ΕΞΗΓΗΣΗ ΤΟΥ ΚΩΔΙΚΑ:

Ο κώδικας MPI.c είναι ένα πρόγραμμα παράλληλης επεξεργασίας δεδομένων γραμμένο σε C, το οποίο χρησιμοποιεί τη βιβλιοθήκη Message Passing Interface (MPI) για την εκτέλεση στατιστικών υπολογισμών σε ένα σύνολο αριθμών.

Επεξεργάζεται τα δεδομένα ως εξής :

(α) Βρίσκει ένα μέσο όρο

(β) Την μέγιστη τιμή

(γ) Διακύμανση

(δ) Υπολογίζει ένα πίνακα Δελτα

Αρχικοποίηση

Αρχικά, για να χρησιμοποιήσουμε το MPI (Message Passing Interface), πρέπει να γίνει η αρχικοποίηση του περιβάλλοντος. Αν προκύψει κάποιο σφάλμα κατά την αρχικοποίηση, το πρόγραμμα θα διακοπεί (Abort).

MPI_Comm_size() δηλώνει τον αριθμό των διεργασιών στο numtask και MPI_Comm_rank θετει ένα ID στην τρέχουσα διεργασία

```
MPI_Status status;
rc = MPI_Init(&argc, &argv); //initialize MPI environment

if (rc != 0) {
printf ("Error in initialising MPI\n");
MPI_Abort(MPI_COMM_WORLD, rc);
}



MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
```

MENU

Στον κώδικα έχει τοποθετηθεί μία do while για να εμφανίζει ένα menu οπου ο χρήστης μπρορεί να επιλέξει για το αμα θέλει να συνεχίσει ή να σταματήσει το προγραμμα.Η διεργασία με rank 0 εμφανίζει το menu και λαμβάνει την επιλογή του χρήστη.

```
do{
//0 process interacts with user
if (rank == 0) {
    printf("---------MENU---------\n");
    printf("1.Continue\n");
    printf("2.Exit\n");
    printf("----------------------\n");
    printf("Enter your choice : \n");

    scanf("%d",&choice);
```

Η διεργασία 0 στέλνει σε όλες τις αλλες τις διεργασίες την επιλογή (MPI_Send) και αυτες τις λαμβάνουν MPI_Recv .Ετσι τις ενημερώνει για το αν θα πρέπει να συνεχίσουν ή οχι

```
5
6
7       for (int dest = 1; dest < numtasks; dest++) {
8               MPI_Send(&choice, 1, MPI_INT, dest, 0, MPI_COMM_WORLD);
9           }
10      } else {
11              // Workers wait to receive the command
12              MPI_Recv(&choice, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
13      }
14
15
```

DYNAMIC MEMORY ALLOCATION

Αν ο χρήστης δώσει 1 , το πρόγραμμα συνεχίζει . Χρειάζεται επομένως n θετικά στοιχεία απο τον χρήστη. Ο χρηστης δίνει πόσα και ποια θα είναι τα στοιχεία για προχωρήσει το προγραμμα στην επεξεργασία των δεδομένων μας. Γίνεται δυναμικά η δημιουργία του πίνακα X[n]. Τελος ενημερώνει τις υπολοιπες διεργασίες.

```
46
45      if(choice == 1){
44
43          if (rank == 0){
42
41              printf("Continuing...\n");
40
39              //giving elements from user
38              printf("Enter the number of n elements :\n");
37
36              do{
35              scanf("%d",&n);
34              }while(n<=0);//want a positive number
33
32
31              //initialize dynamically the array
30              X=(double*)malloc(n*sizeof(double));
29
28              printf("Enter %d elements :\n",n);
27              for(int i=0;i<n;i++){
26                  scanf("%lf",&X[i]);
25              }
24          }
23
22              if (rank == 0) { //first process sends n to all other processes
21              for (int dest = 1; dest < numtasks; dest++) {
20                  MPI_Send(&n, 1, MPI_INT, dest, 10, MPI_COMM_WORLD);
19              }
18               } else {
17                  MPI_Recv(&n, 1, MPI_INT, 0, 10, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
16              }
15
```

N%P!=0

Οι διεργασίες θα πρέπει να χειρίζονται τα δεδομένα ακόμα κι αν δεν γίνεται ίση κατανομή στοιχείων ανα διεργασία. Οπότε η διεργασία 0 θα πάρει τα επιπλέον στοιχεία . Αν εχουμε δηλαδή 4 διεργασίες και n =13 τοτε ο επεξεργαστής rank 0 θα πάρει 4 απο αυτα στοιχεία και οι υπόλοιποι 3.

επιπλέον εχουμε ένα local_n οπου δείχνει πόσα στοιχεία θαχει η κάθε διεργασία. Γίνεται και εδώ ενας δυναμικός πίνακας local_X[local_n] για την κάθε διεργασία και τα στοιχεία της.

Εχουμε και ένα offset για της διαχείριση n%P !=0 (οπου P διεργασίες). Τέλος γίνεται πάλι MPI_Send() , MPI_Recv() για την διανομή στοιχείων.

```
            //handling n%p != 0
            //process 0 will take the extra elements if n%p != 0

            int elemperproc = n/numtasks; //number of elements per process
            int remainder = n % numtasks; //remaining elements
            int local_n ;


            if (rank == 0) {
                local_n = elemperproc + remainder; //process 0 takes the extra elements
             }else{
                local_n = elemperproc;
             }

             local_X =(double*)malloc((local_n*sizeof(double)));// allocate local array for each pro
cess

             if(rank == 0){
                for (int i =0;i<local_n;i++){
                    local_X[i] = X[i];
                }


             int offset = local_n; //starting point for each process for example if proc 0 takes [0,
1,2] (3 elements) then rank 1 will start from 3 till 3+elemperproc-1
             for (int dest =1 ;dest < numtasks; dest++){
                MPI_Send(&X[offset], elemperproc, MPI_DOUBLE, dest, 0, MPI_COMM_WORLD);
                offset += elemperproc;
             }
        }else {
                MPI_Recv(local_X,local_n, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
             }
```

(α) τη μέση τιμή (μ) των στοιχείων του διανύσματος X:

Αυτή είναι η πρώτη επεξεργασία των στοιχείων μας :

$$\mu = (x0+x1+x2+\ldots+xn-1) / n$$

Έχουμε ένα local_sum για κάθε διεργασία ,δηλαδή x0+x1+x2+…+xn-1. Έπειτα η κάθε διεργασία στέλνει στην διεργασία 0 το local_sum της και η διεργασία 0 την παίρνει και την αθροίζει σε total_sum . Τέλος την διαιρεί / n και την βάζει στην μεταβλητή average

```
            //(a) avg calculation
            double local_sum =0.0;
            for (int i=0;i<local_n;i++){
               local_sum += local_X[i]; // each process calculates its local sum
            }

            double total_sum=0.0;
            double average;
            if (rank ==0){
               total_sum = local_sum;
               double temp_sum;
               for (int source =1; source < numtasks; source++){
                   MPI_Recv(&temp_sum, 1, MPI_DOUBLE, source, 20, MPI_COMM_WORLD, MPI_STATUS_IGNORE
);//receive local sums from other processes
                   total_sum += temp_sum;
               }
               average = total_sum / n;
               printf("Total Sum = %lf\n", total_sum);
               printf("Average = %lf\n", average);
               for (int dest = 1; dest < numtasks; dest++) {
                   MPI_Send(&average, 1, MPI_DOUBLE, dest, 25, MPI_COMM_WORLD);// send average to o
ther processes
               }
            } else {
              MPI_Send(&local_sum, 1, MPI_DOUBLE, 0, 20, MPI_COMM_WORLD);
               MPI_Recv(&average, 1, MPI_DOUBLE, 0, 25, MPI_COMM_WORLD, &status);
            }
```

(β) τη μέγιστη τιμή (m) των στοιχείων του διανύσματος X:

Με τον ίδιο τρόπο έχουμε ένα local_max οπου η κάθε διεργασία θα επιστρέφει μέσο MPI_Recv την μέγιστη τιμή της. Η διεργασία 0 παίρνει την τοπική αυτή μέγιστη τιμή και την συγκρίνει με την global_max .

Τέλος την εμφανίζει.

```
            //(b)max calculation , each process calculates its local max
            double local_max = local_X[0];//initialize local max with first element
            for (int i=1;i<local_n;i++){
                if (local_X[i] > local_max){
                    local_max = local_X[i];
                }
            }
            //same procudeure as avg to find global max
            double global_max;
            if(rank==0){
                global_max = local_max;
                double temp_max;
                for (int source =1; source < numtasks; source++){
                    MPI_Recv(&temp_max, 1, MPI_DOUBLE, source, 30, MPI_COMM_WORLD, MPI_STATUS_IGNORE
);
                    if (temp_max > global_max){
                        global_max = temp_max;
                    }
                }
                printf("Global Max = %lf\n", global_max);
                for (int dest = 1; dest < numtasks; dest++) {
                MPI_Send(&global_max, 1, MPI_DOUBLE, dest, 35, MPI_COMM_WORLD);
                }
            } else {
                MPI_Send(&local_max, 1, MPI_DOUBLE, 0, 30, MPI_COMM_WORLD);
                 MPI_Recv(&global_max, 1, MPI_DOUBLE, 0, 35, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            }
```

(γ) τη διασπορά (var) των στοιχείων του διανύσματος X:

Θέλουμε να υπολογίσουμε :

$$var = ((x0–\mu)\ ^2+(x1–\mu)^\ 2+(x2–\mu)\ ^2+…+(xn-1–\mu)^2) / n$$

Εξακολουθεί η ίδια τακτική . Έχουμε μία local_var (x0–μ) ^2+(x1–μ)^ 2+(x2–μ) ^2+…+(xn-1–μ)^2 για να υπολογίζει την τοπική διασπορά της κάθε διεργασίας. Χρησιμοποίησα την συναρτηση pow() από την βιβλιοθήκη math.h. Η διεργασία 0 παίρνει την total_var και την διαιρεί με το n και μας εμφανίζει το επιθυμητό αποτέλεσμα

```
            //(c)
            double local_var = 0.0;
            for (int i=0;i<local_n;i++){
                local_var += pow((local_X[i] - average) ,2);
            }

            if (rank ==0){
                double total_var = local_var;
                double temp_var;
                for (int source =1; source < numtasks; source++){
                    MPI_Recv(&temp_var, 1, MPI_DOUBLE, source, 40, MPI_COMM_WORLD, MPI_STATUS_IGNORE
);
                    total_var += temp_var;
                }
                double variance = total_var / n;
                printf("Variance = %lf\n", variance);
            } else {
                MPI_Send(&local_var, 1, MPI_DOUBLE, 0, 40, MPI_COMM_WORLD);
            }
```

(δ) ένα νέο διάνυσμα Δ (μήκους n στοιχείων δi | i=0…n-1), του οποίου κάθε στοιχείο δi θα ισούται με το τετράγωνο της διαφοράς του αντίστοιχου στοιχείου (xi) του διανύσματος X από τη μέγιστη τιμή m του διανύσματος:

$$\delta i = (xi - m)^2$$

Κάθε διεργασία υπολογίζει τα τοπικά της δi και τα αποθηκεύει στον local_delta δυναμικό πίνακα.Η διεργασία 0 δημιουργεί τον πίνακα delta και συγκεντρώνει τα local_delta από όλες τις διεργασίες. Ο offset ξεκινά από local_n(μετά τα στοιχεία της διεργασίας 0) και αυξάνεται κατά elemperproc για κάθε λήψη. Τέλος, η διεργασία 0 εμφανίζει όλα τα δi.

```
            //(d)
            double *local_delta = (double*)malloc(local_n*sizeof(double));
            for (int i=0;i<local_n;i++){
                local_delta[i] = pow((local_X[i] - global_max),2);//each process calculates its loca
l delta values
            }

            double *delta = NULL;
            if (rank ==0){
                delta = (double*)malloc(n*sizeof(double));


                for (int i=0;i<local_n;i++){
                    delta[i] = pow((local_X[i] - global_max),2) ;
                }

                int offset = local_n;
                for (int source =1; source < numtasks; source++){
                   MPI_Recv(&delta[offset], elemperproc, MPI_DOUBLE, source, 4, MPI_COMM_WORLD, MPI_
STATUS_IGNORE);
                    offset += elemperproc;
                }

                printf("\nDelta values:\n");
                for (int i=0; i<n; i++){
                    printf("Delta[%d] = %lf\n", i, delta[i]);
                }
            }else {
                    MPI_Send(local_delta, local_n, MPI_DOUBLE, 0, 4, MPI_COMM_WORLD);
            }
```

FREE ALLOCATED MEMORY & ΤΕΡΜΑΤΙΣΜΟΣ MPI

Θέλουμε να κάνουμε αποδέσμευση μνήμης.Η διεργασία 0 αποδεσμεύει τον πινακα X και delta και αποδεσμεύονται και οι local_X local_delta πινακες.

```
//free allocated memory
if (rank ==0){
free(X);
free(delta);
}
free(local_X);
free(local_delta);
```

MPI_Finalize() κλείνει με ασφάλεια το περιβάλλον MPI

```
5
4 MPI_Finalize(); //finalize MPI environment
3 return 0;
2
```

ΕΝΔΕΙΚΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ :

Κάνουμε compile το προγραμμα :

```
mpicc -o askisi Askisi.c -lm
```

Τρέχουμε το εκτελέσιμο προγραμμα με 4 διεργασίες:

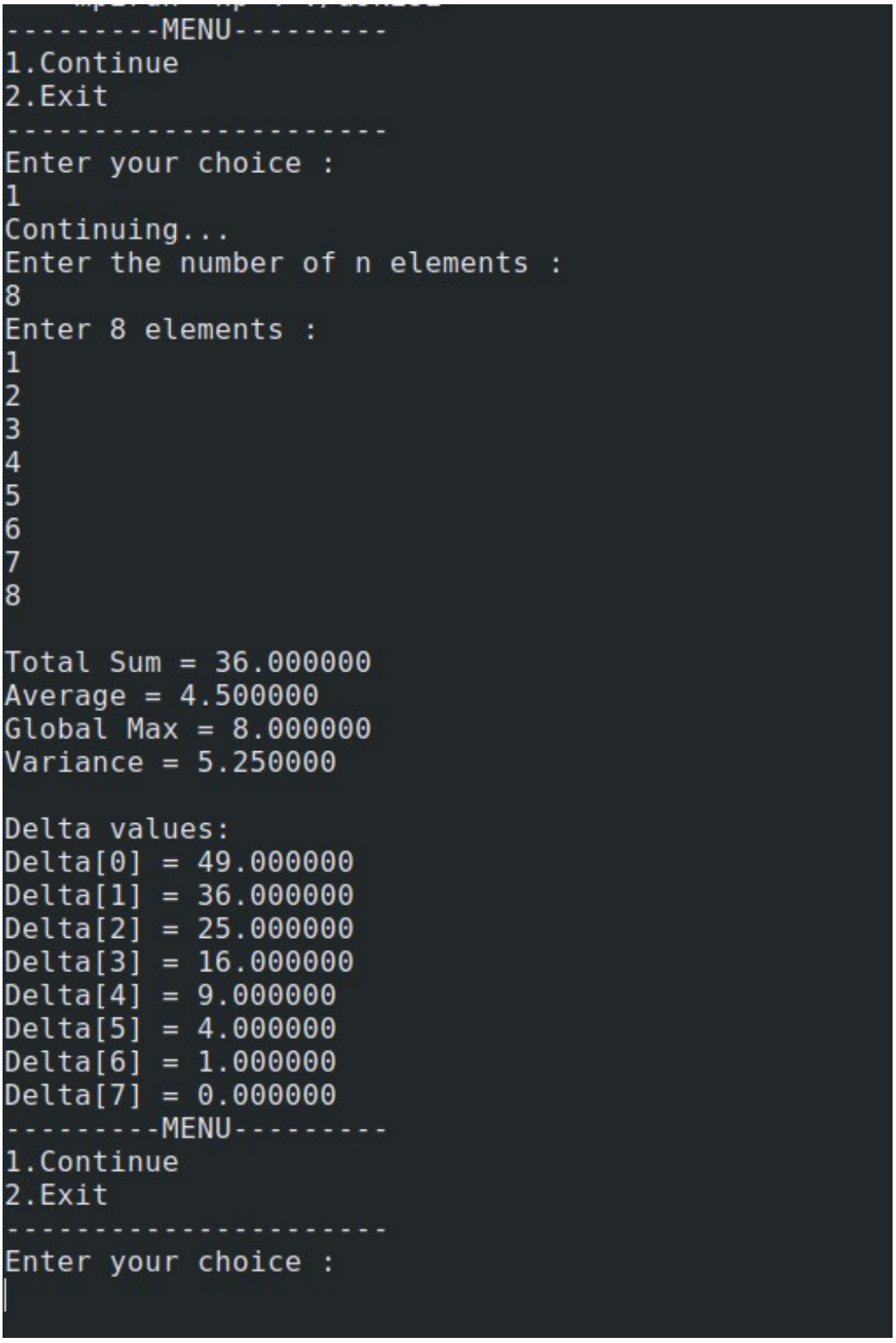
```
mpirun -np 4 ./askisi
```

1. Έλεγχος MENU

```
>>> mpirun -np 4 ./askisi
---------MENU---------
1.Continue
2.Exit
----------------------
Enter your choice :
2
Exiting program...
```

```
---------MENU---------
1.Continue
2.Exit
----------------------
Enter your choice :
3
Invalid choice, please try again.
---------MENU---------
1.Continue
2.Exit
----------------------
Enter your choice :
```

Πρώτη περίπτωση n%P = 0



Κάθε διεργασία θα πάρει 2 elements rank 0 :[1,2] rank 1:[3,4] rank 2:[5,6] rank 3:[7,8].
Το πρόγραμμα μας δίνει τα αναμενόμενα αποτελέσματα.

Δεύτερη περίπτωση n%P != 0

```
---------MENU---------
1.Continue
2.Exit
----------------------
Enter your choice :
1
Continuing...
Enter the number of n elements :
9
Enter 9 elements :
1
2
3
4
5
6
7
8
9

Total Sum = 45.000000
Average = 5.000000
Global Max = 9.000000
Variance = 6.666667

Delta values:
Delta[0] = 64.000000
Delta[1] = 49.000000
Delta[2] = 36.000000
Delta[3] = 25.000000
Delta[4] = 16.000000
Delta[5] = 9.000000
Delta[6] = 4.000000
Delta[7] = 1.000000
Delta[8] = 0.000000
---------MENU---------
1.Continue
2.Exit
----------------------
Enter your choice :
```

Κάθε διεργασία θα πάρει 2 elements εκτός απο την διεργασία 0 οπου θα πάρει 3.
rank 0 :[1,2,3] rank 1:[4,5] rank 2:[6,7] rank 3:[8,9].
Το πρόγραμμα μας δίνει τα αναμενόμενα αποτελέσματα.