Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης

Πολυτεχνική Σχολή

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Μάθημα: Παράλληλα και διανεμημένα συστήματα

Ακαδημαικό Έτος: 2017-2018

1η Εργασία

Φοιτητής:**Παπαδόπουλος Σταύρος**

ΑΕΜ:**8697**

Περιεχόμενα

***Οι ενότητες(4.3,6.3) δεν έχουν συμπληρωθεί λόγω κακού προγραμματισμού του χρόνου ,για την ολοκλήρωση της εργασίας,από τον φοιτητή.**

1.Παραλληλισμός-Παραλληλος Προγραμματισμός.

Σκοπός του παράλληλου προγραμματισμού είναι να επιτύχει την αύξηση των υπολογιστικών επιδόσεων και μείωση του απαιτούμενου χρόνου εκτέλεσης μιας εφαρμογής.Η λειτουργία του βασίζεται στην «εκμετάλλευση» της ύπαρξης πολλαπλών επεξεργαστικών μονάδων σε έναν πολυεπεξεργαστή η πολυυπολογιστή. Έτσι ο αλγόριθμος παραλληλοποιείται με την διάσπασή του σε πολλαπλά τμήματα τα οποία ανατίθενται σε ξεχωριστά νήματα ή διεργασίες και έτσι εκτελούνται παράλληλα σε διαφορετικές επεξεργαστικές μονάδες.

Τα μοντέλα παράλληλου προγραμματισμού για πολυεπεξεργαστές και πολυυπολογιστές είναι το μοντέλο κοινού χώρου διευθύνσεων (π.χ.**pthreads**, **OpenMP**) και το μοντέλο μεταβίβασης μηνυμάτων (π.χ. PVM, MPI), αντιστοίχως.

Η παρούσα εργασία έχει ως απαιτούμενο την χρήση των μοντέλων των **pthreads,openMP και CilkPlus**.Σκοπός της εργασίας αυτής είναι να παραλληλοποιήσουμε τον σειριακό αλγόριθμο της ταξινόμησης **bitonic sort** που μας δόθηκε απο τους καθηγητές μας με χρήση των μεθόδων:

- **CilkPlus** (αναδρομικά και επαναληπτικά)
- **openMP**(αναδρομικά και επαναληπτικά)
- **pthreads** (αναδρομικά η επαναληπτικά)

2.Bitonic Sort-Τρόπος λειτουργίας.

Η Bitonic Sort ταξινομεί λίστες που έχουν διτονικη μορφή.Αυτό σημαίνει ότι πρώτο μισό της λίστας ειναι ταξινομημένο σε αύξουσα σειρα και το δεύτερο μισό σε φθίνουσα.Η υλοποίηση της αύξουσας ταξινόμησης επιτυγχάνεται συγκρίνοντας κάθε στοιχείο της άυξουσας λίστας με το αντίστοιχο σε θέση αυτού της φθίνουσας λίστας.Στην συνέχεια αν το πρώτο είναι μεγαλύτερο γίνεται αντιμετάθεση.Έπειτα κάθε μισο διαιρείται στην μέση και γίνονται αντίστοιχες συγκρίσεις μεταξύ των στοιχείων των δύο μισών που βρίσκονται στην ίδια θέση.Η διαδικασία αυτή επαναλαμβάνεται έως ότου η τελευταία σύγκριση αφορά ανα δύο διαδοχικά στοιχεία.Σε περιπτώση πους μας δίνεται μία τυχαία λίστα(χωρις συγκεκριμενη διαταξη αριθμων),είναι απαραίτητο να μετατρέψουμε την λίστα έτσι ώστε να πάρει διτονική μορφή με σκοπό έπειτα της χρήσης της bitonic sort,που όπως ανέφερα παραπάνω ταξινομεί διτονικές λίστες.Για την μετατροπή αυτήν διαιρούμε ακέραια και διαδοχικά την λίστα στην μέση της έως ότου αυτό να μην ειναι εφικτο.Σε καθε στάδιο ταξινομούμε το πρώτο και το δεύτερο μισό σε αύξουσα και φθινουσα σειρά

αντίστοιχα.Ξεκινώντας από την τελευταία διαίρεση πραγματοποιεί συγκρίσεις φέρνει το αντιστοιχο μερος σε διτονική μορφή και το ταξινομεί ανάλογα κρινοντας με βαση το «μισο» που ανήκει στο αμέσως επόμενο(χαμηλότερο) επίπεδο(μέρος).Η διαδικασία αυτή συνέχιζεται απο τα «υψηλότερα» προς τα «χαμηλότερα» επίπεδα διαίρεσης και έχεις ως αποτέλεσμα τηνς μετατροπή της λίστας σε διτονική μορφη έτσι ώστε να είναι εφικτή η ταξινόμηση της με την χρηση της **bitonic sort**.

3.Αναδρομική Διτονική Ταξινόμηση-Recursive Bitonic Sort.

Για την υλοποιηση της αναδρομικής bitonic sort δημιουργήθηκαν οι ακόλουθες συναρτήσεις:

- ***void recursiveBitonicSort(int lo, int cnt, int dir)***

  - Σκοπός αυτής της συνάρτησης ειναι να μετρατρέψει την μορφή μιας ακολουθίας σε διτονική.
  - Το ορισμα lo αντισχοιεί στο πρώτο στοιχείο της ακολουθίας.Το cnt δείχνει το μέγεθος της λίστας και το dir την κατεύθυνση της ταξινόμησης.
  - Μέσω κλήσεων στον εαυτό της(αναδρομή) επιτυγχάνεται η διάσπαση που ειναι απαραίτητη για να αρχίσει η μετατροπή από το ανώτερο επίπεδο διαίρεσης(οπως περιγραφτηκε παραπάνω).

- ***void bitonicMerge(int lo, int cnt, int dir)***

  - Σκοπός αυτής της συνάρτησης είναι να ταξινομήσει σε αύξουση η φθίνουσα σειρά μια διατονική λίστα.
  - Το ορισμα lo αντισχοιεί στο πρώτο στοιχείο της ακολουθίας.Το cnt δείχνει το μέγεθος της λίστας και το dir την κατεύθυνση της ταξινόμησης.
  - Έτσι κάνεις τις πρωτες N/2 συγκρισεις(οπου να το μέγεθος του πινακα) συγκρίνοντας καθε στοιχείο του πρώτου μισού της λίστας με το αντίστοιχο σε θέση του δεύτερου μισού(οπως περιγράφτηκε παραπάνω).Επειτα μέσω μια αναδρομικής διαδικασίας κανει την διάσπαση που χρειάζεται έτσι ώστε να συγκρίνει τα πιο κοντινά στοιχεία του πίνακα σύμφωνα με την διατονική διαδικασία που αναφέρθηκε σε προηγούμενη παράγραφο.

4.Παραλληλοποίηση

Για την παραλληλοποίηση του σειριακού κώδικα της **bitonic sort** απαιτείται η υλοποίηση της να γίνει μέσω **pthreads,openMP και CilkPlus**.

4.1.Παραλληλοποίση με χρήση των pthreads.

- Για την υλοποίηση με pthreads δημιουργήθηκαν επιπλέον δύο συναρτήσεις οι οποίες αποτελούν την παράλληλη υλοποίηση των αντιστοιχων δύο που αναφέρθηκαν παραπάνω(void recursiveBitonicSort(int lo, int cnt, int dir) , void bitonicMerge(int lo, int cnt, int dir))

- ***void *PrecursiveBitonicSort(void *td)***

  - Η λογική της λειτουργίας της ειναι ιδια με την αντιστοιχη της με την διαφορά οτι στην παράλληλη εκδοση ανοίγονται νήματα οπου αναλαμβάνουν την εκτέλεση 2 αναδρομικών κλήσεων ταυτόχρονα.Αρχικά ελέγχεται αν το μήκος του πινακα ειναι μεγαλύτερο του 1,οπου αν δεν ειναι επιστρέφει και στην συνέχεια αν ο αριθμός των νημάτων ειναι μικρότερος από τον επιτρεπτό δημιοργούνται δύο νήματα(για καθε αναδρομική κλήση) και στην πρώτη αναδρομική κλήση ανατίθεται το πρώτο μισό του πίνακα κατά αύξουσα κατεύθυνση και στην δεύτερη το άλλο μισο αντίστοιχα.Ετσι κάθε κλήση θα δημιουργησει εκ νέου από δύο νηματα κατά τον ίδιο τρόπο.Όταν εξαντληθεί ο αριθμός των νημάτων καλείται η void recursiveBitonicSort(int lo,int cnt,int dir) για να συνεχιστεί η διαδικασία σειριακά.Οταν επιστρέψουν οι παράλληλες διαδικασίες σε αυτήν που τις κάλεσε καλείται η void *PbitonicMerge(void *td) οπου αυτή ειναι υπευθηνη για την ταξινόμηση δημουργώντας ακολουθία τέτοιας κατεύθυνσης που έχει λάβει ως όρισμα.Στην ουσία καθε ζεύγος νημάτων δημιουργεί μια διτονική ακολουθία που στην συνέχεια ταξινομείται με κατεύθυνση που ορίζεται από το νήμα προηγούμενου επιπέδου και με το πέρας των αναδρομών ο πίνακας έχει ταξινομηθεί.

- ***void *PbitonicMerge(void *td)***

  - Τρέχει και αυτή παράλληλα με δημιουργία δύο νημάτων για κάθε αναδρομή.Μετά το τέλος των αναδρομών ο πίνακας που εχει δωθεί ταξινομείται πλήρως.Τα νήματα δημιουργούνται με την pthread_create() και μέσω της pthread_join() ο αλγόριθμος περιμένει μέχρι τα νήματα να τελειώσουν και έπειτα να συνεχίσει.Για την αποφυγή της σύγχησης κατά την πρόσβαση και επεγεργασία των νημάτων στην ιδια περιοχή μνήμης κλειδώνονται οι μεταβλητές μεσω **mutex**.

4.2 Παραλληλοποίηση με χρήση openMP.

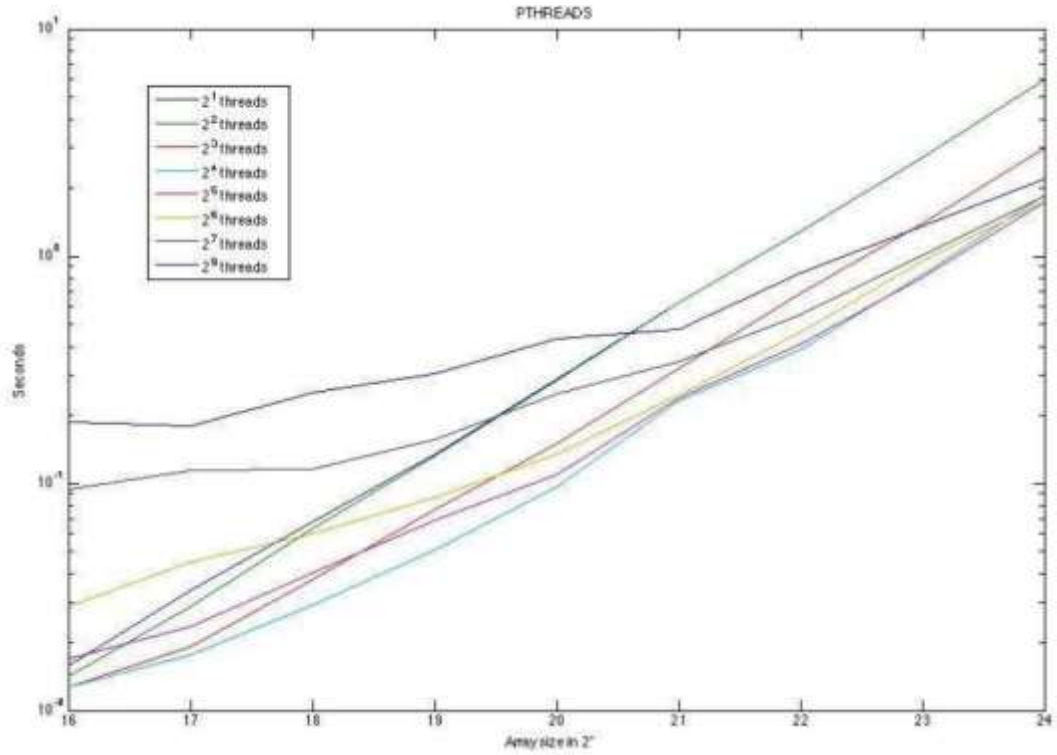
- Η λογική της μεθόδου αυτής ειναι η ίδια με την διαφορά να υπάρχει στις εντολές.Αρχικά,μια εφαρμογη σε openMP ξεκινά με ένα μονο νήμα, το οποιο ονομάζεται master thread.Όταν το πρόγραμμα εισέρχεται σε μια περιοχή που έχει ορίσει ο προγραμματιστής να εκτελεστεί παράλληλα(παράλληλη περιοχή) δημιουργούνται αρκετά νήματα(fork-join model) και το μέρος του κώδικα που βρίσκεται μεσα στην παράλληλη περιοχή εκτελείται παράλληλα.Οταν ολοκληρωθεί ο υπολογισμός της παράλληλης περιοχής όλα τα νήματα τερματίζουν και συνεχίζει μονο το master thread.Για να δηλώσουμε την περιοχή του κώδικα που θα εκτελεστεί παράλληλα χρησιμοποιούμε την εντολή **#pragma omp parallel**.Στην συνέχεια για να ορίσουμε τα τμήματα του κώδικα που θα ανατεθούν στα νήματα χρησιμοποιούμε την εντολή **#pragma omp sections** . Στην περίπτωση μας σε κάθε section αναθέτουμε από μια αναδρομή.Μεταβλητές κλειδώνονται αντίστοιχα με την εντολή **omp_set_lock(&key).**

5.Έλεγχος ορθότητας.

Για τον έλεγχο ορθότητας κανουμε χρηση της συνάρτησης test().Η συνάρτηση αυτή καλείται μετά την ταξινόμηση και ελέγχει αν τα στοιχεία είναι διατεταγμένα σε αύξουσα σειρά.Ο έλεγχος πραγματοποιείται κανοντας σειριακή σύγκριση στα διαδοχικά στοιχεία ελέγχοντας αμα ισχύει η αυξουσα διαδοχή τους.Αν ειναι ταξινομημένα ορθα η συνάρτηση προσθέτει στην μεταβλητή pass το λογικά 1,διαφορετικά προσθέται το λογικό μέσω της **pass&=(a[i-1]<=a[i])**.Αν έστω και μία συγκριση δεν ειναι ορθή το pass θα πάρει την τιμή 0.Στην περιπτωση που το pass ισούται με το λογικό 1 εκτυπώνεται μήνυμα επιτυχίας.Αντιστοιχα στην περίπτωση που ισούται με το λογικό 0 εκτυπώνεται αντιστοιχα χαρακτηριστικό μηνυμα αποτυχίας.

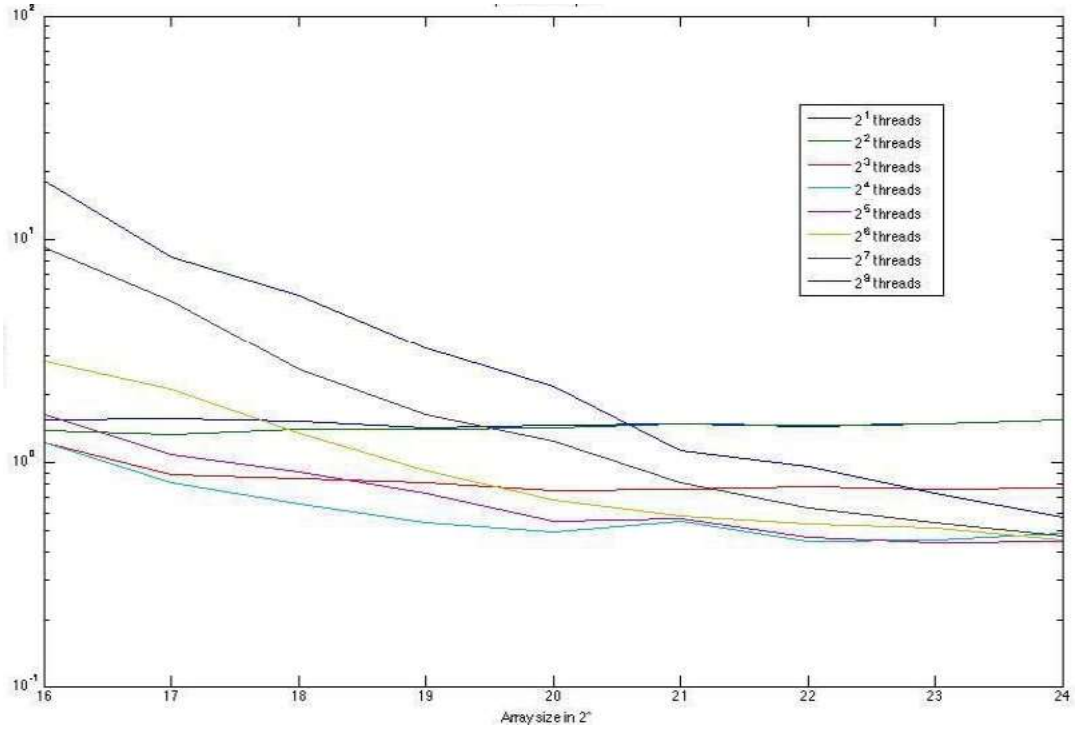
6.Ανάλυση αποτελεσμάτων-συγκριση

➢ 6.1.Χρήση pthreads.



Εικ1.1*:Στην εικόνα αυτή παρουσιάζεται ο χρόνος εκτέλεσης της παράλληλης ταξινόμησης σε σχέση με το μέγεθος του πίνακα και το πλήθος των νημάτων, με χρηση pthreads .*

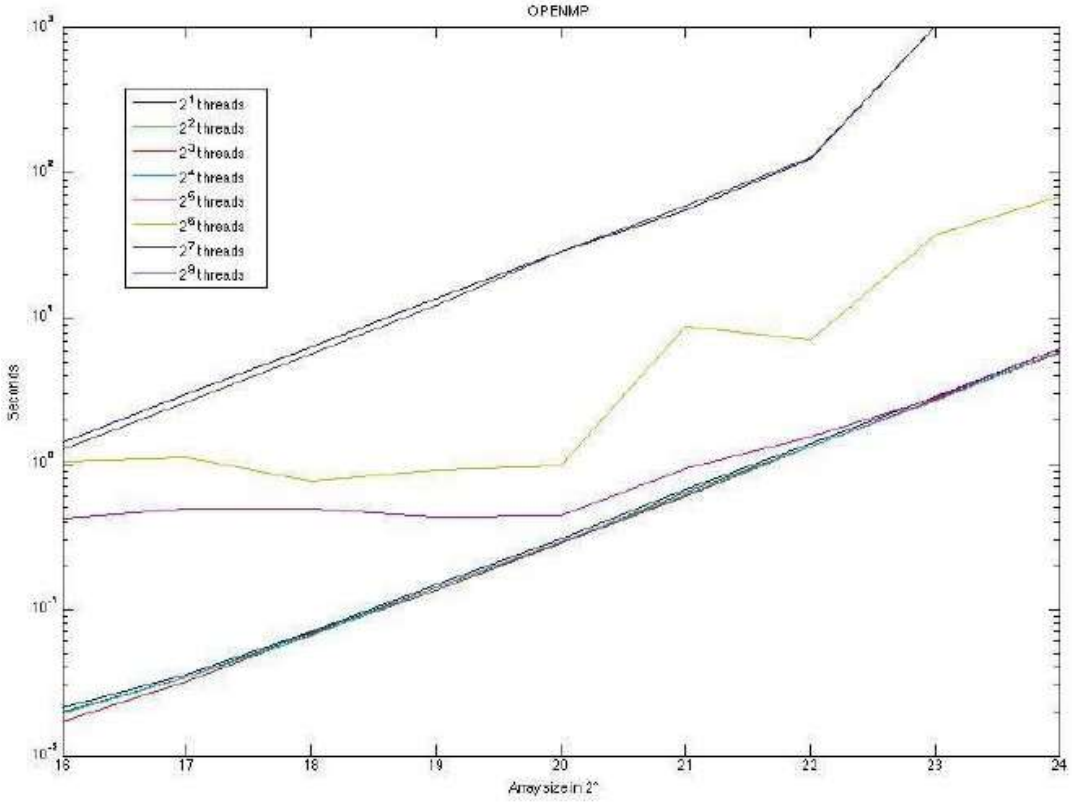
Πόρισμα:Ο χρόνος εκτέλεσης του αλγορίθμου ειναι αντιστρόφως ανάλογος με τον αριθμό των νημάτων.



Εικ1.2:Χρόνος εκτέλεσης ταξινόμησης(εξαρτόμενος απο το πλήθος νημάτων) σε σχέση με το σειριακό αλγόριθμο της bitonic sort μέσω του λόγου των δύο χρόνων εκτέλεσης(χρονος pthread/χρονος serial bitonic sort) για διάφορα μεγέθη πινάκων.

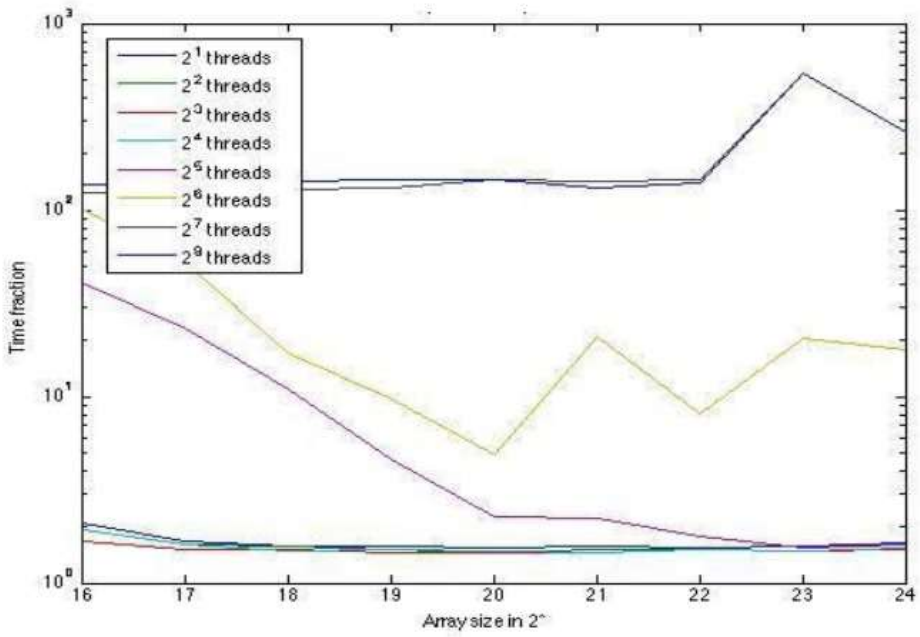
Πόρισμα:Για μεγάλο αριθμό νημάτων έχουμε καλύτερη απόδοση και ειδικότερα για μεγάλα μεγέθη πινάκων.

➢ 6.2 χρήση openMP.



Εικ2.1:*Χρόνος εκτέλεσης παράλληλης ταξινόμησης bitonicsort σε σχέση με το μεγεθος του πίνακα και το πλήθος των νημάτων,με μεθόδους openMP.*

Πόρισμα:Η ταχύτητα της εκτέλεσης ειναι ανάλογη του πλήθους νημάτων.



Εικ2.2*:Λόγος των χρόνων εκτελεσης αλγορίθμοτ με παραλληλη υλοποιηση(openMP) και σειριακής υλοποίησης της bitonic sort σε σχέση με τα μεγέθη των πινάκων.*

Πόρισμα:Όσο αυξάνονται τα μήκη των πινάκων και ο αριθμος των νημάτων οι δύο χρόνοι τείνουν να συμπέσουν.

7.Συμπέρασμα.

Συμπερασματικά μπορούμε να πούμε πως η απόδοση σε γεννικές γραμμές αυξάνεται με την χρήση παράλληλων τεχνικών και ειδικότερα όσο αυξάνεται η πολυπλοκότητα και ο φόρτος εργασίας.Παρ'όλα αυτλα το κάθε πρόβλημα διαφέρει απο τα υπόλοιπα και ο προγραμματιστής οφείλει να κανει σαφή ανάλυση αυτού έτσι ώστε να καταληξει στην εφαρμογή της αποδοτικότερης λύσης.**Όσο αφορά την παραλληλοποίηση της bitonic sort βλέπουμε πως η τεχνική των pthreads είναι αποδοτικότερη απο την αντίστοιχη των openMP.**