Ταξινόμηση πίνακα με P > 2 συμμετέχοντες κόμβους

Κατανεμημένος & Παράλληλος Προγραμματισμός

Συλεόπουλος Αναστάσιος

Εισαγωγή

Αντικείμενο

Η παραλληλοποίηση ενός αλγορίθμου ταξινόμησης με περισσότερους από δύο συμμετέχοντες κόμβους. Ο αλγόριθμος Ταξινόμηση με Επιλογή (Selection Sort) θα υλοποιηθεί παράλληλα με την χρήση της MPICH.

Ταξινόμηση με Επιλογή

Στην ταξινόμηση με επιλογή ο πίνακας διαιρείται σε δύο υπό-πίνακες, τον ταξινομημένο και τον αταξινόμητο. Βρίσκει το μικρότερο/μεγαλύτερο (αύξουσα, φθίνουσα) στοιχείο στον αταξινόμητο υπό-πίνακα ανταλλάζοντάς το με το αριστερό στοιχείο του υπό-πίνακα και μετακινεί το όριο (τείχος) του ταξινομημένου υπό-πίνακα κατά ένα στοιχείο δεξιά. Η πολυπλοκότητα του αλγορίθμου είναι $O(n^2)$.

Ψευδοκώδικας

```
Selection-Sort(A[1...n])
    for i ← 1 to n-1 do
        min_pos ← i; min_key ← A[i];
        for j ← i+1 to n do
            if A[j] < min_key then
                min_pos ← j; min_key ← A[j];
        A[min_pos] ← A[i]; A[i] ← min_key;
```

Ανάλυση του προβλήματος

Σκοπός

Πέρα από την υλοποίηση της παραλληλοποίησης του αλγορίθμου Ταξινόμηση με Επιλογή το ζητούμενο είναι το παράλληλο πρόγραμμα που θα υλοποιηθεί να επιτυγχάνει και την επιτάχυνση και την κλιμάκωση.

Επιτάχυνση: Ένα συγκριτικό μέτρο το οποίο μας δίνει το κέρδος από τη παράλληλη εκτέλεση με βάση την σειριακή εκτέλεση του προγράμματος.

Κλιμάκωση: Η ικανότητα του συστήματος να αυξάνει την επίδοση του προγράμματος αυξάνοντας των αριθμό των κόμβων.

Προβλήματα

Αρχίζουν τα ερωτήματα "τι θα παραλληλίσω;", "πως θα το παραλληλίσω;", "τι θα στείλω", "θα δουλέψει ο master;" πολλά τα ερωτήματα και λίγες οι απαντήσεις.
Πρέπει να βρεθεί ένας τρόπος ώστε να έχουμε λίγες επικοινωνίες και σωστή διαμοίραση του προβλήματος στους εκάστοτε διαθέσιμους κόμβους. Η κλασσική τεχνική παραλληλισμού της for {for(i=myid;i<N;i+=P)} δεν έχει κανένα αποτέλεσμα στο πρόβλημα της ταξινόμησης διότι η κάθε γραμμή του πίνακα δεν μπορεί να λειτουργήσει αυτόνομα θα χρειαστεί και τις άλλες γραμμές ώστε να γίνουν οι συγκρίσεις για να ταξινομηθούν σωστά τα στοιχεία του πίνακα.
Αν κάθε κόμβος επικοινωνεί με τους υπόλοιπους για το τι στοιχείο έχει; Μεγάλος αριθμός επικοινωνιών, ότι πάμε να κερδίσουμε σε χρόνο θα το χάσουμε λόγω επικοινωνίας.
Αν διαμοιράσουμε τον πίνακα στους κόμβους, κάθε κόμβος ταξινομήσει το κομμάτι του και στο τέλος να συγκεντρώσουμε τα κομμάτια; Δεν θα έχουμε ταξινομημένο πίνακα θα χρειαστεί επιπλέον δουλειά ώστε να ενωθούν σωστά τα κομμάτια.
Ωραία αυτή η λύση, αλλά είναι αποδοτική;

Σχεδίαση της λύσης

Επικοινωνία

Θα σταθούμε στην τελευταία πρόταση και θα την αναλύσουμε ως προς τις επικοινωνίες. Να διαμοιράσουμε τον πίνακα στους κόμβους. Αυτό γίνεται με την εντολή MPI_Scatter, η οποία παίρνει στα ορίσματά της τον πίνακα τον οποίο θέλουμε να διαμοιράσουμε, την ποσότητα των στοιχείων και τον πίνακα υποδοχής. Για να μπορούν να αποθηκευτούν τα δεδομένα θα πρέπει να δημιουργηθούν οι κατάλληλοι πίνακες υποδοχής με το σωστό μέγεθος. Άρα πριν την εκτέλεση της MPI_Scatter θα χρειαστεί να στείλουμε σε όλους του κόμβους τη τιμή του κομματιού ώστε να δημιουργηθεί ο πίνακας υποδοχής. Αυτό γίνεται με την εντολή MPI_Bcast. Αφού ολοκληρωθεί η ταξινόμηση θα χρειαστεί να συγκεντρωθούν τα αποτελέσματα. Αυτό γίνεται με την εντολή MPI_Gather η οποία παίρνει στα ορίσματά της τον πίνακα προς συγκέντρωση, την ποσότητα των στοιχείων και τον πίνακα για την συγκέντρωση των αποτελεσμάτων.
Επομένως θα χρειαστούμε την εντολή MPI-Bcast για την αποστολή της τιμής του κομματιού, την εντολή MPI_Scatter για την διαμοίραση του πίνακα και την εντολή MPI_Gather για την συγκέντρωση των αποτελεσμάτων. Με αυτές τις τρείς εντολές κρατάμε το κόστος επικοινωνίας χαμηλό.

Ένωση των κομματιών

Το δυσκολότερο κομμάτι του προγράμματος είναι η ένωση των κομματιών, γιατί πρέπει να συγκριθούν τα στοιχεία των κόμβων και το μικρότερο/μεγαλύτερο να τοποθετηθεί στην τρέχουσα θέση του ταξινομημένου πίνακα. Οι επιμέρους πίνακες είναι ταξινομημένοι, αυτό κάνει την δουλεία μας λίγο πιο εύκολη γιατί δεν χρειάζεται να συγκριθούν όλα τα στοιχεία παρά μόνο το αρχικό στοιχείο κάθε κόμβου που δεν έχει ταξινομηθεί ακόμα. Κάθε κόμβος θα χρειαστεί έναν δείκτη ο οποίος θα δείχνει στη θέση που βρίσκεται το μικρότερο/μεγαλύτερο στοιχείου προς ταξινόμηση.
Εν συνεχεία, αφού βρεθεί και τοποθετηθεί το στοιχείο στην τρέχουσα θέση του πίνακα ταξινόμησης ο δείκτης του κόμβου δείχνει στο επόμενο στοιχείο του.
Οι συνολικές συγκρίσεις στην χειρότερη περίπτωση, αν N είναι το μέγεθος και p είναι οι κόμβοι, είναι p^2 συγκρίσεις επί N φορές.
Η πολυπλοκότητα της ένωσης είναι $O(N*p^2)$ $p^2<<N$ και η πολυπλοκότητα της ταξινόμησης είναι $O(part^2)$ $part=N/p$ => $O(N^2 / p^2)$.
Άρα η πολυπλοκότητα είναι πολύ μικρότερη από το σειριακό πρόγραμμα.

Υλοποίηση της λύσης

Λύση σε βήματα

1. Δήλωση μεταβλητών

   int myid, numprocs; -> myid: το id του κόμβου || numprocs: το πλήθος των ενεργών κόμβων.
   int N, part; -> N: μέγεθος πίνακα || part: μέγεθος κομματιού
   int *thesi, *Aw; -> thesi: πίνακας δεικτών για τους κόμβους || Aw: πίνακας υποδοχής
   int *A, *As; -> A: μη-ταξινομημένος πίνακας || As: ταξινομημένος πίνακας
   double ta, tt; -> ta: αρχή χρόνου || tt: τέλος χρόνου
   int temp, check; -> temp: ανάθεση τιμής (Selection Sort) || check: ανάθεση τιμής (Ένωση)
   int tp; -> ο δείκτης θέσης του κόμβου
   int done; -> όταν βρεθεί το μικρότερο done=1 αλλιώς 0

2. Αρχική δουλεία του Master

   a. Ο χρήστης πληκτρολογεί το μέγεθος.
   b. Δεσμεύεται μνήμη και γεμίζει ο πίνακας με στοιχεία.
   c. Υπολογίζει την τιμή του κομμάτιου(part) και ξεκινάει την χρονομέτρηση.

```
if(myid == 0){
        printf("\nPlhktrologiste to megethos tou pinaka: ");
        scanf("%d",&N);
        A = malloc(N*sizeof(int));
        As = malloc(N*sizeof(int));
        if(A==NULL || As==NULL){
                printf("\nAdunamia desmeushs mnhmhs!\nTelos programmatos!\n");
                exit(1);
        }
        FilltheArray(A,N); // gemisma pinaka me stoixeia
        part = N/numprocs;  // upologismos kommatiou
        ta = MPI_Wtime(); // arxizei h xronometrish

        thesi=malloc(numprocs*sizeof(int));
        // Anathesh timwn ston (thesi) me thn arxikh thesh tou stoixeiou kathe komvou
        for(i=0;i<numprocs;i++)
                thesi[i]=i*part;
}.
```

3. Αποστολή σε όλους τους κόμβους το part.

```
MPI_Bcast(&part,1,MPI_INT,0,MPI_COMM_WORLD);
```

4. Δέσμευση μνήμης για τον πίνακα που θα εφαρμοστεί ο Selection Sort.

```
Aw=malloc(part*sizeof(int));
```

5. Διαμοίραση του μη-ταξινομημένου πίνακα(A) στον πίνακα υποδοχής(Aw)

```
MPI_Scatter(&A[0],part,MPI_INT,&Aw[0],part,MPI_INT,0,MPI_COMM_WORLD);
```

Υλοποίηση της λύσης

6. Εφαρμογή του αλγορίθμου με αύξουσα σειρά.

```
for(i=0;i<part;i++){
      for(j=i+1;j<part;j++){
            if(Aw[i]>Aw[j]){
                  temp=Aw[i];
                  Aw[i]=Aw[j];
                  Aw[j]=temp;
             }
       }
}
```

7. Συγκέντρωση αποτελεσμάτων των ταξινομημένων πινάκων.

```
MPI_Gather(&Aw[0],part,MPI_INT,&A[0],part,MPI_INT,0,MPI_COMM_WORLD);
```

8. Τελική δουλεία του Master.

a. Γίνεται η ένωση των ταξινομημένων πινάκων.
b. Σταματά η χρονομέτρηση και εκτυπώνεται ο χρόνος εκτέλεσης.

```
if(myid==0){
        /*
          Gia kathe thesi tou taksinomhmenou pinaka(As) vriskei
          to mikrotero stoixeio tou pinaka Aw tou kathe komvou.
          Anathetei thn timh ston pinaka(As) kai auksanei ton deikth
          thesis apo ton komvo pou vrike to stoixeio kata ena.
        */

        for(i=0;i<N;i++){
             tp=0;
            done=0;
            while(done!=1){
                if(thesi[tp]<(tp+1)*part){
                      check = thesi[tp];
                      for(j=tp+1;j<numprocs;j++){
                            if(A[check]>A[thesi[j]] && thesi[j] < (j+1)*part){
                                  check = thesi[j];
                                   tp=j;
                             }
                       }
                       done=1;
                }else
                      tp++;
            }
            As[i]=A[check];
            thesi[tp]++;
        }

        tt = MPI_Wtime(); // telos xronometrisis
        printf("\nO xronos ekteleshs: %.16f\n",tt-ta);
}
```

9. Τέλος προγράμματος.

```
MPI_Finalize();
```

Επεξήγηση του βήματος 8

Διευκρίνιση: Ο πίνακας thesi έχει μέγεθος τον αριθμό των κόμβων(numprocs). Κάθε θέση του πίνακα αντιστοιχεί σε έναν κόμβο και παίρνει τιμές τις θέσεις όπου βρίσκονται τα στοιχεία (i*part έως (i+1)*part i ={0 έως numprocs }).

Παράδειγμα:

Έστω numprocs=3 N=12 part=4 ο πίνακας thesi θα είναι:

thesi[0]	thesi[1]	thesi[2]
0	4	8

thesi[0] αναφέρεται για τον 0^{ο} κόμβο
thesi[1] αναφέρεται για τον 1^{ο} κόμβο και
thesi[2] αναφέρεται για τον 2^{ο} κόμβο.

Οι τιμές που μπορεί να πάρει ο πίνακας είναι:
thesi[0] (0 έως 4), thesi[1] (4 έως 8), thesi[2] (8 έως 12)

Κώδικας:

```
for(i=0;i<N;i++){
   tp=0;
   done=0;
   while(done!=1){
      if(thesi[tp]<(tp+1)*part){
            check = thesi[tp];
            for(j=tp+1;j<numprocs;j++){
                  if(A[check]>A[thesi[j]] && thesi[j] < (j+1)*part){
                        check = thesi[j];
                        tp=j;
                  }
            }
            done=1;
       }else
             tp++;
    }
    As[i]=A[check];
    thesi[tp]++;
}
```

Όσο δεν έχει βρεθεί το μικρότερο στοιχείο (while(done!=1)) για να τοποθετηθεί στον ταξινομημένο πίνακα ελέγχεται ο δείκτης του κόμβου(tp) αν έχει ταξινομήσει όλα τα στοιχεία του η όχι. Αν τα έχει ταξινομήσει τότε ο δείκτης κόμβων μεταφέρεται στον επόμενο κόμβο(tp++). Αν δεν τα έχει ταξινομήσει αποθηκεύεται η θέση του στοιχείου προς ταξινόμηση στη μεταβλητή check. Το στοιχείο συγκρίνεται με όλα τα στοιχεία προς ταξινόμηση. Αν βρεθεί κάποιο στοιχείο μικρότερο αποθηκεύεται και συνεχίζεται ο έλεγχος για τους υπόλοιπους κόμβους. Όταν τελειώσει η σύγκριση των στοιχείων θα έχει βρεθεί το μικρότερο στοιχείο (done=1) όπου τοποθετείται στην τρέχουσα θέση του ταξινομημένου πίνακα (As[i]=A[check]) και η τιμή του δείκτη του κόμβου αυξάνεται κατά ένα (thesi[tp]++). Αυτή η διαδικασία γίνεται μέχρι να γεμίσει ο ταξινομημένος πίνακας.

Ορθότητα & Επιδόσεις

Ορθότητα

```
sakis@ThinkPad:~/MCs TEI/Distributed & Parallel Systems DPS/Sabbas/project$ mpiexec -np 3 ./selsort_p
Selection Sort: Parallhlo programma gia p > 2

Plhktrologiste to megethos tou pinaka: 12
O arithmos komvwn(numprocs): 3 To megethos(N): 12 To kommati(part): 4

Mh-taksinomhmenos pinakas(A)
83 |86 |77 |15 |93 |35 |86 |92 |49 |21 |62 |27 |

~Komvos 0 -> not-sorted pinakas(Aw)
83 |86 |77 |15 |

~Komvos 1 -> not-sorted pinakas(Aw)
93 |35 |86 |92 |

~Komvos 2 -> not-sorted pinakas(Aw)
49 |21 |62 |27 |

******** SELECTION SORT DONE ********

>Komvos 0 -> sorted pinakas(Aw)
15 |77 |83 |86 |

>Komvos 1 -> sorted pinakas(Aw)
35 |86 |92 |93 |

>Komvos 2 -> sorted pinakas(Aw)
21 |27 |49 |62 |

O pinakas(A) meta thn GATHER
15 |77 |83 |86 |35 |86 |92 |93 |21 |27 |49 |62 |

pinakas deiktwn(thesi)
0 |4 |8 |

O taksinomhmenos pinakas(As)
15 |21 |27 |35 |49 |62 |77 |83 |86 |86 |92 |93 |
sakis@ThinkPad:~/MCs TEI/Distributed & Parallel Systems DPS/Sabbas/project$ _
```

Σχολιασμός: Βλέπουμε τη σωστή λειτουργία του αλγορίθμου σε κάθε φάση, στον διαχωρισμό, τη ταξινόμηση, τη συγκέντρωση και την τελική ένωση με τη σωστή σειρά.

Ορθότητα & Επιδόσεις

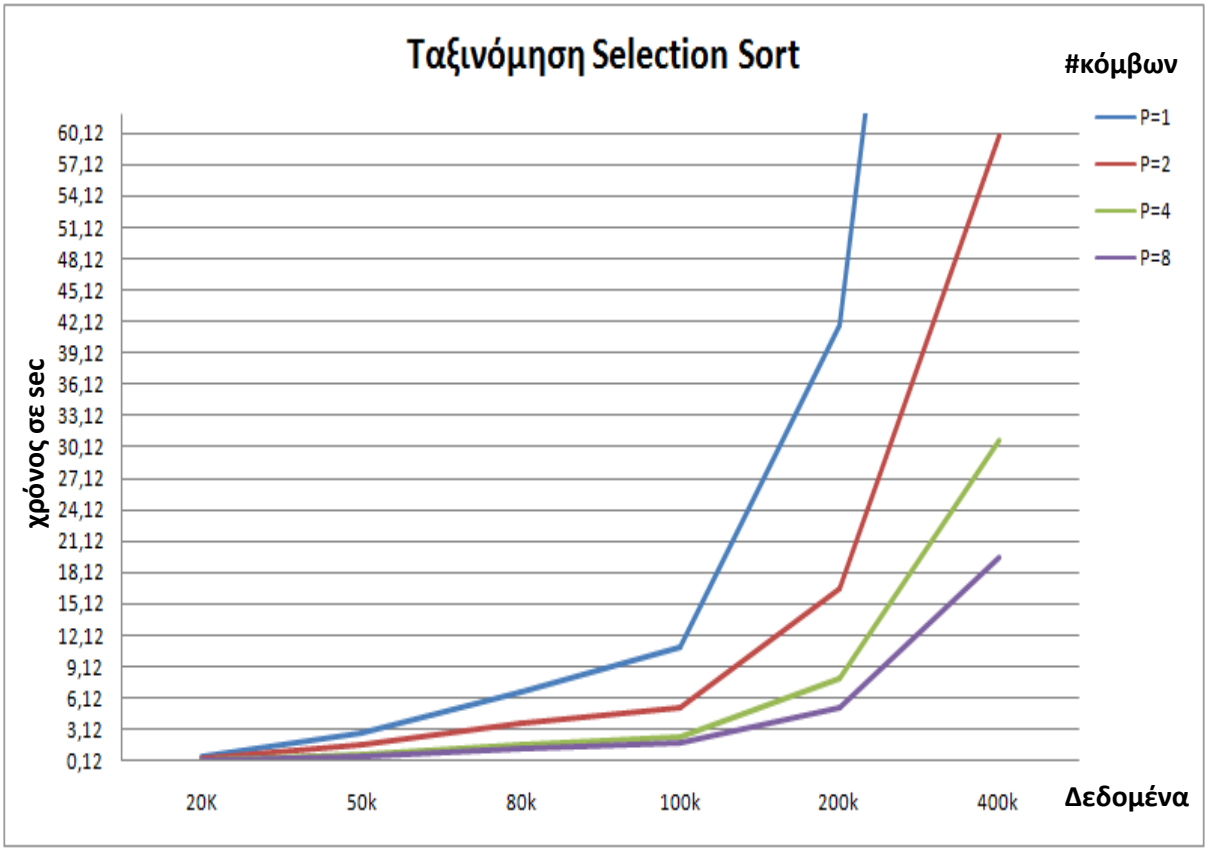
Επιδόσεις

```
sakis@ThinkPad:~/MCs TEI/Distributed & Parallel Systems DPS/Sabbas/project$ mpiexec ./sel_sort_s
Selection Sort: Seiriako programma
Plhktrologiste to megethos tou pinaka: 200000


Xronos ektelesis: 41.8082442283630371
sakis@ThinkPad:~/MCs TEI/Distributed & Parallel Systems DPS/Sabbas/project$ mpiexec -np 2 ./selsort_p
Selection Sort: Parallhlo programma gia p > 2
Plhktrologiste to megethos tou pinaka: 200000
O arithmos komvwn(numprocs): 2 To megethos(N): 200000 To kommati(part): 100000

O xronos ekteleshs: 16.5355484485626221
sakis@ThinkPad:~/MCs TEI/Distributed & Parallel Systems DPS/Sabbas/project$ mpiexec -np 4 ./selsort_p
Selection Sort: Parallhlo programma gia p > 2
Plhktrologiste to megethos tou pinaka: 200000
O arithmos komvwn(numprocs): 4 To megethos(N): 200000 To kommati(part): 50000

O xronos ekteleshs: 7.9190628528594971
sakis@ThinkPad:~/MCs TEI/Distributed & Parallel Systems DPS/Sabbas/project$ mpiexec -np 8 ./selsort_p
Selection Sort: Parallhlo programma gia p > 2
Plhktrologiste to megethos tou pinaka: 200000
O arithmos komvwn(numprocs): 8 To megethos(N): 200000 To kommati(part): 25000

O xronos ekteleshs: 5.1667113304138184
sakis@ThinkPad:~/MCs TEI/Distributed & Parallel Systems DPS/Sabbas/project$ mpiexec -np 16 ./selsort_p
Selection Sort: Parallhlo programma gia p > 2
Plhktrologiste to megethos tou pinaka: 200000
O arithmos komvwn(numprocs): 16 To megethos(N): 200000 To kommati(part): 12500

O xronos ekteleshs: 3.2464687824249268
sakis@ThinkPad:~/MCs TEI/Distributed & Parallel Systems DPS/Sabbas/project$ mpiexec -np 32 ./selsort_p
Selection Sort: Parallhlo programma gia p > 2
Plhktrologiste to megethos tou pinaka: 200000
O arithmos komvwn(numprocs): 32 To megethos(N): 200000 To kommati(part): 6250

O xronos ekteleshs: 2.1298882961273193
sakis@ThinkPad:~/MCs TEI/Distributed & Parallel Systems DPS/Sabbas/project$ mpiexec -np 64 ./selsort_p
Selection Sort: Parallhlo programma gia p > 2
Plhktrologiste to megethos tou pinaka: 200000
O arithmos komvwn(numprocs): 64 To megethos(N): 200000 To kommati(part): 3125

O xronos ekteleshs: 2.5781729221343994
sakis@ThinkPad:~/MCs TEI/Distributed & Parallel Systems DPS/Sabbas/project$ _
```

Σχολιασμός: Βλέπουμε με μια γρήγορη ματιά ότι υπάρχει σημαντική βελτίωση της απόδοσης με βάση τη σειριακή υλοποίησης της ταξινόμησης με επιλογή.

Ορθότητα & Επιδόσεις

Γραφική Αναπαράσταση



CPU: Intel I5-2540M Dual Core Quad Threads

Σχολιασμός: Όλες οι μετρήσεις έγιναν κάτω από τις ίδιες συνθήκες. Βλέπουμε την επίτευξη της επιτάχυνσης και της κλιμάκωσης.