ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

ΛΟΓΟΘΕΤΗΣ ΑΛΕΞΙΟΣ-ΑΝΤΩΝΙΟΣ – ICE 20390132

ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΑΡΑΛΛΗΛΟ ΥΠΟΛΟΓΙΣΜΟ – ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 2Η

12/1/2023

Αρχικα έχουμε την δήλωση μεταβλητών και την δημιουργία ενός struct που χρησιμοποιείτε παρακάτω.

```c
#include <stdio.h>
#include "mpi.h"

struct vector{
    float delta;
    int pos;
};

int main(int argc, char** argv){
    int my_rank;
    int nop; // number of processes
    int root = 0;
    int data[100];
    int local_data[100];
    int final_greater,final_less;
    int min, max,n,sum;
    float loc_disp,final_disp;
    float avrg;
    struct vector tmp_vec;
    struct vector max_vec;
    float tmpd;
    int k,ch,tmp;
    int tag=100;
    int prefix[100];
    int local_prefix[100];
    int loc_num_arr[100];
    int displacement[100];
    int lta; //less than average counter
    int gta; //greater than average counter

    MPI_Status status;
```

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &nop);

do{
    avrg=0;
    sum=0;
    if (my_rank == 0){
        // system("clear");
        printf("Input how many numbers: ");
        scanf("%d", &n);


        printf("Input the elements of the array: ");
        for(int i=0; i<n; i++)
            scanf("%d", &data[i]);

        //calculating the min max and average in p0
        max=data[0];
        min=data[0];
        sum+=data[0];

        for(int i=1; i<n; i++){
            sum+=data[i];
            if (data[i]>max){
                max=data[i];
            }
            if (data[i]<min){
                min=data[i];
            }
        }
        avrg=sum/(float)n;
```

Αρχικοποιούμε τον mpi και δημιουργούμε μια επανάληψη για την επαναληπτική διαδικασία του menu. Έπειτα στον μηδέν ζητάμε απο τον χρήστη το πλήθος των αριθμών και τις τιμές τους.Υπολογίζουμε το max ,το min και το average.

```
    avrg=sum/(float)n;

    //calculating the amount of numbers each process is going to take
    for(int i=0;i<nop-1;i++)
        loc_num_arr[i]=n/nop;
    loc_num_arr[nop-1]=(n/nop)+(n%nop);
    //last process takes also the numbers that are left if n%nop!=0

    //calculating displacement for scatterv and gatherv
    for(int i=0;i<nop;i++)
        displacement[i]=(n/nop)*i;
}

//bcasting all the variables that we will need
MPI_Bcast(&n, 1, MPI_INT, root, MPI_COMM_WORLD);
MPI_Bcast(&max, 1, MPI_INT, root, MPI_COMM_WORLD);
MPI_Bcast(&min, 1, MPI_INT, root, MPI_COMM_WORLD);
MPI_Bcast(&avrg, 1, MPI_FLOAT, root, MPI_COMM_WORLD);
MPI_Bcast(loc_num_arr,100,MPI_INT,root,MPI_COMM_WORLD);
MPI_Bcast(displacement,100,MPI_INT,root,MPI_COMM_WORLD);

//scattering the data to each process
MPI_Scatterv(data, loc_num_arr ,displacement, MPI_INT,
&local_data,loc_num_arr[my_rank], MPI_INT, root, MPI_COMM_WORLD);

//counters
lta=0;
gta=0;
loc_disp=0;
```

Έπειτα υπολογίζουμε το πλήθος των αριθμών που θα διαχειριστεί κάθε διεργασία και το αποθηκέυουμε στο loc_num_arr, σε περίπτωση που το πλήθος των αριθμών δεν είναι ακέραιο πολλαπλάσιο με το πλήθος των επεξεργαστών τότε ο τελευταίος επεξεργαστής θα πάρει και τους “περισευόμενους” αριθμούς.Μετά υπολογίζουμε το displacement που θα χρειαστεί η scatterv και η gatherv και κάνουμε broadcast ολες τις μεταβλητές που θα χρειαστούμε για τους υπολογισμούς μας παρακάτω.

```
for(int i=0; i< loc_num_arr[my_rank]; i++){
    if(local_data[i]<avrg){
        lta++; //less than avrg counter
    }
    if(local_data[i]>avrg){
        gta++; //greater than avrg counter
    }
    loc_disp+=(local_data[i]-avrg)*(local_data[i]-avrg); //calculating dispersion

    tmpd=((float)(local_data[i]-min)/(max-min))*100; //calculating delta
    printf("Delta [%d] : %.2f\n",i + (n/nop)*my_rank,tmpd);

    //finding max delta and its pos in original data table
    if(i==0){
        tmp_vec.delta=tmpd;
        tmp_vec.pos=i + (n/nop)*my_rank;
    }else if(tmpd>tmp_vec.delta){
        tmp_vec.delta=tmpd;
        tmp_vec.pos=i + (n/nop)*my_rank;
    }
}
```

Υπολογίζουμε σε κάθε επεξεργαστή ξεχωριστά πόσοι απο τους αριθμούς του είναι μεγαλύτεροι και πόσοι μικρότεροι απο το average και τους κρατάμε στους μετρητές lta και gta.Έπειτα υπολογίζουμε την διασπορά των στοιχείων και την μεταβλητή δέλτα βάση των συναρτήσεων που μας δόθηκαν.Για την εύρεση της μέγιστης δ χρησιμοποιούμε το struct vector για να αποθηκεύουμε στην ίδια μεταβλητή την τιμή αλλα και την θέση στην οποία βρίσκεται.Η θέση υπολογίζεται βάση της θέσης της στον προσωρινό πίνακα του επεξεργαστή ,το rank του επεξεργαστή και το πλήθος των αριθμών που διαχειρίζεται κάθε επεξεργαστής.

```
//if amount of numbers is greater than number of processes
if(n>nop){

    //calculating the first step for the prefix
    local_prefix[0]=local_data[0];
    for(int i=1;i<loc_num_arr[my_rank];i++)
        local_prefix[i]=local_prefix[i-1]+local_data[i];

    //rank 0 only sends but doesnt recv
    if(my_rank==0)
        MPI_Send(&local_prefix[loc_num_arr[my_rank]-1], 1, MPI_INT, my_rank+1, tag, MPI_COMM_WORLD);

    if(my_rank!=0){
        MPI_Recv(&tmp, 1, MPI_INT,my_rank-1, tag, MPI_COMM_WORLD, &status);
        for(int i=0;i<loc_num_arr[my_rank];i++){
            local_prefix[i]+=tmp;  //calculating the second step of the prefix
        }
        if(my_rank!=nop-1){
            MPI_Send(&local_prefix[loc_num_arr[my_rank]-1], 1, MPI_INT, my_rank+1, tag, MPI_COMM_WORLD);
        }
    }

}
```

Στο παραπάνω τμήμα κώδικα υπολογίζουμε το prefix στην περίπτωση που το πλήθος των αριθμών είναι μεγαλύτερο απο το πλήθος των επεξεργαστών.Αρχικά υπολογίζουμε το “τοπικό” prefix του κάθε επεξεργαστή και το αποθηκεύουμε στον πίνακα local_prefix. Έπειτα κάθε επεξεργαστής στέλνει στον επόμενο τον τελευταίο του αριθμό απο τον πίνακα local_prefix για να το προσθέσει στα δικά του στοιχεία και να στείλει το καινούριο του τελευταίο στοιχείο στον επόμενο κ.ο.κ , προυπόθεση οτι ο πρώτος επεξεργαστής δεν κάνει recv και ο τελευταίος δεν κάνει send.

```
//calculating prefix for n=number of processes
if(nop==n){
    MPI_Scan(local_data, prefix,loc_num_arr[my_rank], MPI_INT, MPI_SUM, MPI_COMM_WORLD);
    printf("prefix[%d] == %d \n",my_rank,prefix[0]);
}else if(n>nop){
    MPI_Gatherv(local_prefix,loc_num_arr[my_rank],MPI_INT,prefix,loc_num_arr,displacement,MPI_INT,root,MPI_COMM_WORLD);
}


//gathering the information that we need back at p0
MPI_Reduce(&tmp_vec,&max_vec ,1, MPI_FLOAT_INT, MPI_MAXLOC,root, MPI_COMM_WORLD);
MPI_Reduce(&loc_disp, &final_disp,1, MPI_FLOAT, MPI_SUM,root, MPI_COMM_WORLD);
MPI_Reduce(&gta, &final_greater,1, MPI_FLOAT, MPI_SUM,root, MPI_COMM_WORLD);
MPI_Reduce(&lta, &final_less,1, MPI_FLOAT, MPI_SUM,root, MPI_COMM_WORLD);
```

Στην περίπτωση που το πλήθος των αριθμών είναι ίδιο με το πλήθος των επεξεργαστών τότε τα πράγματα είναι πιο απλά αφου ο mpi μας προσφέρει την συνάρτηση MPI_Scan στην οποία βάζουμε τον τοπικό πίνακα με τους αριθμούς κάθε επεξεργαστή , δήλαδη έναν αριθμο εφόσον n=nop, το operation MPI_SUM και τον πίνακα στον οποίον αποθηκέυετε το τελικό prefix. Για την προηγούμενη περίπτωση τώρα όπου n>nop και ο υπολογισμός του prefix έγινε απο εμάς , κάνουμε gatherv τα δεδομένα στο prefix στον επεξεργαστή root ,στην περιπτώση μας root==0 , για να εκτυπώσουμε εκεί τα αποτελέσματα.

Εκτύπωση αποτελεσμάτων και υλοποίηση menu.

```
if(my_rank==0){

    if(n>nop){
        for(int i=0;i<n;i++){
            printf("prefix[%d] == %d \n",i,prefix[i]);
        }
    }

    printf("Final average: %.2f \n", avrg);
    printf("Final dispersion: %.2f \n", final_disp/(float)n);
    printf("Final greater: %d \n", final_greater);
    printf("Final less: %d \n", final_less);
    printf("Final max: %d |delta: %.2f |pos: %d\n",data[max_vec.pos],max_vec.delta,max_vec.pos);

}
```

```
    if(my_rank==0){
        k=0;
        while(1){
            if(k>0) printf("There is no such option such as %d , please give again!\n",ch);
                printf("\n============MENU============\n");
                printf("---Option 1  :: --Continue--\n");
                printf("---Option 2  :: ----Exit----\n");
                printf("\nGive option  :: ");
                scanf("%d",&ch);
                if(!(ch==1 || ch==2)){
                    k++;
                    system("clear"); /* printing menu and getting the choice to continue or exit from the user*/
                }else break;
        }
        for(int i=1;i<nop;i++)
            MPI_Send(&ch, 1, MPI_INT,i , tag, MPI_COMM_WORLD);
        if(ch==2){
            MPI_Finalize();    /*exiting the programm if ch==2*/
        }
    }else{
        MPI_Recv(&ch, 1, MPI_INT,0, tag, MPI_COMM_WORLD, &status);
        if(ch==2){
            MPI_Finalize();
        }
    }
}while(ch!=2);
```

Ενδεικτικά τρεξίματα.

```
alex@Alex:~/Desktop$ mpicc -o erg2menu erg2menu.c && mpirun -np 3 ./erg2menu
Input how many numbers: 3
Input the elements of the array: 1 2 3
Delta [0] : 0.00
Delta [2] : 100.00
Delta [1] : 50.00
prefix[0] == 1
prefix[1] == 3
prefix[2] == 6
Final average: 2.00
Final dispersion: 0.67
Final greater: 1
Final less: 1
Final max: 3 |delta: 100.00 |pos: 2

============MENU============
---Option 1  :: --Continue--
---Option 2  :: ----Exit----

Give option  :: 1
Input how many numbers: 9
Input the elements of the array: 1 2 3 4 5 6 7 8 9
Delta [0] : 0.00
Delta [1] : 12.50
Delta [2] : 25.00
Delta [3] : 37.50
Delta [4] : 50.00
Delta [5] : 62.50
Delta [6] : 75.00
Delta [7] : 87.50
Delta [8] : 100.00
prefix[0] == 1
prefix[1] == 3
prefix[2] == 6
prefix[3] == 10
prefix[4] == 15
prefix[5] == 21
prefix[6] == 28
prefix[7] == 36
prefix[8] == 45
Final average: 5.00
Final dispersion: 6.67
Final greater: 4
Final less: 4
Final max: 9 |delta: 100.00 |pos: 8

============MENU============
---Option 1  :: --Continue--
---Option 2  :: ----Exit----
```

```
alex@Alex:~/Desktop$ mpicc -o erg2menu erg2menu.c && mpirun -np 4 ./erg2menu
Input how many numbers: 9
Input the elements of the array: 1 2 3 4 5 6 7 8 9
Delta [2] : 25.00
Delta [4] : 50.00
Delta [3] : 37.50
Delta [5] : 62.50
Delta [0] : 0.00
Delta [1] : 12.50
Delta [6] : 75.00
Delta [7] : 87.50
Delta [8] : 100.00
prefix[0] == 1
prefix[1] == 3
prefix[2] == 6
prefix[3] == 10
prefix[4] == 15
prefix[5] == 21
prefix[6] == 28
prefix[7] == 36
prefix[8] == 45
Final average: 5.00
Final dispersion: 6.67
Final greater: 4
Final less: 4
Final max: 9 |delta: 100.00 |pos: 8

============MENU============
---Option 1  :: --Continue--
---Option 2  :: ----Exit----

Give option  :: 2
```