ij>i so comparing a[2] with a[3] / i&k=2
------ i=3 / ij=2
-- k=4
---- j=2
------ i=0 / ij=2 / ij>i so comparing a[0] with a[2] / i&k=0
------ i=1 / ij=3 / ij>i so comparing a[1] with a[3] / i&k=0
------ i=2 / ij=0
------ i=3 / ij=1
---- j=1
------ i=0 / ij=1 / ij>i so comparing a[0] with a[1] / i&k=0
------ i=1 / ij=0
------ i=2 / ij=3 / ij>i so comparing a[2] with a[3] / i&k=0
------ i=3 / ij=2
Imperative wall clock time = 0.000051
 TEST PASSed
```

Συμπέρανα λοιπόν (γενικεύοντας) ότι για 2^q στοιχεία ο εξωτερικός βρόγχος τρέχει $\log_2(q)$ φορές, ο μεσαίος τρέχει $\log_2(k)$ για κάθε τιμή του k και ο εσωτερικός βρόχος τρέχει 2^q φορές για κάθε τιμή του j . Το σημαντικό είναι ότι για μια συγκεκριμένη τιμή του j καμιά από τις συγκρίσεις και αντιμεταθέσεις που πραγματοποιούνται στον εσωτερικό βρόγχο δεν ακουμπά στοιχείο που έχει ήδη προσπελαστεί .Αυτό σημαίνει ότι οι επαναληπτικές συγκρίσεις και αντιμεταθέσεις που πραγματοποιούνται για συγκεκριμένο j είναι ανεξάρτητες μεταξύ τους και μπορούν να παραλληλοποιηθούν (να ανατεθούν δηλαδή σε ξεχωριστά threads).
Αν τα threads (T) είναι περισσότερα ή όσα και τα στοιχεία (Q) τότε κάθε εσωτερική επανάληψη θα ανατεθεί σε ξεχωριστό thread, ενώ αν T<Q κάθε thread(που θα εκτελείται παράλληλα με τα υπόλοιπα T-1) θα αναλάβει Q/T επαναλήψεις (που θα εκτελούνται σειριακά).

Αναδρομικός

Η recBitonicSort() καλεί τον εαυτό της δύο φορές. Μια για το κάτω μισό του πίνακα που της ανατίθεται και μία για το πάνω .Στη συνέχεια ταξινομεί τα δύο μισά κατά αύξουσα και φθίνουσα σειρά αντίστοιχα και τα συνενώνει(ταξινομημένα).Έτσι αν σκεφτούμε τις αναδρομικές αυτές κλίσεις σαν ένα δυαδικό δέντρο και εξετάσουμε ένα τυχαίο επίπεδό του, μπορούμε να συμπεράνουμε ότι όλες οι ενέργειες(κόμβοι του επιπέδου) που εκτελούνται στο επίπεδο αυτό, αφορούν διαφορετικό μέρος του πίνακα η κάθε μία. Αυτό σημαίνει ότι είναι ανεξάρτητες μεταξύ τους, δηλαδή μπορούν να ανατεθούν σε ξεχωριστά threads και να παραλληλοποιηθούν .

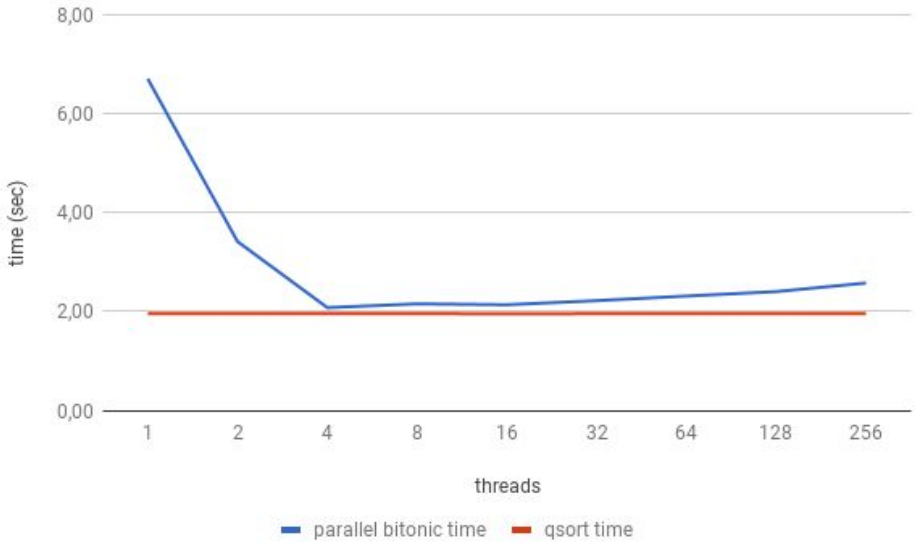
Έτσι, μέσα στην recBitonicSort() επιλέγω να παραλληλοποιήσω τις 2 κλίσεις πού κάνει στον εαυτό της.
Όταν τα νήματα που δίνει ο χρήστης δεν φτάνουν για να ολοκληρωθεί η ταξινόμηση με παραλληλοποίηση σε κάθε επίπεδο, καλείται η quicksort για να συνεχίσει τη διαδικασία σε κάθε νήμα, πράγμα που οδηγεί και σε βελτίωση του χρόνου εκτέλεσης.

Έλεγχος της ορθότητας & δοκιμές

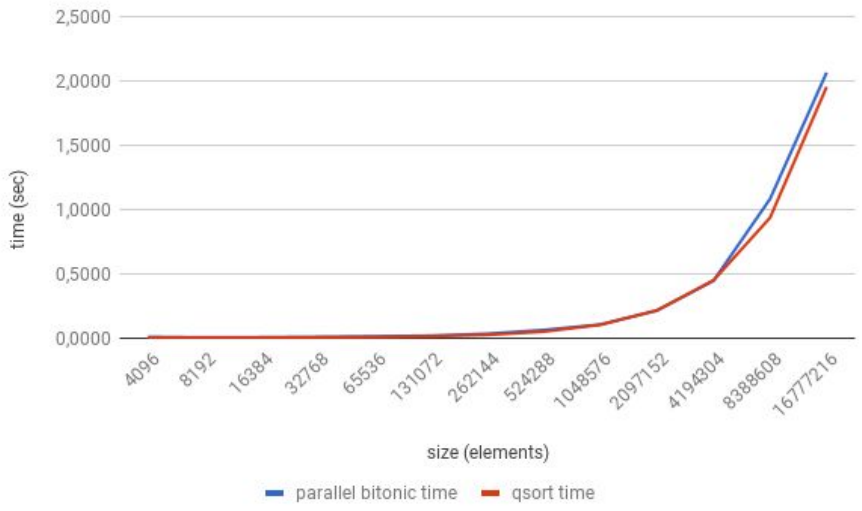
Ο έλεγχος της ορθότητας των αλγορίθμων έγινε με τη χρήση της δοθείσας συνάρτησης test().
Έπειτα από επαναληπτικές εκτελέσεις για διαφορετικά πλήθη νημάτων και στοιχείων(προς ταξινόμηση) προέκυψαν τα γραφήματα που παρουσιάζονται στη συνέχεια και αφορούν τους χρόνους εκτέλεσης (σε δευτερόλεπτα) του παράλληλου bitonic σε κάθε εκδοχή του.
!!! Σημειώνεται ότι οι δοκιμές έγιναν σε τετραπύρινο σύστημα και όχι στο diades. !!!

Υλοποίηση παράλληλου bitonic με pthreads

Για την εκδοχή με pthreads επέλεξα την επαναληπτική υλοποίηση.
Η υλοποίηση αυτής της λογικής βρίσκεται στο αρχείο *pthreads_bitonic.c* .



Γ1) Παρουσίαση χρόνων για παράλληλη bitonic(με pthreads) και (σειριακή)quicksort σε σχέση με τον αριθμό των threads (για 2^{24} στοιχεία).



Γ2) Παρουσίαση χρόνων για παράλληλη bitonic(με pthreads) και (σειριακή)quicksort σε σχέση με τον αριθμό των στοιχείων (για 4 threads).

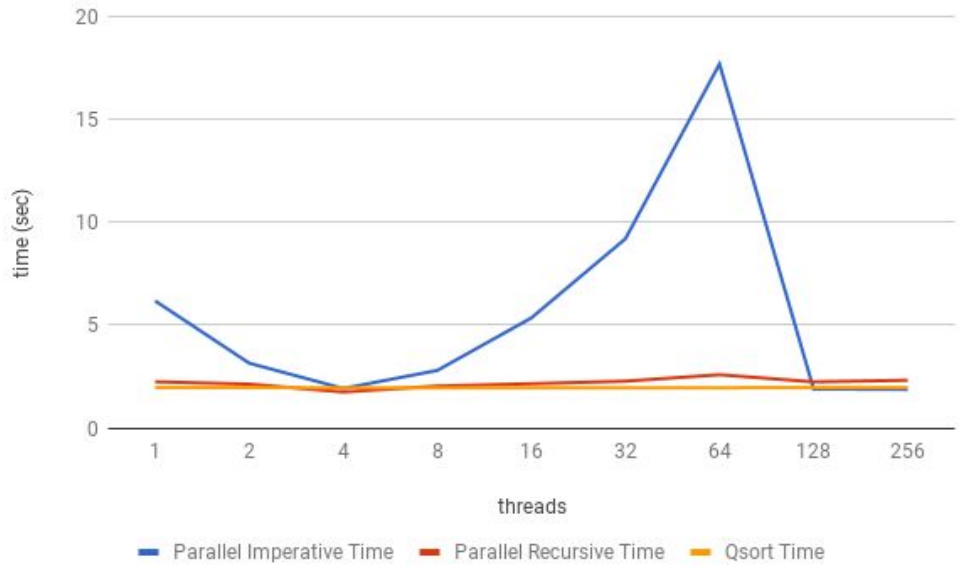
Όπως προκύπτει από το Γ1, ο καλύτερος χρόνος εκτέλεσης της παράλληλης bitonic επιτυγχάνεται για 4 threads και πλησιάζει σημαντικά τον σταθερό χρόνο της quicksort. Έπειτα από 10 εκτελέσεις (για 2^{24} στοιχεία & 4 threads) και αφού αφαιρέθηκαν οι ακραίες τιμές οι χρόνοι στους οποίους αναφέρομαι είναι οι εξής:

- parallel bitonic : *2,0842385 sec*
- qsort : *1,9501514 sec*

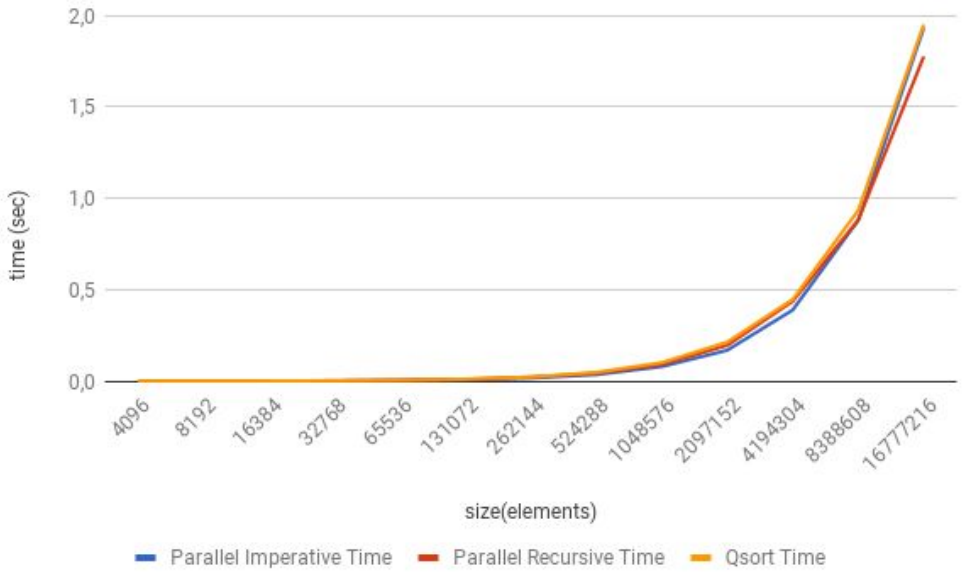
Το Γ2 επιβεβαιώνει ότι για 4 threads ο χόνος εκτέλεσης του παράλληλου bitonic είναι σχεδόν ίδιος με αυτόν της quicksort, ανεξάρτητα του πλήθους των στοιχείων.

Υλοποίηση παράλληλου bitonic με cilkplus

Παραλληλοποιήθηκε με χρήση της cilkplus τόσο η επαναληπτική όσο και η αναδρομική εκδοχή της bitonic. Στην αναδρομική χρησιμοποιήθηκε και η quicksort για βελτίωση του χρόνου.
Η υλοποιήσεις βρίσκονται στο αρχείο *cilk_bitonic.c*



Γ3) Παρουσίαση χρόνων για παράλληλη bitonic(επαναληπτική και αναδρομική σε cilkplus) και (σειριακή)quicksort σε σχέση με τον αριθμό των threads (για 2^{24} στοιχεία).



Γ4) Παρουσίαση χρόνων για παράλληλη bitonic(επαναληπτική και αναδρομική σε cilkplus) και (σειριακή)quicksort σε σχέση με τον αριθμό των στοιχείων (για 4 threads).

!!! Παρατηρήθηκε ότι η cilkplus κατά την απόπειρα δημιουργίας περισσότερων από 64 thread έβγαζε error και έκανε reset στα 4(τα default δηλαδή).Αυτός είναι και ο λόγος που στο Γ3 έχουμε ξαφνική πτώση του χρόνου της parallel imperative(μπλε) μετά από τα 64 thread.

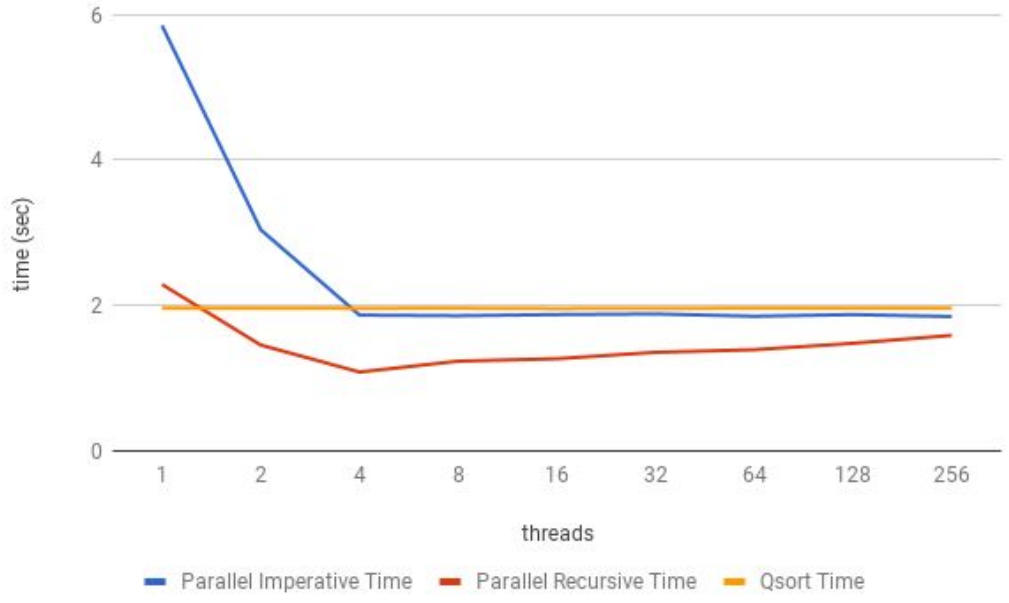
Όπως προκύπτει από το Γ3, ο καλύτερος χρόνος εκτέλεσης της παράλληλης(με cilkplus) bitonic τόσο επαναληπτικής όσο και αναδρομικής(+χρήση qsort) επιτυγχάνεται για 4 threads και ξεπερνά(είναι μικρότερος) οριακά τον σταθερό χρόνο της quicksort. Ωστόσο οι χρόνοι της παράλληλης επαναληπτικής bitonic αυξάνονται δραματικά για thread πλήθους διαφορετικού του 4(default), πράγμα που μάλλον οφείλεται στον τρόπο υλοποίησης της _Cilk_for που χρησιμοποιήθηκε για την παραλληλοποίηση του εσωτερικού βρόγχου.
Έπειτα από 10 εκτελέσεις (για 2^{24} στοιχεία & 4 threads) και αφού αφαιρέθηκαν οι ακραίες τιμές οι χρόνοι στους οποίους αναφέρομαι είναι οι εξής:

- parallel imperative bitonic : 1,88326625 sec
- parallel recursive bitonic : 1,78834075 sec
- qsort : 1,9501514 sec

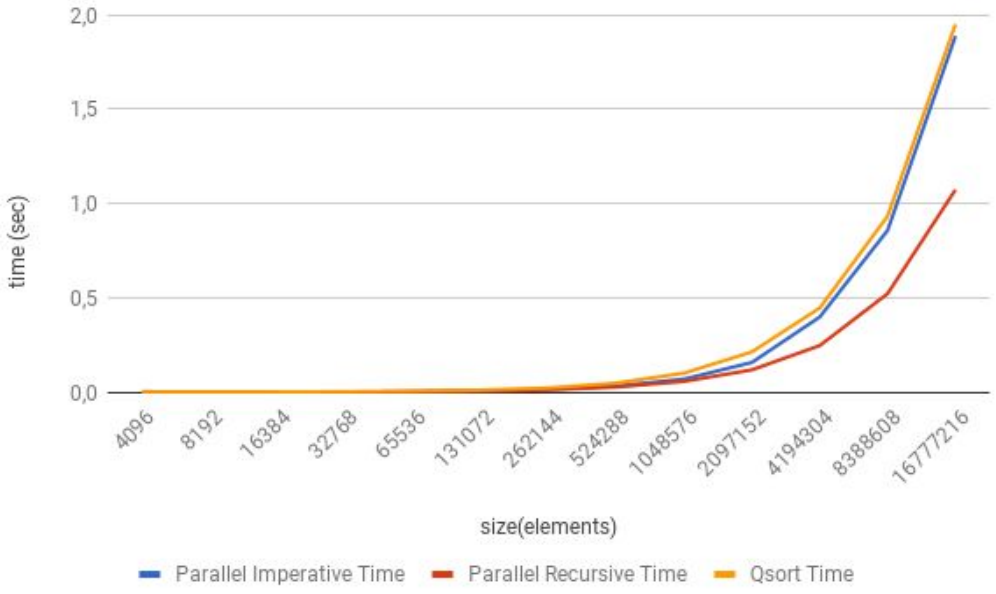
Το Γ4 επιβεβαιώνει ότι για 4 threads ο χρόνος εκτέλεσης του παράλληλων αυτών εκδοχών της bitonic είναι σχεδόν ίδιος με αυτόν της quicksort, ανεξάρτητα του πλήθους των στοιχείων.

Υλοποίηση παράλληλου bitonic με openmp

Παραλληλοποιήθηκε με χρήση της cilkplus τόσο η επαναληπτική όσο και η αναδρομική εκδοχή της bitonic. Στην αναδρομική χρησιμοποιήθηκε και η quicksort για βελτίωση του χρόνου.
Η υλοποιήσεις βρίσκονται στο αρχείο *openmp_bitonic.c*



Γ5) Παρουσίαση χρόνων για παράλληλη bitonic(επαναληπτική και αναδρομική σε openmp) και (σειριακή)quicksort σε σχέση με τον αριθμό των threads (για 2^{24} στοιχεία).



Γ6) Παρουσίαση χρόνων για παράλληλη bitonic(επαναληπτική και αναδρομική σε openmp) και (σειριακή)quicksort σε σχέση με τον αριθμό των στοιχείων (για 4 threads).

Όπως προκύπτει από το Γ5, ο καλύτερος χρόνος εκτέλεσης της παράλληλης(με opnmp) bitonic με αναδρομή(+quicksort) επιτυγχάνεται για 4 threads και ξεπερνά κατα πολύ τον σταθερό χρόνο της απλής quicksort.
Για την επαναληπτική υλοποίηση δεν επιτυγχάνεται απαραίτητα ο καλύτερος χρόνος στα 4 threads, όμως φαίνεται ότι από αυτό το σημείο και έπειτα σταθεροποιείται και είναι βελτιωμένος(σε μικρό βαθμό) σε σχέση με αυτόν της quicksort.
Έπειτα από 10 εκτελέσεις (για 2^{24} στοιχεία & 4 threads) και αφού αφαιρέθηκαν οι ακραίες τιμές οι χρόνοι στους οποίους αναφέρομαι είναι οι εξής:

- parallel imperative bitonic : 1,860579875 sec
- parallel recursive bitonic : 1,074406875 sec
- qsort : 1,9501514 sec

Το Γ6 επιβεβαιώνει ότι για 4 threads ο χρόνος εκτέλεσης της επαναληπτικής παράλληλης bitonic είναι σχεδόν ίδιος με αυτόν της quicksort, ανεξάρτητα του πλήθους των στοιχείων, ενώ η αναδρομική παράλληλη εκδοχή(+χρήση qsort) γίνεται αρκετά πιο γρήγορη από την απλή quicksort όσο αυξάνονται τα στοιχεία.