

**2η Άσκηση**  
**Παράλληλα & Διανεμημένα Συστήματα**  
**Υπολογιστών**

ΟΝΟΜΑ : ΠΕΤΡΟΣ  
ΕΠΙΘΕΤΟ: ΕΥΑΓΓΕΛΑΚΟΣ  
ΑΕΜ :7680  
e-mail: petrosei@auth.gr

ΟΝΟΜΑ : ΚΩΝΣΤΑΝΤΙΝΟΣ  
ΕΠΙΘΕΤΟ: ΧΑΜΖΑΣ  
ΑΕΜ :7798  
e-mail: chamzask@auth.gr

# Παράλληλα και Διανεμημένα Συστήματα 2η Εργασία

## ΠΕΡΙΕΧΟΜΕΝΑ

1. Ανάλυση της παράλληλης υλοποίησης της Bitonic Sort με MPI.....σελ.3
2. Επιπλέον σχόλια για την τρόπο που τρέξαμε τους αλγορίθμους....σελ.11
3. Ανάλυση του ελέγχου ορθότητας της μεθόδου.....σελ.12
4. Συγκριτικά διαγράμματα σειριακής και παράλληλης υλοποίησης..σελ.13

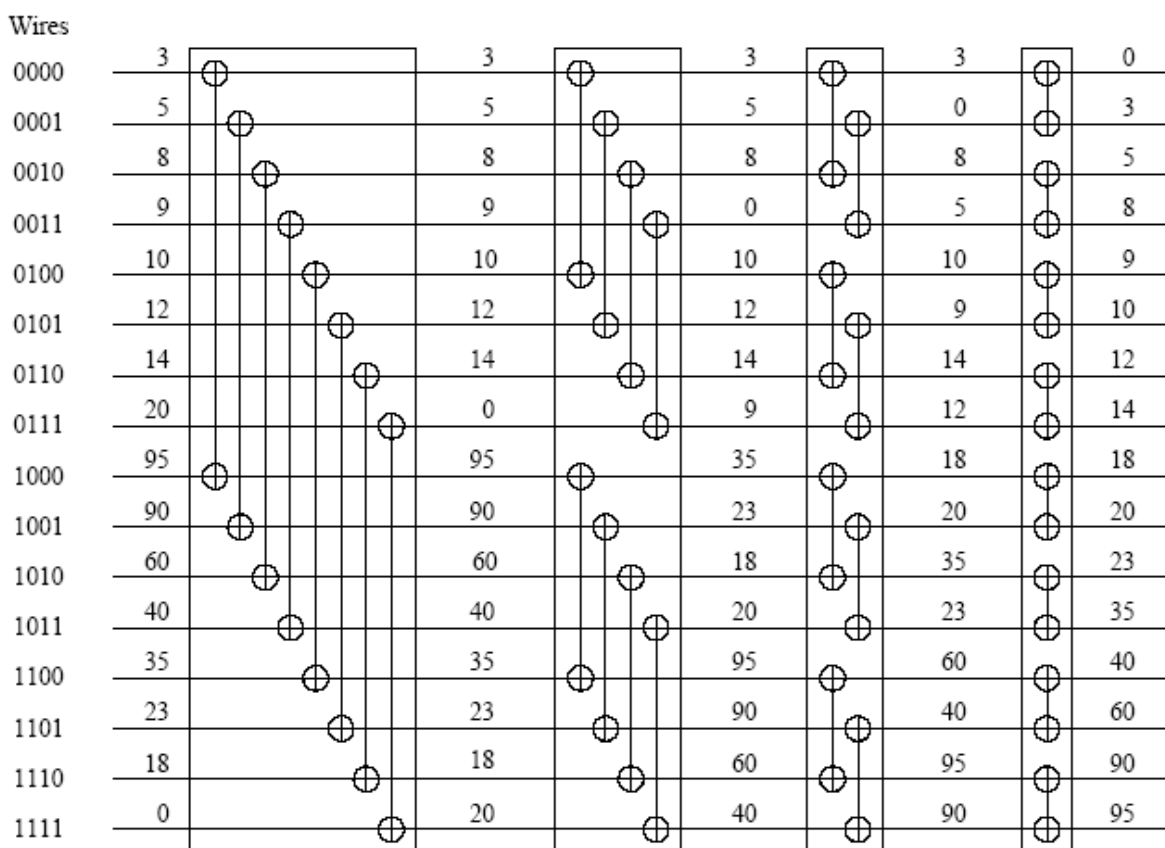
# Παράλληλα και Διανεμημένα Συστήματα 2η Εργασία

## 1. Ανάλυση της παράλληλης υλοποίησης της Bitonic Sort με MPI

Στην συγκεκριμένη υλοποίηση χρησιμοποιήσαμε την επαναληπτική έκδοση του αλγορίθμου της Bitonic Sort για να κατατάξουμε μία ακολουθία αριθμών σε αύξουσα σειρά. Γενικά η Bitonic Sort χρειάζεται μια bitonic ακολουθία που αποτελείται από μία ακολουθία σε αύξουσα σειρά και μια σε φθίνουσα.

Συγκρίνοντας τα στοιχεία μεταξύ τους δημιουργούμε με  $N$  συγκρίσεις 2 λίστες εκ των οποίων τα στοιχεία της μιας είναι μικρότερα από της άλλης και είναι η καθεμία τους μία bitonic\_sequence. Εφαρμόζοντας αναδρομικά την διαδικασία καταλήγουμε σε μία ταξινομημένη ακολουθία.

Σχεδιάγραμμα του bitonic merging



## Παράλληλα και Διανεμημένα Συστήματα 2η Εργασία

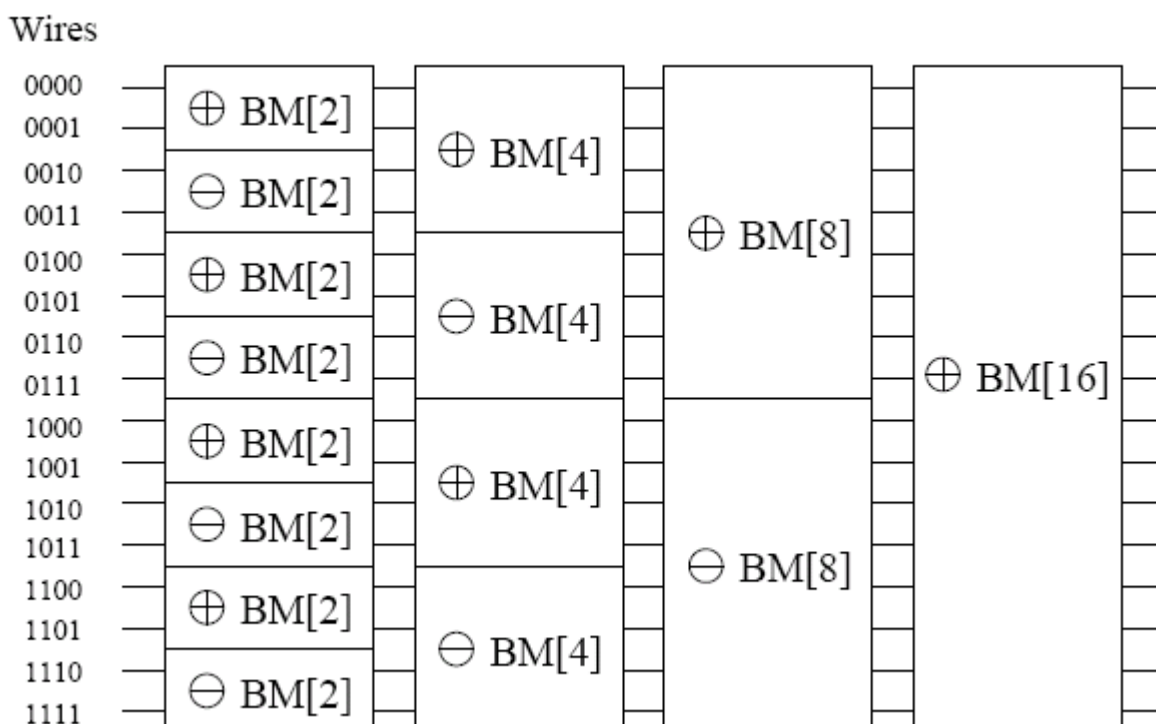
Μπορούμε τώρα να θεωρήσουμε τη δημιουργία των δύο παραπάνω μονοτονικών ακολουθιών ως υποπροβλήματα που λύνονται και αυτά με χρήση της ίδιας μεθόδου. Το πρόβλημα μπορεί να διασπαστεί σε διαδοχικά υποπροβλήματα μέχρι να φτάσουμε στην πιο μικρή δυνατή bitonic ακολουθία που είναι η ακολουθία δυο στοιχείων. Έτσι θα έχουμε δυο ακολουθίες που αποτελείται η κάθε μια από ένα στοιχείο και μπορούν να θεωρηθούν αύξουσα και φθίνουσα αντίστοιχα. Έτσι λύνοντας διαδοχικά όλα αυτά τα υποπροβλήματα καταλήγουμε τελικά σε μία ταξινομημένη ακολουθία.

Το σχήμα που ακολουθεί περιγράφει αυτή την διαδικασία

$\oplus$ BM[N] Σημαίνει αύξουσα ταξινόμηση N στοιχείων με Bitonic Merging

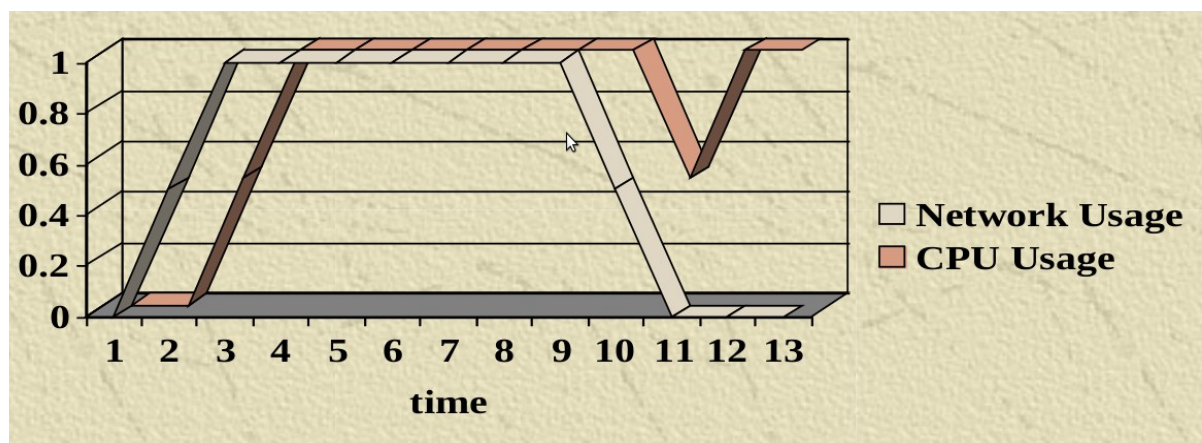
$\ominus$ BM[N] Σημαίνει φθίνουσα ταξινόμηση N στοιχείων με Bitonic Merging

### Δημιουργία Bitonic sequence



## Παράλληλα και Διανεμημένα Συστήματα 2η Εργασία

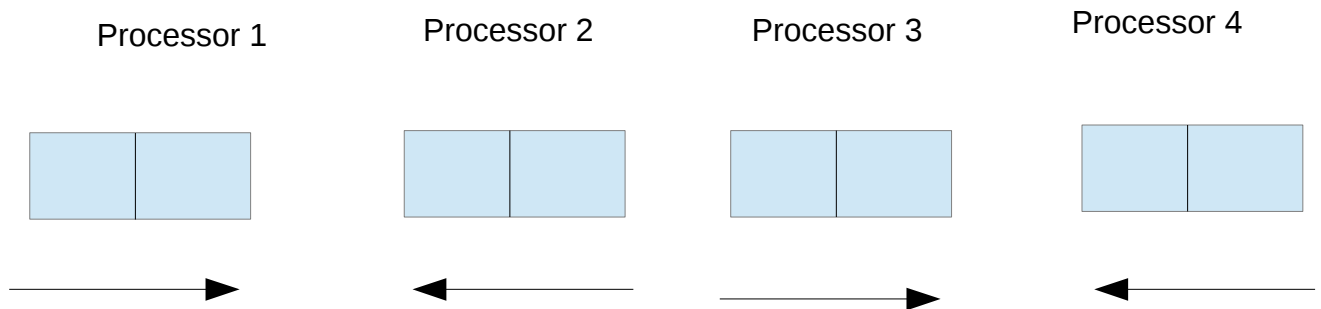
Χρησιμοποιώντας τώρα την παραπάνω λογική μπορούμε να χωρίσουμε σε περισσότερους επεξεργαστές ίσα κομμάτια της αρχικής αταξινόμητης ακολουθίας. Αρχικά μπορούμε να ταξινομήσουμε την κάθε υπακολουθία με χρήση μιας γρήγορης σειριακής μεθόδου ταξινόμησης, όπως είναι η QuickSort. Η ταξινόμηση των υπακολουθιών εναλλάσσεται από αύξουσα σε φθίνουσα σειρά και το αντίστροφο μεταξύ δυο διαδοχικών επεξεργαστών. Στη συνέχεια θεωρούμε ως μικρότερη bitonic ακολουθία το σύνολο των στοιχείων δύο γειτονικών επεξεργαστών, όπου ο κάθε ένας έχει μία μονοτονική ακολουθία αντίθετης φοράς μεταξύ τους. Έτσι παίρνοντας μυνήματα με χρήση της MPI ανάμεσα στους διαφορετικούς επεξεργαστές εφαρμόζουμε την παραπάνω λογική κάνοντας Bitonic Merging σε κάθε bitonic ακολουθία. Έτσι σε κάθε εξωτερικό βήμα δημιουργούμε διπλάσιες σε μέγεθος bitonic ακολουθίες και μισές σε πλύθος. Οπότε στο τέλος καταλήγουμε σε μία ενιαία ταξινομημένη ακολουθία. Τώρα σε κάθε εσωτερικό βήμα κάνουμε όπως είπαμε Bitonic Merging σε κάθε bitonic ακολουθία. Το πρώτο βήμα στο Bitonic Merging αν έχουμε μία bitonic ακολουθία είναι να συγκρίνουμε ένα προς ένα τα στοιχεία της αύξουσας με τα στοιχεία της φθίνουσας ακολουθίας και να κάνουμε ανταλλαγές όπου χρειάζεται έτσι ώστε στο τέλος τα στοιχεία της μίας ακολουθίας να είναι μικρότερα από της άλλης. Στη συνέχεια σπάμε κάθε ακολουθία στη μέση και εκτελούμε την ίδια διαδικασία σε κάθε υπακολουθία. Αυτό συνεχίζεται μέχρι να φτάσουμε σε δύο στοιχεία. Εδώ επειδή σε κάθε επεξεργαστή θα υπάρχει ένα μόνο κομμάτι της bitonic ακολουθίας (πχ. σε δύο επεξεργαστές κάθε ένας έχει μισή από την bitonic ακολουθία) θα παίρνουμε τα στοιχεία που θέλουμε να συγκρίνουμε σε κάθε επεξεργαστή και όταν ολοκληρωθούν οι συγκρίσεις θα τα επιστρέφουμε. Οπότε κάθε επεξεργαστής θα επικοινωνήσει διδοχικά με όποιον επεξεργαστή χρειάζεται σε κάθε βήμα του bitonic merging μέχρις ότου να έχει μια bitonic sequence της οποίας τα στοιχεία δε θα υπάρχουν ή θα είναι ίσα με τα στοιχεία των υπόλοιπων επεξεργαστών. Στη συνέχεια θα εκτελεί κάθε επεξεργαστής τα υπόλοιπα βήματα του bitonic merging χωρίς να χρειάζεται να επικοινωνήσει με κανέναν. Με αυτόν τον τρόπο καταφέρνουμε να εκτελούμε διαρκώς παράλληλα το bitonic merging σε όλους τους επεξεργαστές. Επίσης τις αποστολές των δεδομένων τις κάνουμε με Isend ώστε να μπορούμε να εκμεταλευτούμε τον χρόνο που δαπανάτε κατά την αποστολή σε εκμεταλεύσιμο υπολογιστικό χρόνο (Σχ 1.1). Από την άλλη τις λήψεις των δεδομένων τις κάνουμε με Recv έτσι ώστε να πετυχαίνουμε έναν καλύτερο συγχρονισμό μεταξύ των επεξεργαστών ώστε να εκτελούνται σωστά τα βήματα του bitonic merging.



# Παράλληλα και Διανεμημένα Συστήματα 2η Εργασία

## 1.1

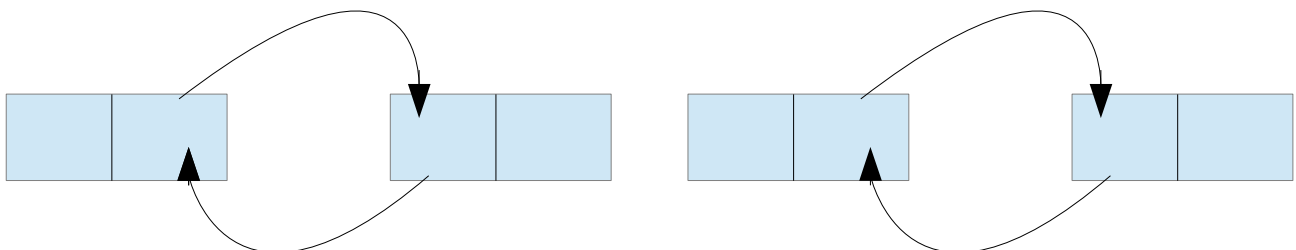
Τώρα θα αναλύσουμε τα βήματα που ακολουθούμε για την υλοποίηση της παραπάνω διαδικασίας. Για ευκολότερη αλλά και για πλήρη κατανόηση της μεθόδου επιλέγουμε το παράδειγμα των τεσσάρων επεξεργαστών.



Σχήμα 1.2

### Βήμα 1ο

Αρχικά έχουμε δύο bitonic ακολουθίες που η κάθε μια χωρίζεται σε δυο επεξεργαστές. Επειδή οι συγκρίσεις των στοιχείων γίνονται ένα προς ένα στοιχείο μεταξύ των δύο μισών μιας bitonic ακολουθίας μπορούμε να ανταλλάξουμε τα μισά στοιχεία μεταξύ των επεξεργαστών Processor1- Processor2 και Processor3- Processor4 , όπως φαίνεται στο παρακάτω σχήμα (Σχ. 1.3).

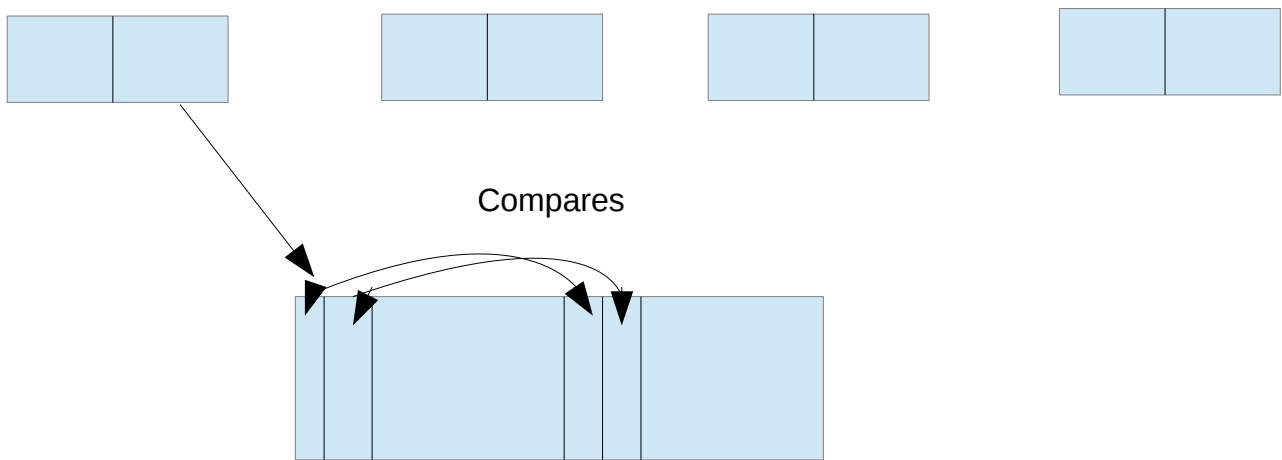


Σχήμα 1.3

## Παράλληλα και Διανεμημένα Συστήματα 2η Εργασία

### Βήμα 2ο

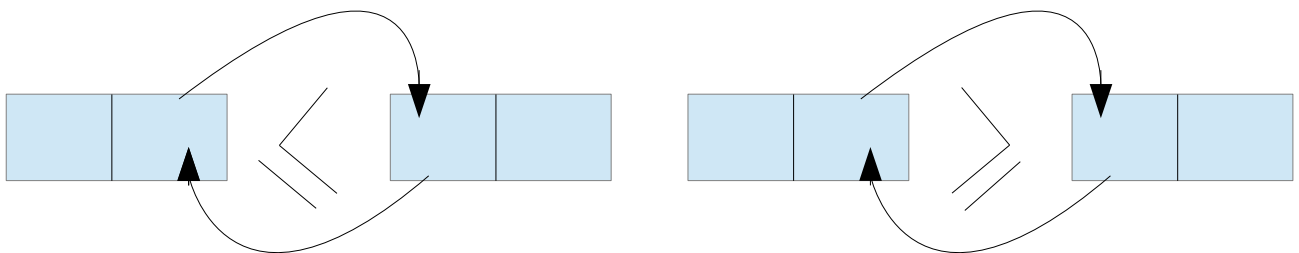
Στη συνέχεια εκτελούμε τις συγκρίσεις (Σχ.1.4)



Σχήμα 1.4

### Βήμα 3ο

Έπειτα ξαναεπιστρέφουμε τα στοιχεία στους επεξεργαστές και ξέρουμε ότι τα στοιχεία του ενός είναι μικρότερα από τα στοιχεία του άλλου(Σχ.1.4)

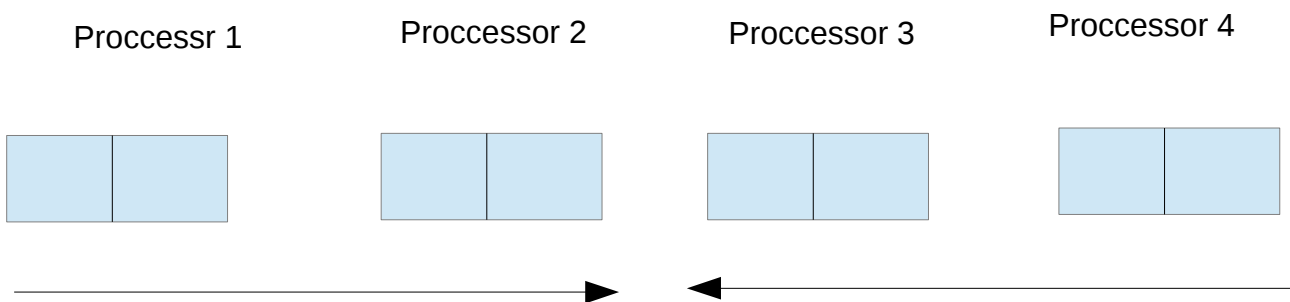


Σχήμα 1.5

## Παράλληλα και Διανεμημένα Συστήματα 2η Εργασία

### Βήμα 4ο

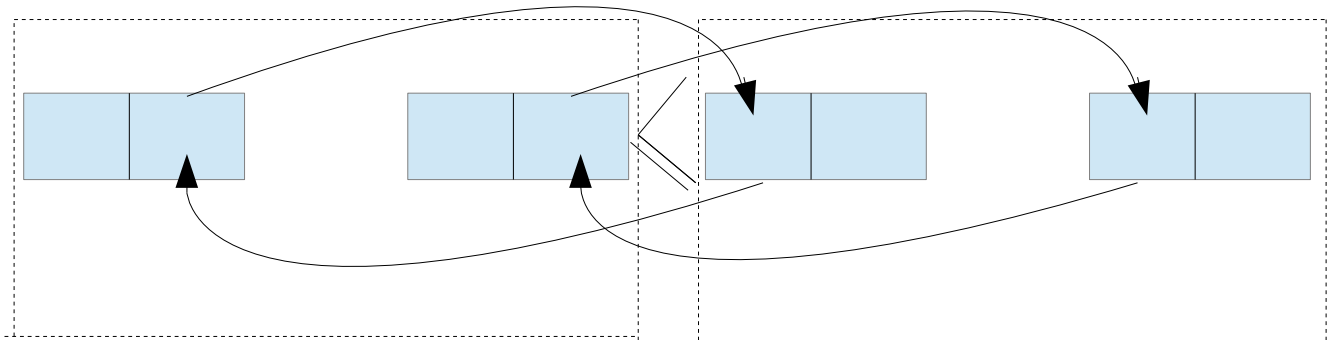
Στη συνέχεια εκτελούμε το Bitonic Merging σε κάθε επεξεργαστή ανεξάρτητα από τους άλλους. Έτσι, ενώ αρχικά είχαμε δύο Bitonic ακολουθίες, καταλήγουμε σε μία εννιαία Bitonic ακολουθία (Σχ.1.6).



Σχήμα 1.6

### Βήμα 5ο

Τώρα έχουμε μια bitonic ακολουθία η οποία χωρίζεται σε 4 επεξεργαστές. Για να εκτελέσουμε το bitonic merging σε αυτή την ακολουθία έχουμε πάλι σαν πρώτο βήμα τη σύγκριση των στοιχείων μεταξύ των μονοτονικών υπακολουθιών οι οποίες χωρίζονται σε δύο επεξεργαστές η κάθε μια. Όμως επειδή όπως είπαμε οι συγκρίσεις γίνονται ένα προς ένα στοιχείο κάθε επεξεργαστής θα επικοινωνήσει μόνο με έναν επεξεργαστή ο οποίος έχει τα στοιχεία με τα οποία θέλει να κάνει τις συγκρίσεις. Στη συγκεκριμένη περίπτωση Processor1- Processor3 και Processor2- Processor4 (Σχ.1.7).



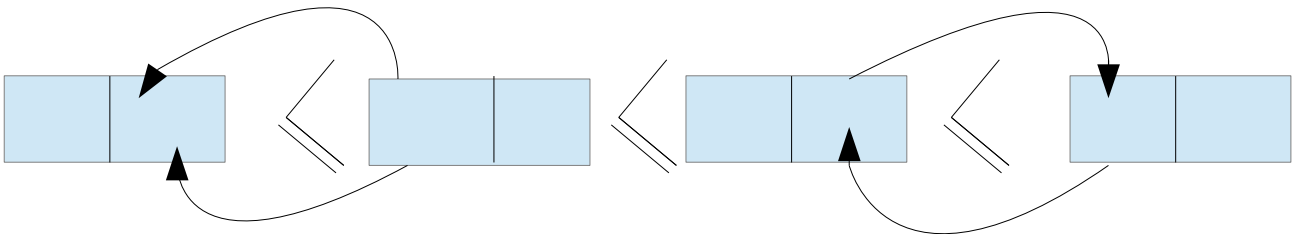
Σχήμα 1.7



## Παράλληλα και Διανεμημένα Συστήματα 2η Εργασία

### Βήμα 6ο

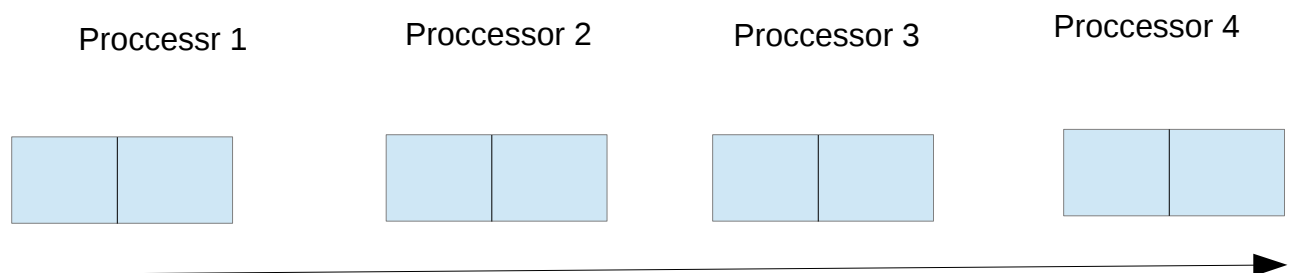
Τώρα η αρχική ακολουθία έχει χωριστεί σε δύο ακολουθίες των οποίων τα στοιχεία της μίας είναι μικρότερα της άλλης και κάθε μία από αυτές είναι χωρισμένη σε δύο επεξεργαστές. Οπότε για να εκτελέσουμε το επόμενο βήμα της bitonic merging πρέπει να εκτελέσουμε τις συγκρίσεις ανάμεσα σε αυτούς τους επεξεργαστές Processor1- Processor2 και Processor3- Processor4(Σχ. 1.8).



Σχήμα 1.8

### Βήμα 7ο

Τέλος έχουμε σε κάθε επεξεργαστή μια ακολουθία της οποίας τα στοιχεία ή δεν υπάρχουν ή είναι ίσα με των άλλων επεξεργαστών, όπως φαίνεται στο παραπάνω σχήμα (1.8). Οπότε τώρα μπορούμε να εκτελέσουμε το bitonic merging σε κάθε επεξεργαστή ξεχωριστά χωρίς να επικοινωνεί με κανέναν άλλο. Έτσι καταλήγουμε σε μια ενιαία αύξουσα ακολουθία (Σχ.1.9).



Σχήμα 1.9

## Παράλληλα και Διανεμημένα Συστήματα 2η Εργασία

### 2.Επιπλέον σχόλια για την τρόπο που τρέξαμε τους αλγορίθμους

Την παράλληλη υλοποίηση της bitonic την τέξαμε Grid όμως για λόγο καθυστέρησης εξαγωγής των αποτελεσμάτων την τρέξαμε και στο user interface του Hellas Grid. Για την πρώτη περίπτωση δυμιουργήσαμε ένα script στην python που δημιουργούσε τα shell script (script.sh) με όλα τα απαραίτητα για τις μετρήσεις με ορίσματα p και q. Το όρισμα q δίνεται απευθείας στο πρόγραμμα που έχουμε υλοποιημένο στη C ενώ το q δίνεται με τη μορφή nodes=x:rrn=y όπου επιλέγαμε πάντα το x=1 εκτός για q=7 που έπρεπε να είναι nodes=2:rrn=64 μιας και δεν υπάρχει ένα node με 128 proccesses. Οπότε μέσα στο στο πρόγραμμά μας παίρνουμε τον αριθμό των proccesses από την εντολή `MPI_Comm_size(MPI_COMM_WORLD, &numtasks)`. Το script υπάρχει στον φάκελο `/uiscripts/parallel_bitonic/GRID/uirun.py` μαζί με το αρχείο `mpi-bitonic.c`. Τώρα για να τρέξουμε το πρόγραμμα στο User Interface δημιουργήσαμε πάλι ένα script στην python το οποίο τρέχει για όλες τις περιπτώσεις τον εκτελέσιμο πρόγραμμα με την εντολή `"mpirun -n (2^p) ./mpi-bitonic q"`. Επίσης από το πογραμμα της C γράφουμε σε αρχεία τα αποτελέσματα. Το script μαζί με τον πηγαίο κώδικα στη C υπάρχει στον φάκελο `/uiscripts/parallel_bitonic/UserInterface/uirun.py`. Επίσης η μεταγλώττιση του κώδικα γίνεται με make και το Makefile υπάρχει στην αρχή του φακέλου μας. Τα σειριακά προγράμματα της bitonic και της qsort τα τρέξαμε στο User Interface γιατί τρέξαμε ορισμένα στο Grid και είδαμε ότι δεν έχει σημαντική διαφορά στο χρόνο εκτέλεσης. Η εκτέλεση όλων των περιπτώσεων έγινε κι εδώ με ένα script στην python που έτρεχε το εκάστοτε εκτελέσιμο με ορίσματα το q και το σύνολο των επεξεργαστών ( $2^p$ ). Τα αποτελέσματα και εδώ γράφονται απευθείας σε αρχεία. Τα αρχεία βρίσκονται στους φακέλους `/uiscripts/bitonic_serial` και `/uiscripts/qsort_serial`. Τέλος όλα τα προγράμματα για καλύτερη αξιοπιστία των αποτελεσμάτων, τρέχουν για 10 επαναλήψεις και παίρνουμε το μέσο όρο των μετρήσεων. Όλες οι μετρήσεις βρίσκονται στον φάκελο results.

## Παράλληλα και Διανεμημένα Συστήματα 2η Εργασία

### 3.Ανάλυση του ελέγχου ορθότητας της μεθόδου test()

Για τον έλεγχο ορθότητας των αποτελεσμάτων της μεθόδου που χρησιμοποιήσαμε, δηλαδή αν η τελική ακολουθία είναι όντως ταξινομημένη κατά αύξουσα σειρά, χρησιμοποιήσαμε τοπικά σε κάθε επεξεργαστή τη συνάρτηση test() που μας δόθηκε. Όμως δε μπορούσαμε να τη χρησιμοποιήσουμε για όλο το πρόβλημα γιατί θα έπρεπε να μεταφέρουμε όλα τα δεδομένα από όλους τους επεξεργαστές σε έναν γεγονός που θα δημιουργούσε έναν πίνακα που, όπου σε μεγαλύτερα προβλήματα, μπορεί να μη χωρούσε στη μνήμη. Έτσι εκτελούμε τοπικά σε κάθε επεξεργαστή τη συνάρτηση test() ελαφρός τροποποιημένη έτσι ώστε να επιστρέφει έναν ακέραιο ο οποίος θα είναι 1 αν πέτυχε το test και 0 αν απέτυχε. Στη συνέχεια στέλνουμε σε έναν επεξεργαστή (συγκεκριμένα στον pid=0) έναν πίνακα τριών στοιχείων, το πρώτο είναι ο ακέραιος που επιστρέφει η test(), το δεύτερο το πρώτο (ελάχιστο) στοιχείο του πίνακα του κάθε επεξεργαστή και το τρίτο το τελευταίο (μέγιστο) στοιχείο του πίνακα του κάθε επεξεργαστή. Τέλος στον επεξεργαστή pid=0 αθροίζουμε σε μια μεταβλητή sum τις επιστρεφόμενες τιμές της test() από κάθε επεξεργαστή. Επίσης ελέγχουμε με τον ίδιο τρόπο που δουλεύει η test() αν το μέγιστο στοιχείο από κάθε επεξεργαστή είναι μικρότερο ή ίσο από το ελάχιστο στοιχείο του αμέσως επόμενου επεξεργαστή, αν ισχύει αυτό για όλους η μεταβλητή pass θα παραμείνει 1 αλλιώς θα γίνει 0. Έτσι ελέγχουμε αν πληρούνται ταυτόχρονα οι προϋποθέσεις το sum να είναι ίσο με τον αριθμό των επεξεργαστών (numtasks) και το pass να ισούτε με 1, τότε εκτυπώνουμε το μήνυμα "TEST PASSEd" αλλιώς εκτυπώνουμε το μήνυμα "TEST FAILEd". Ο υλοποιημένος κώδικας φαίνεται παρακάτω.

```
if(taskid==MASTER){  
  
    int i;  
    int sum,pass=1;  
    sum=test();  
    for(i=1;i<numtasks;i++){  
  
        MPI_Recv(&c[3*i], 3, MPI_INT,i, FROM_WORKER,MPI_COMM_WORLD, &status);  
        sum=c[3*i];  
        if (i==0) pass &= (a[N-1] <= c[3*i+1]);  
        else pass &= (c[3*(i-1)+2] <= a[3*i+1]);  
    }  
    if(sum==numtasks && pass) printf("TEST PASSEd \n");  
    else printf("TEST FAILEd \n");  
}  
else{  
    b[0]=test();  
    b[1]=a[0];  
    b[2]=a[N-1];  
    MPI_Send (b,3,MPI_INT,0, FROM_WORKER,MPI_COMM_WORLD);  
}  
MPI_Barrier(MPI_COMM_WORLD);
```

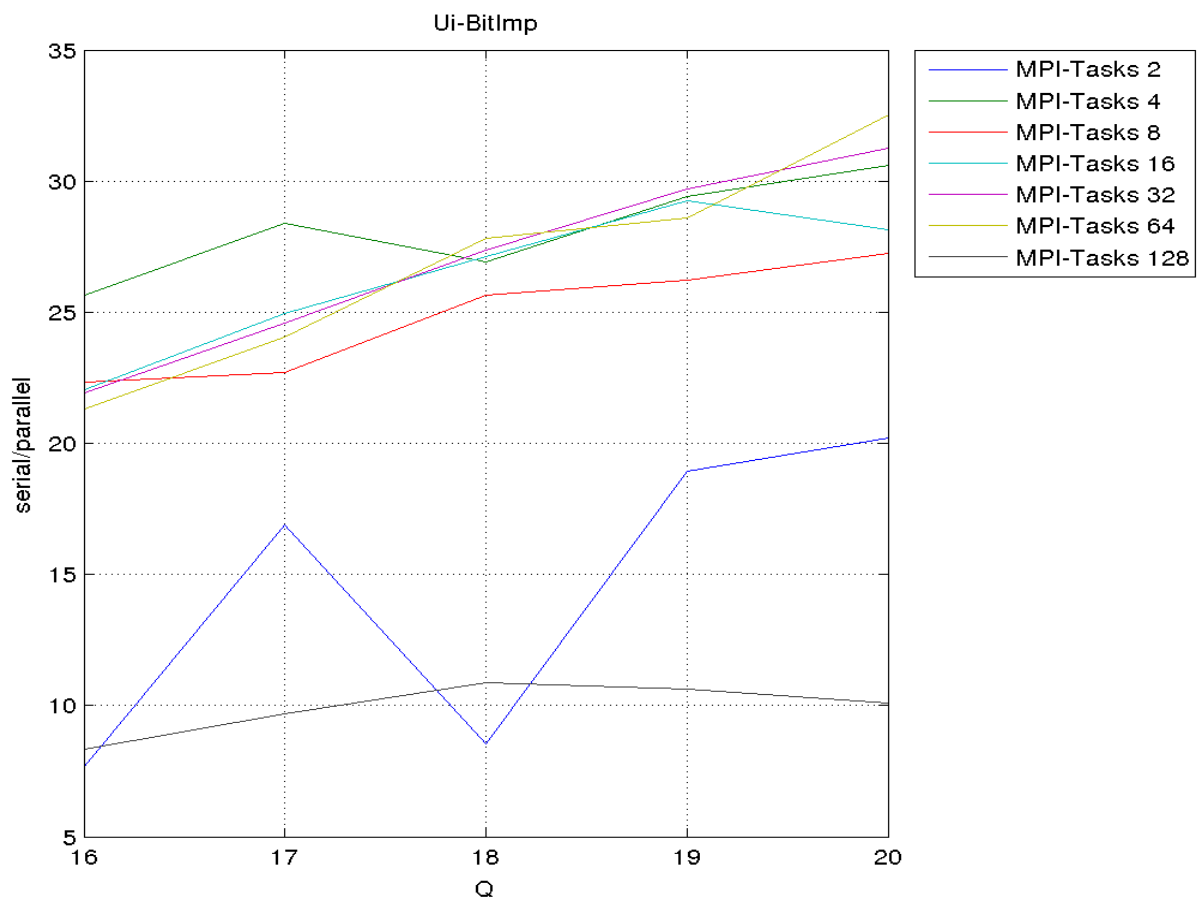
## Παράλληλα και Διανεμημένα Συστήματα 2η Εργασία

### 4. Συγκριτικά διαγράμματα σειριακής και παράλληλης υλοποίησης

Επειδή Το grid δεν μας είχε δώσει τα δεδομένα ακόμα .Τρέξαμε στο Ui του hellas grid και πήραμε τα εξής Αποτελέσματα Ακολουθούν του Grid

Στα σχεδιαγράμματα φαίνεται ο λόγος του χρόνου του σειριακού κώδικα προς τον παράλληλο

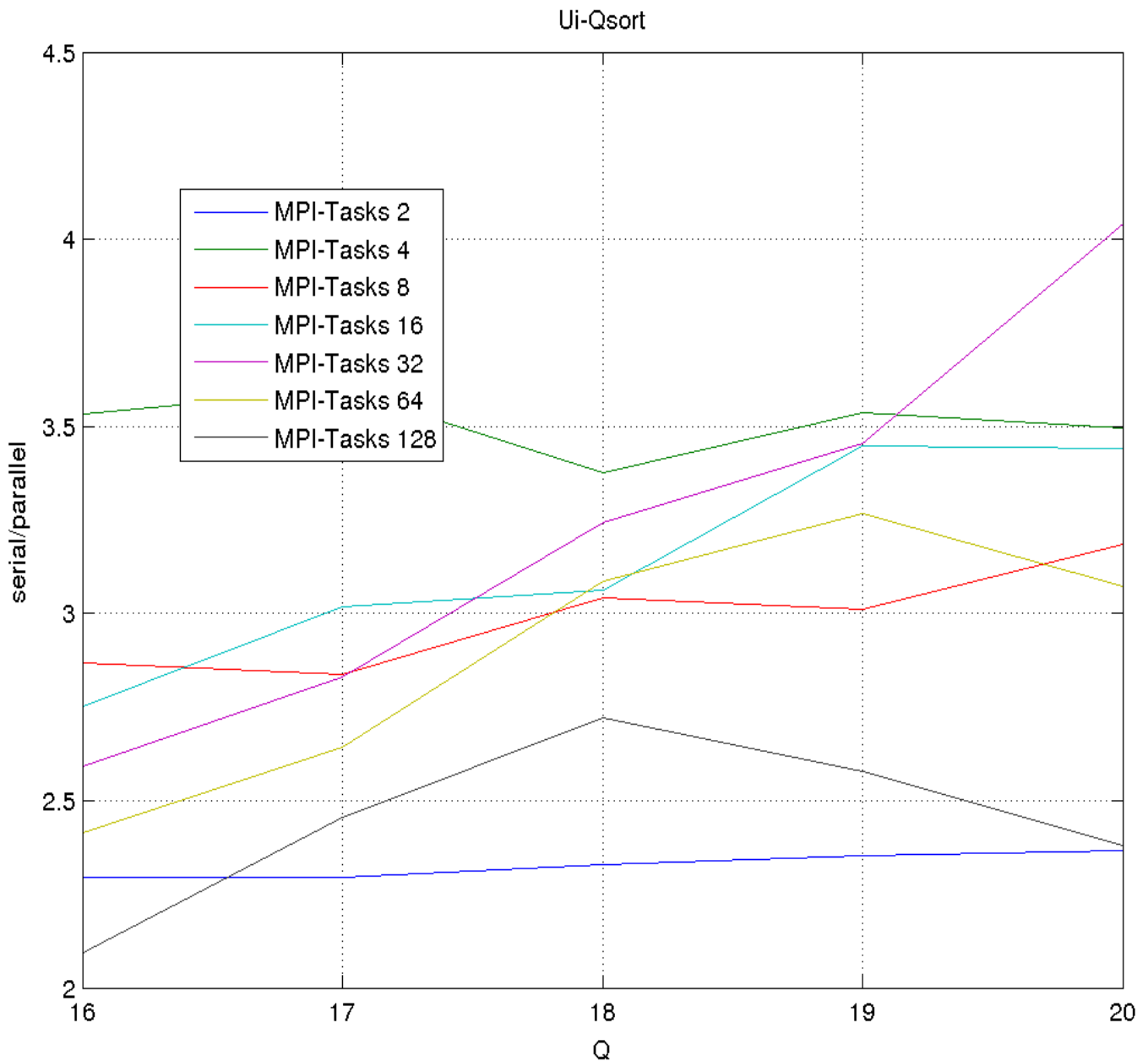
Σε αυτο το σχήμα συγκρίνεται ο Παράλληλος με την Bitonic (1 πυρήνα παράλληλη μέθοδος)



Παρατηρούμε μεγάλη βελτίωση μέχρι τα 4 processes Έπειτα Επειδή δεν έχει άλλους πυρήνες (UI) η επικοινωνία αναμεσα στα task κοστίζει σημαντικά στην απόδοση (στα 128 πέφτει κάτω από τα 2!) Επίσης όσο αυξάνεται το μέγεθος του προβλήματος τόσο περισσότερο κερδίζουμε στη παραλληλοποίηση. Καθώς Η επικοινωνία πιάνει μικρότερο ποσοστό στο χρόνο

## Παράλληλα και Διανεμημένα Συστήματα 2η Εργασία

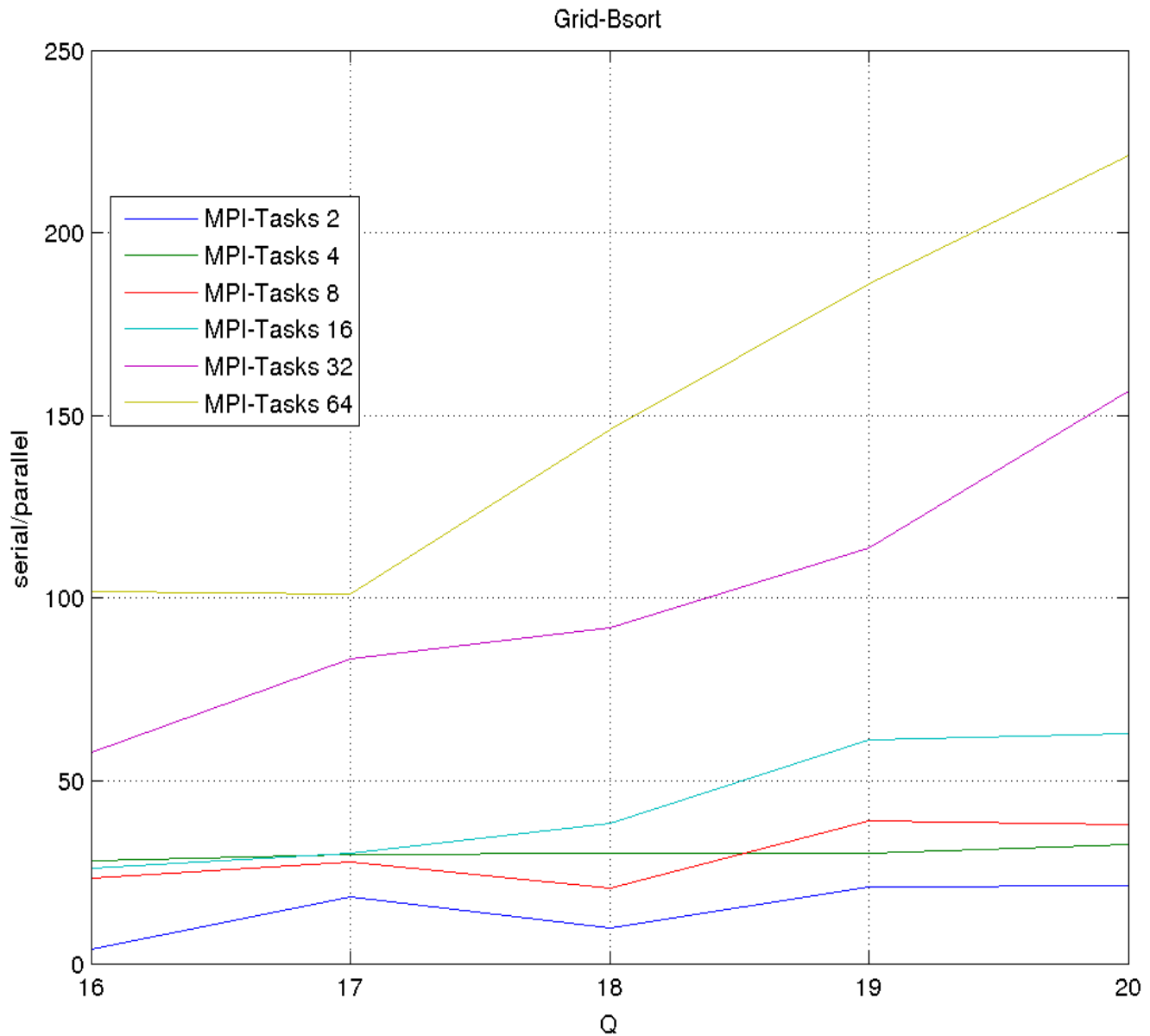
Σε αυτο το σχήμα συγκρίνεται ο Παράλληλος με την qsort



Παρατηρούμε αντιστοιχα αποτελέσματα με την bsort με Την Διαφορα ότι έχουμε βελτίωση κοντα στον αριθμο των επεξεργαστών Σημειώνουμε Ότι η παράλληλη μεθοδος μας χρησιμοποιεί την bsort για επικοινωνία επεξεργαστών και την qsort για ταξινόμηση τοπική

## Παράλληλα και Διανεμημένα Συστήματα 2η Εργασία

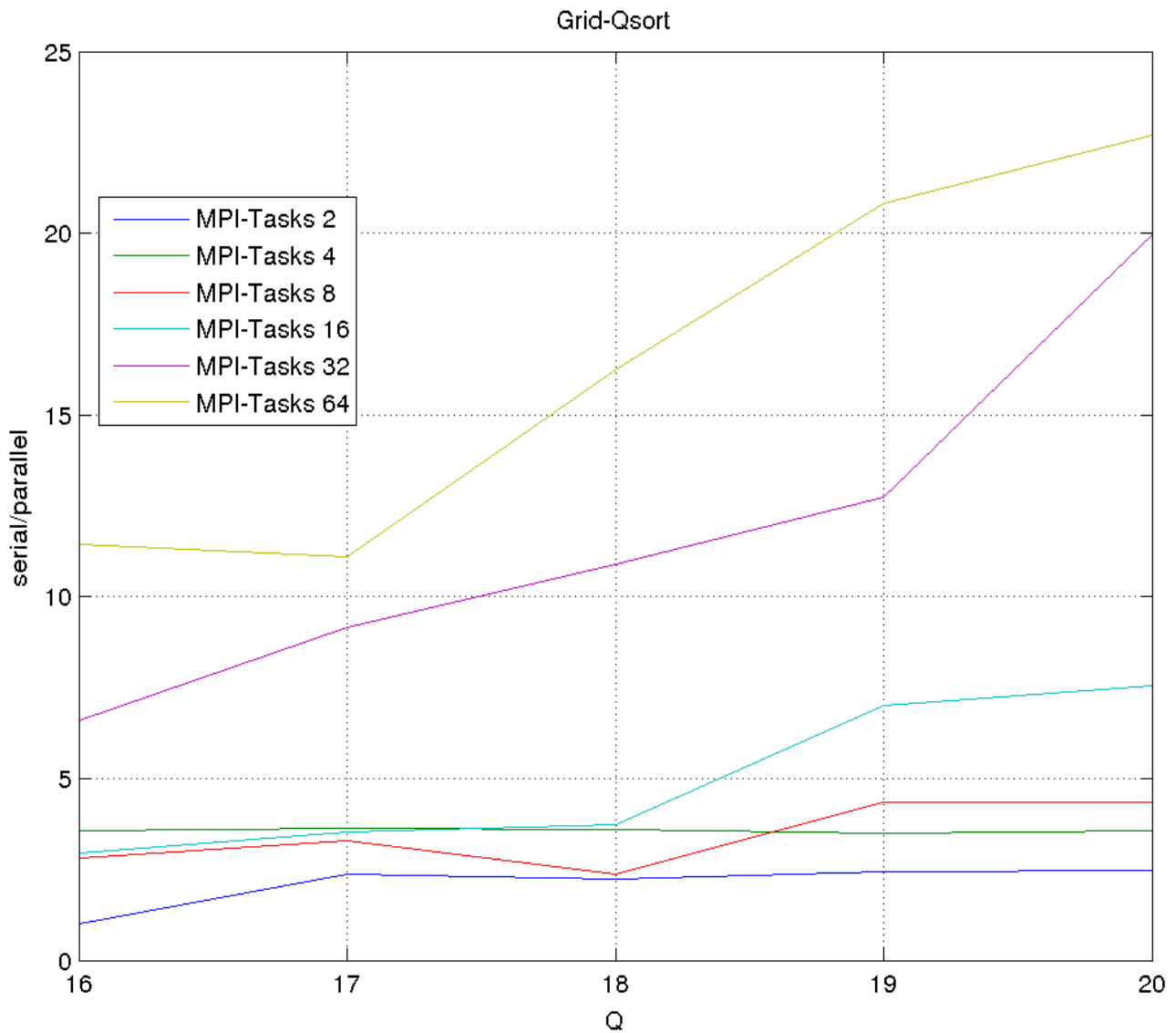
Παίρνοντας τα αποτελέσματα από το Grid είχαμε τα Εξής



Σε αυτό το σχήμα συγκρίνεται ο Παράλληλος με την Bitonic (1 πυρήνα παράλληλη μέθοδος)  
Παρατηρούμε τεράστια βελτίωση. Κάθε φορά που αυξάνουμε τους πυρήνες αλλά και τα δεδομένα

## Παράλληλα και Διανεμημένα Συστήματα 2η Εργασία

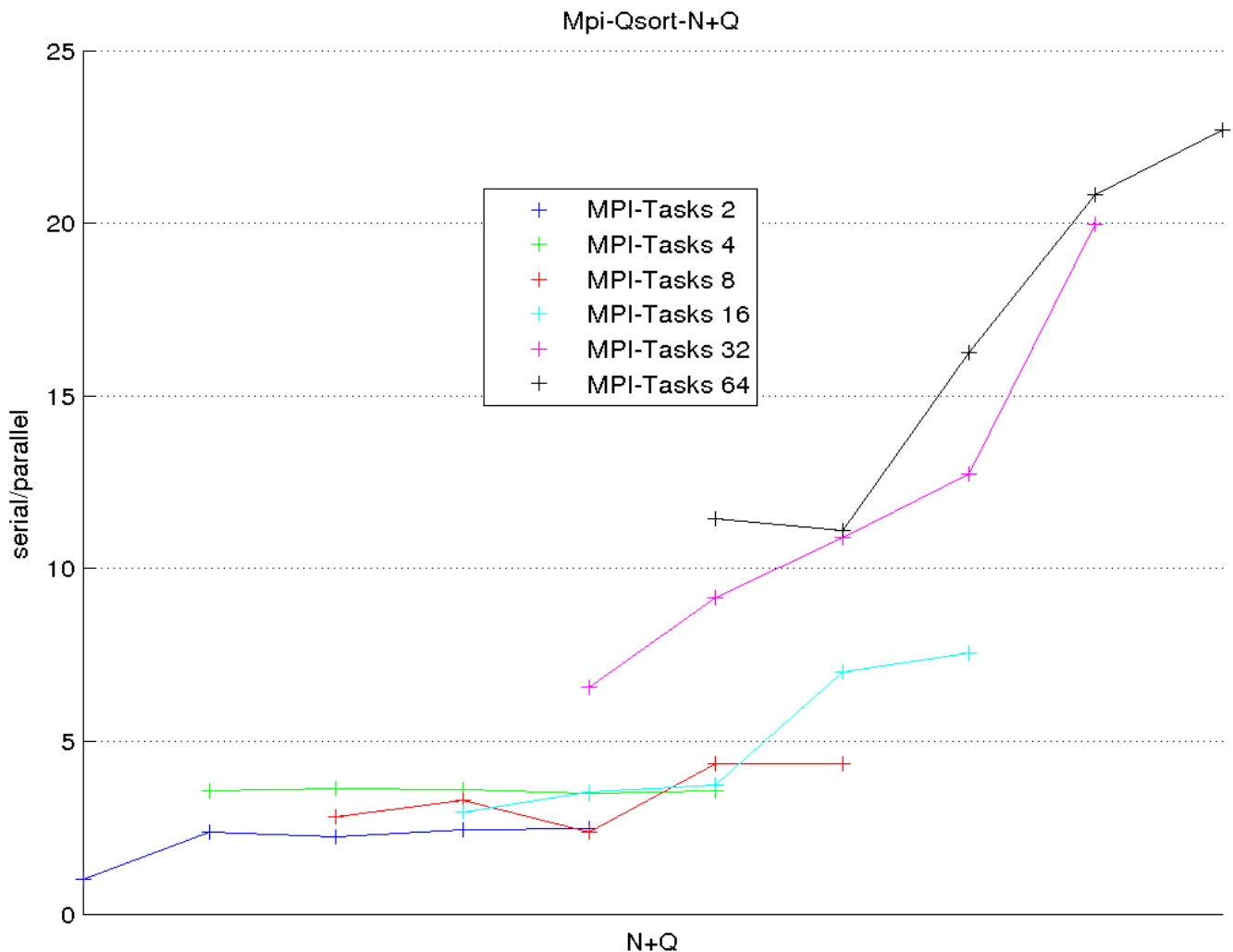
Σε αυτό το σχήμα συγκρίνεται ο Παράλληλος με την qsort



Σημειώνουμε επιπλέον ότι σε ορισμένες περιπτώσεις πετυχαίνουμε καλύτερο χρόνο για το ίδιο πρόβλημα. Αν χρησιμοποιήσουμε λιγότερους πυρήνες και με μεγαλύτερους πίνακες όσο το (εξαρτάται από το μέγεθος του προβλήματος)

## Παράλληλα και Διανεμημένα Συστήματα 2η Εργασία

Αυτο Το σχήμα είναι παρόμοιο με το Παραπάνω με την διαφορά ότι ο Αξονας x απεικονίζει την συνολική  $2^{(N+Q)}$  Πολυπλοκότητα



Συνολικά Παρατηρούμε :

<>Βελτίωση σχεδόν ανάλογη με του πυρήνες που χρησιμοποιούμε μέχρι τα 4 tasks.

<>Μικρότερη της ανάλογης βελτίωσης όσο αύξάνονται οι πυρήνες  
Επειδή αυξάνεται η επικοινωνία (Σε ορισμένες περιπτώσεις παρατηρούμε ότι πετυχαίνουμε ίση ή σχεδόν καλύτερη απόδοση με λιγότερους πυρήνες για το ίδιο πρόβλημα!)

<>Η βελτίωση τείνει στην ανάλογη Όσο αυξάνεται το μέγεθος του αριθμού του κάθε πυρήνα Καθώς Το ποσοστό επικοινωνίας σε σχέση με τις εσωτερικές πράξεις μειώνεται