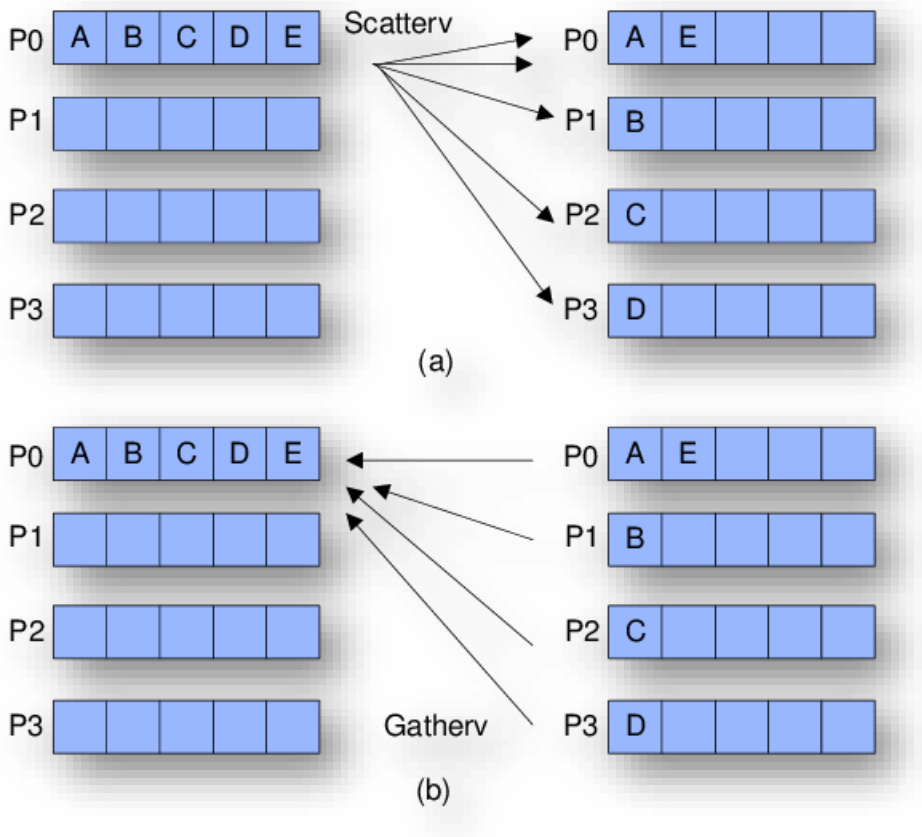
ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ : ΤΜΗΜΑ
ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΥΠΟΛΟΓΙΣΤΩΝ

ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΑΡΑΛΛΗΛΟ ΥΠΟΛΟΓΙΣΜΟ

Θέμα: Παράλληλος Υπολογισμός MPI – Υλοποίηση σε C



Εργαστήριο 2020-2021, Τμήμα: Ε2 (Δευτέρα 1-2)
Κος Μ. Ιορδανάκης

" MPI – Συναρτήσεις Συλλογικής Επικοινωνίας"

ΒΕΛΑΣΚΟ ΠΑΟΛΑ
ΑΜ: cs161020
Πρόγραμμα ΠΑΔΑ
Εξάμηνο 9ο

Εργασία 2
Ημ. Παράδοσης: 10 Ιαν. 2021

Πίνακας περιεχομένων

ΣΚΟΠΟΣ ΕΡΓΑΣΙΑΣ – ΕΙΣΑΓΩΓΗ

Η εκπόνηση της 2ης εργαστηριακής εργασίας έχει ως σκοπό να μας εμβαθύνει τις γνώσεις μας για τον παράλληλο υπολογισμό σε συνέχεια από την εργασία 1. Συγκεκριμένα ασχολούμαστε με το MPI (Message Passing Interface), υλοποιημένη στη γλώσσα προγραμματισμού C, και με τις συναρτήσεις συλλογικής επικοινωνίας.

Στη συλλογική επικοινωνία (collective) επικοινωνία μπορούν να εμπλέκονται και περισσότερες των δύο διεργασιών, σε αντίθεση με την point to point. Έτσι, επιτυγχάνουμε την αποστολή ενός μηνύματος σε πολλούς παραλήπτες ή την παραλαβή ενός μηνύματος από πολλούς αποστολείς, με μία μόνον κλήση συνάρτησης συλλογικής επικοινωνίας. (1)

Γνωστές συναρτήσεις και συναρτήσεις που χρησιμοποιήθηκαν στην εργασία.

- **MPI_Bcast**

```
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
```

- **MPI_Scatterv**

```
int MPI_Scatterv (void *sendbuf; int *sendcnts; int *displs; MPI_Datatype sendtype; void *recvbuf; int recvcnt; MPI_Datatype recvtype; int root; MPI_Comm comm;)
```

- **MPI_Gatherv**

```
int MPI_Gatherv (void *sendbuf; int sendcnt; MPI_Datatype sendtype; void *recvbuf; int *recvcnts; int *displs; MPI_Datatype recvtype; int root; MPI_Comm comm;)
```

- **MPI_Reduce**

```
int MPI_Reduce (void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
```

ΑΝΑΠΤΥΞΗ ΚΩΔΙΚΑ

Παρατηρήσεις

- Το πρόγραμμα υλοποιήθηκε με επαναληπτική χρήση μενού
- Το πρόγραμμα **λειτουργεί** για n **μη πολλαπλάσιο** του p (`Scatterv`, `Gatherv`) , ωστόσο παρατηρήθηκε ότι για n < p δε γίνονται σωστά οι υπολογισμοί.
- Δεν υλοποιήθηκε το ερώτημα (ε) λόγω δυσκολίας

Κώδικας

Ο σχολιασμένος κώδικας βρίσκεται στο αρχείο 161020PV.c

Καθώς και στο παράρτημα στο τέλος του εγγράφου (μετάβαση)

ΤΕΚΜΗΡΙΩΣΗ ΚΩΔΙΚΑ

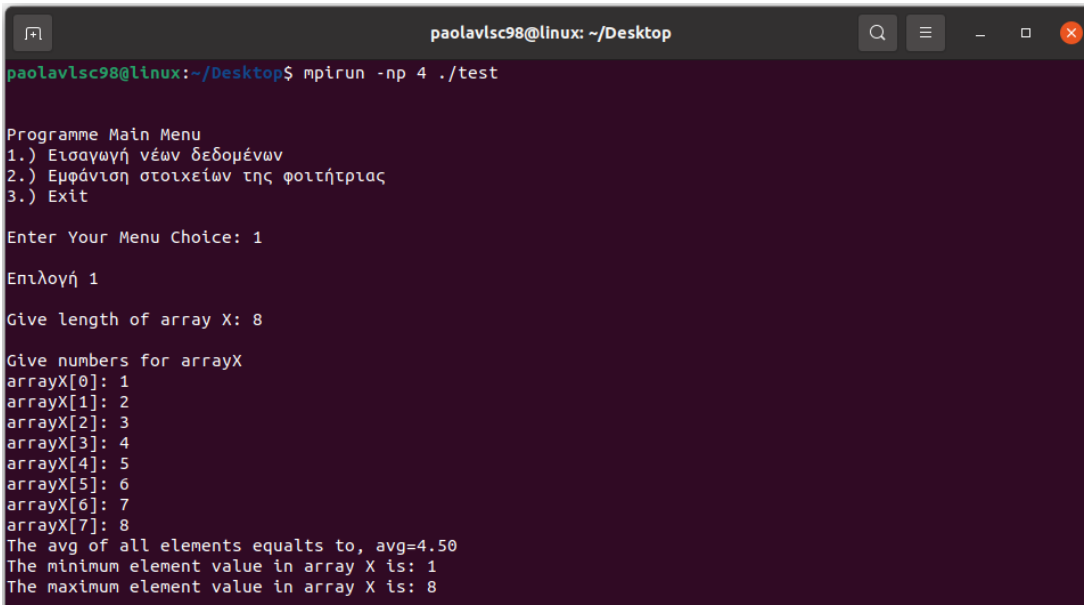
Το πρόγραμμά μας αρχικά ξεκινάει από μια main menu όπου σε αυτή γίνεται η διεπαφή επικοινωνίας με το χρήστη. Ζητά να επιλέξει 1) Εισαγωγή νέων δεδομένων. 2) Προβολή στοιχείων της φοιτήτριας 3) Τερματισμός προγράμματος.

```
paolavlsc98@linux:~/Desktop$ mpirun -np 4 ./test


Programme Main Menu
1.) Εισαγωγή νέων δεδομένων
2.) Εμφάνιση στοιχείων της φοιτήτριας
3.) Exit

Enter Your Menu Choice:
```

```
Enter Your Menu Choice: 2

Επιλογή 2

cs161020
Paola Velasco
Semester 9th
```

```
Enter Your Menu Choice: 3

Επιλογή 3
Exiting Program...
paolavlsc98@linux:~/Desktop$
```

Αν ο χρήστης επιλέξει την επιλογή 1 τότε γίνεται η έναρξη του κύριου προγράμματός μας. Αρχικά ζητά από το χρήστη να εισάγει το μήκος του πίνακα X και ύστερα να τον αρχικοποιήσει με ακέραιες τιμές που θα δίνονται από τον ίδιον.



Στη συνέχεια, υπολογίζεται πόσα νούμερα θα λάβει η κάθε διεργασία και αποθηκεύονται στον πίνακα `sendCounts[]`, ενώ γίνεται και ο υπολογισμός και η αρχικοποίηση του πίνακα `displacements[]` (πίνακας που δείχνει το offset του μπλοκ στοιχείων που θα σταλεί σε κάθε διεργασία από το σημείο εκκίνησης).

Κώδικας: Υπολογισμός sendCounts[]

```
1.  // dividing how many numbers should be given to each process & save it to array
    sendCounts[]
2.  int mod;
3.  mod = arrayX_size % numtasks;
4.  for (i = 0; i < numtasks; i++)
5.  {
6.
7.      if (i < mod)
8.      {
9.          *(sendCounts + i) = arrayX_size / numtasks + 1;
10.     }
11.     else
12.     {
13.         *(sendCounts + i) = arrayX_size / numtasks;
14.     }
15. }
```

Κώδικας: Υπολογισμός displacements[]

```
1.  // Αρχικοποίηση του πίνακα displacements
2.  int index = 0;
3.  for (i = 0; i < numtasks; i++)
4.  {
5.
6.      if (i == 0)
7.      {
8.          displacements[i] = index;
9.      }
10.     else
```

```
    {
        index += *(sendCounts + (i - 1));
        displacements[i] = index;
    }
}
```

Αφού αρχικοποιηθούν οι τοπικοί πίνακες localArrayX, ακολουθείται ο παράλληλος υπολογισμός της μέσης τιμής και ο υπολογισμός της min & max. Για τον υπολογισμό της μέσης τιμής η κάθε διεργασία αθροίζει τα στοιχεία του πίνακα localArrayX της και τη στέλνει πίσω στην ρίζα με τη συνάρτηση MPI_Reduce σε συνδυασμό τη MPI_SUM. Το τελικό αποτέλεσμα αποθηκεύεται στη μεταβλητή της ρίζας tot_sum και η ρίζα στη συνέχεια διαιρεί αυτή την τιμή με το μέγεθος του πίνακα X. Για τον υπολογισμό της μικρότερης τιμής, γίνεται η αντίστοιχη διαδικασία. Κάθε διεργασία, ψάχνει τη μικρότερη τιμή της μεταξύ των στοιχείων του τοπικού τους πίνακα localArrayX και τη στέλνει με MPI_Reduce στην ρίζα. Σε συνδυασμό της MPI_MIN γίνεται και η αυτόματη εύρεση μεταξύ των στοιχείων που στέλνουν οι διεργασίες της μικρότερης τιμής. Με αντίστοιχο τρόπο υλοποιείται και η εύρεση της μέγιστης τιμής.

Κώδικας: Υπολογισμός μέσης τιμής

```
sum = 0;

for (i = 0; i < sendCounts[my_rank]; i++)
{
    sum += localArrayX[i];
}

MPI_Reduce(&sum, &tot_sum, 1, MPI_FLOAT, MPI_SUM, root, MPI_COMM_WORLD);
if (my_rank == root)
{
    avg = tot_sum / (float)arrayX_size;
    printf("The avg of all elements equalts to, avg=%.2f\n", avg);
}
```

Κώδικας: Υπολογισμός min & max

```
local_min = localArrayX[0];
local_max = localArrayX[0];

for (i = 0; i < sendCounts[my_rank]; i++)
{
    if (localArrayX[i] < local_min)
    {
        local_min = localArrayX[i];
    }

    if (localArrayX[i] > local_max)
    {
        local_max = localArrayX[i];
    }
}

MPI_Reduce(&local_min, &min, 1, MPI_INT, MPI_MIN, root, MPI_COMM_WORLD);
MPI_Reduce(&local_max, &max, 1, MPI_INT, MPI_MAX, root, MPI_COMM_WORLD);

if (my_rank == root)
```

```
21. {
22.     printf("The minimum element value in array X is: %d\n", min);
23.     printf("The maximum element value in array X is: %d\n", max);
24. }
```

```
Give length of array X: 8

Give numbers for arrayX
arrayX[0]: 1
arrayX[1]: 2
arrayX[2]: 3
arrayX[3]: 4
arrayX[4]: 5
arrayX[5]: 6
arrayX[6]: 7
arrayX[7]: 8
The avg of all elements equals to, avg=4.50
The minimum element value in array X is: 1
The maximum element value in array X is: 8
```

Στη συνέχεια του προγράμματος, εμφανίζεται το μενού για τα υποερωτήματα που μας δίνονται στην άσκηση.

```
Programme Main Menu
1.) Πόσα στοιχεία του X έχουν μικρότερη και πόσα μεγαλύτερη τιμή από τη μέση τιμή;
2.) Διασπορά των στοιχείων του διανύσματος X
3.) Υπολόγισε τη ποσοστιάια σχέση των στοιχείων του X με τη διαφορά μεγίστου-ελαχίστου
4.) Ποιά είναι η μεγαλύτερη τιμη του διανύσματος Δ και για ποιό στοιχείο
5.) Go back to main menu
6.) Exit

Enter Your Menu Choice:
```

Επιλογή 1: (βλέπε εδώ)

Επιλογή 2: (βλέπε εδώ)

Επιλογή 3: (βλέπε εδώ)

Επιλογή 4: (βλέπε εδώ)

Επιλογή 5: Επιστρέφει το χρήστη στο αρχικό μενού

Επιλογή 6: Τερματισμός προγράμματος

```
Enter Your Menu Choice: 5

Choice 5


Programme Main Menu
1.) Εισαγωγή νέων δεδομένων
2.) Εμφάνιση στοιχείων της φοιτήτριας
3.) Exit

Enter Your Menu Choice: 6

This is not a valid Menu Option! Please Select Another
```

1. Πόσα στοιχεία του διανύσματος X έχουν μικρότερη και πόσα μεγαλύτερη τιμή από τη μέση τιμή (m) αυτού.

Κώδικας

```
1.  void greaterLess(int countGreater, int countLess, int sendCounts[], int my_rank,
     int localArrayX[], float avg, int *totalCountGreater, int root, int *totalCount
    Less)
2.  {
3.
4.      int i = 0;
5.
6.      countGreater = 0;
7.      countLess = 0;
8.      for (i = 0; i < sendCounts[my_rank]; i++)
9.      {
10.         if (localArrayX[i] > avg)
11.             countGreater++;
12.         else if (localArrayX[i] < avg)
13.             countLess++;
14.         else
15.         {
16.             printf("localArray[%d] = %d is equal with avg = %.2f\n", i, localArr
    ayX[i], avg);
17.         }
18.     }
19.
20.
21.     MPI_Reduce(&countGreater, totalCountGreater, 1, MPI_INT, MPI_SUM, root, MPI_
    COMM_WORLD);
22.     MPI_Reduce(&countLess, totalCountLess, 1, MPI_INT, MPI_SUM, root, MPI_COMM_W
    ORLD);
23. }
```

Επεξήγηση

Έχοντας ήδη λάβει από την ρίζα την τιμή της μέσης τιμής, η κάθε διεργασία υπολογίζει στη δικιά τους μνήμη και δεδομένα, πόσες τιμές έχουν μεγαλύτερη τιμή (`countGreater`) και πόσες μικρότερες (`countless`) από τη μέση τιμή. Στη συνέχεια στέλνει τα αποτελέσματα πίσω στην ρίζα με την MPI_Reduce, όπου γίνονται αυτόματα τα αθροίσματα. Τα τελικά αποτελέσματα αποθηκεύονται στις μεταβλητές της ρίζας `totalCountGreater` και `totalCountLess`.

Αποτελέσματα για διαφορετικά τρεξίματα

```
mpiexec -n 4 ./test
```

```
Give length of array X: 8

Give numbers for arrayX
arrayX[0]: 1
arrayX[1]: 2
arrayX[2]: 3
arrayX[3]: 4
arrayX[4]: 5
arrayX[5]: 6
arrayX[6]: 7
arrayX[7]: 8
The avg of all elements equals to, avg=4.50
The minimum element value in array X is: 1
The maximum element value in array X is: 8


Programme Main Menu
1.) Πόσα στοιχεία του X έχουν μικρότερη και πόσα μεγαλύτερη τιμή από τη μέση τιμή;
2.) Διασπορά των στοιχείων του διανύσματος X
3.) Υπολόγισε τη ποσοστιάια σχέση των στοιχείων του X με τη διαφορά μεγίστου-ελαχίστου
4.) Ποιά είναι η μεγαλύτερη τιμη του διανύσματος Δ και για ποιό στοιχείο
5.) Go back to main menu
6.) Exit

Enter Your Menu Choice: 1

Choice 1
Total elements that are greater than the average value: 4
Total elements that are less than the average value: 4
```

```
mpiexec -n 3 ./test
```

```
paolavlsc98@linux: ~/Desktop

Give numbers for arrayX
arrayX[0]: 6
arrayX[1]: 8
arrayX[2]: 5
arrayX[3]: 2
arrayX[4]: 10
arrayX[5]: 7
arrayX[6]: 77
arrayX[7]: 9
arrayX[8]: -6
arrayX[9]: 87
arrayX[10]: 104
arrayX[11]: 89
arrayX[12]: 45
arrayX[13]: -64
The avg of all elements equals to, avg=27.07
The minimum element value in array X is: -64
The maximum element value in array X is: 104


Programme Main Menu
1.) Πόσα στοιχεία του X έχουν μικρότερη και πόσα μεγαλύτερη τιμή από τη μέση τιμή;
2.) Διασπορά των στοιχείων του διανύσματος X
3.) Υπολόγισε τη ποσοστιάια σχέση των στοιχείων του X με τη διαφορά μεγίστου-ελαχίστου
4.) Ποιά είναι η μεγαλύτερη τιμη του διανύσματος Δ και για ποιό στοιχείο
5.) Go back to main menu
6.) Exit

Enter Your Menu Choice: 1

Choice 1
Total elements that are greater than the average value: 5
Total elements that are less than the average value: 9


Programme Main Menu
1.) Πόσα στοιχεία του X έχουν μικρότερη και πόσα μεγαλύτερη τιμή από τη μέση τιμή;
2.) Διασπορά των στοιχείων του διανύσματος X
3.) Υπολόγισε τη ποσοστιάια σχέση των στοιχείων του X με τη διαφορά μεγίστου-ελαχίστου
4.) Ποιά είναι η μεγαλύτερη τιμη του διανύσματος Δ και για ποιό στοιχείο
5.) Go back to main menu
6.) Exit

Enter Your Menu Choice:
```

2. Τη διασπορά των στοιχείων του διανύσματος X

Κώδικας

```
1.  void var(float numerator, int sendCounts[], int my_rank, int localArrayX[], floa
    t avg, float *totalNumerators, int root, int arrayX_size)
2.  {
3.      int i;
4.      numerator = 0.0;
5.
6.      for (i = 0; i < sendCounts[my_rank]; i++)
7.      {
8.          numerator += (localArrayX[i] - avg) * (localArrayX[i] - avg);
9.      }
10.
11.     MPI_Reduce(&numerator, totalNumerators, 1, MPI_FLOAT, MPI_SUM, root, MPI_COM
    M_WORLD);
12. }
```

Επεξήγηση

Κάθε διεργασία υπολογίζει το άθροισμα του τετραγώνου $((x_i - m)^2$ μεταξύ των στοιχείων που έχει στον τοπικό τους πίνακα localArrayX. Αφού τις υπολογίσει, επιστρέφει το αποτέλεσμα στην ρίζα με την MPI_Reduce σε συνδυασμό MPI_SUM. Στη συνέχεια , έχοντας ήδη λάβει η ρίζα στη μεταβλητή totalNumerators, η ρίζα τη διαιρεί με το arrayX_size και εκτυπώνει το τελικό αποτέλεσμα της διασποράς.

Αποτελέσματα για διαφορετικά τρεξίματα

```
mpiexec -n 4 ./test
```

```
Give length of array X: 8

Give numbers for arrayX
arrayX[0]: 1
arrayX[1]: 2
arrayX[2]: 3
arrayX[3]: 4
arrayX[4]: 5
arrayX[5]: 6
arrayX[6]: 7
arrayX[7]: 8
```

```
Enter Your Menu Choice: 2

Choice 2
Total var of elements: var = 5.25
```

```
mpiexec -n 3 ./test
```

```
Give length of array X: 14

Give numbers for arrayX
arrayX[0]: 6
arrayX[1]: 8
arrayX[2]: 5
arrayX[3]: 2
arrayX[4]: 10
arrayX[5]: 7
arrayX[6]: 77
arrayX[7]: 9
arrayX[8]: -6
arrayX[9]: 87
arrayX[10]: 104
arrayX[11]: 89
arrayX[12]: 45
arrayX[13]: -64
```

```
Enter Your Menu Choice: 2

Choice 2
Total var of elements: var = 2035.07
```

3. Ένα νέο διάνυσμα Δ όπου κάθε στοιχείο δi θα ισούται με την ποσοστιαία σχέση του αντίστοιχου στοιχείου (xi) του διανύσματος Χ με τη διαφορά μεγίστου-ελαχίστου των τιμών όλου του διανύσματος Χ

Κώδικας

```
1.  localArrayD = (double *)malloc(sizeof(double) * sendCounts[my_rank]);
2.  if (localArrayD == NULL)
3.  {
4.      printf("Error: Not available memory\n");
5.      exit(EXIT_FAILURE);
6.  }
7.
8.  for (i = 0; i < sendCounts[my_rank]; i++)
9.  {
10.     localArrayD[i] = ((localArrayX[i] - min) / (double)(max - min)) * 100.0;
11. }
12.
13. if (my_rank == root)
14. {
15.     arrayD = (double *)malloc(sizeof(double) * arrayX_size);
16.     if (arrayD == NULL)
17.     {
18.         printf("Error: Not available memory\n");
19.         exit(EXIT_FAILURE);
20.     }
21. }
22.
23. MPI_Gatherv(localArrayD, sendCounts[my_rank], MPI_DOUBLE, arrayD, sendCounts, di
    splacements, MPI_DOUBLE, root, MPI_COMM_WORLD);
24.
25. if (my_rank == root)
26. {
27.     for (i = 0; i < arrayX_size; i++)
28.     {
29.         printf("arrayD[%d] = %.15f\n", i, arrayD[i]);
```

```
30.     }
31. }
```

Επεξήγηση

Έχοντας ήδη λάβει την πληροφορία min, max από την ρίζα, η κάθε διεργασία δημιουργεί τοπικά τον δικό τους localArrayD όπου θα αποθηκεύονται οι υπολογισμοί $\delta_i = ((x_i - x_{min}) / (x_{max} - x_{min})) * 100$ των κάθε στοιχείων localArrayX τους.

Αφού τις υπολογίσει, η κάθε διεργασία στέλνει στην ρίζα με MPI_Gather

Αποτελέσματα για διαφορετικά τρεξίματα

```
mpiexec -n 4 ./test
```

```
Give numbers for arrayX
arrayX[0]: 5
arrayX[1]: 6
arrayX[2]: 7
arrayX[3]: 0
arrayX[4]: -5
arrayX[5]: 97
arrayX[6]: 4
arrayX[7]: 3
```

```
Enter Your Menu Choice: 3

Choice 3
arrayD[0] = 9.803921568627452
arrayD[1] = 10.784313725490197
arrayD[2] = 11.764705882352940
arrayD[3] = 4.901960784313726
arrayD[4] = 0.000000000000000
arrayD[5] = 100.000000000000000
arrayD[6] = 8.823529411764707
arrayD[7] = 7.843137254901960
```

```
mpiexec -n 3 ./test
```

```
Give length of array X: 14

Give numbers for arrayX
arrayX[0]: 6
arrayX[1]: 8
arrayX[2]: 5
arrayX[3]: 2
arrayX[4]: 10
arrayX[5]: 7
arrayX[6]: 77
arrayX[7]: 9
arrayX[8]: -6
arrayX[9]: 87
arrayX[10]: 104
arrayX[11]: 89
arrayX[12]: 45
arrayX[13]: -64
```

```
Enter Your Menu Choice: 3

Choice 3
arrayD[0] = 41.666666666666671
arrayD[1] = 42.857142857142854
arrayD[2] = 41.071428571428569
arrayD[3] = 39.285714285714285
arrayD[4] = 44.047619047619044
arrayD[5] = 42.261904761904759
arrayD[6] = 83.928571428571431
arrayD[7] = 43.452380952380956
arrayD[8] = 34.523809523809526
arrayD[9] = 89.880952380952380
arrayD[10] = 100.000000000000000
arrayD[11] = 91.071428571428569
arrayD[12] = 64.880952380952380
arrayD[13] = 0.000000000000000
```

4. Ποια είναι η μεγαλύτερη τιμή του διανύσματος Δ και για ποιο στοιχείο xi συγκεκριμένα παρατηρείται (θα πρέπει να τυπώνεται η θέση i του στοιχείου στο διάνυσμα, η τιμή του στοιχείου και το δi του).

Κώδικας

```
1.  if (localArrayD == NULL)
2.  {
3.      if (my_rank == 0)
4.          printf("Question 3 should be first executed\n");
5.  }
6.  else
7.  {
8.
9.      inmin.value = localArrayD[0];
10.     inmax.value = localArrayD[0];
11.     inmin.index = 0;
12.     inmax.index = 0;
13.
14.     for (i = 1; i < sendCounts[my_rank]; i++)
15.     {
16.         if (inmin.value > localArrayD[i])
17.         {
18.             inmin.value = localArrayD[i];
19.             inmin.index = i;
20.         }
21.
22.         if (inmax.value < localArrayD[i])
23.         {
24.             inmax.value = localArrayD[i];
25.             inmax.index = i;
26.         }
27.     }
28.
29.     inmin.index = my_rank * sendCounts[my_rank] + inmin.index;
30.     inmax.index = my_rank * sendCounts[my_rank] + inmax.index;
31.
32.     MPI_Reduce(&inmin, &outmin, 1, MPI_FLOAT_INT, MPI_MINLOC, root, MPI_COMM_WOR
    LD);
33.     MPI_Reduce(&inmax, &outmax, 1, MPI_FLOAT_INT, MPI_MAXLOC, root, MPI_COMM_WOR
    LD);
34.
35.     if (my_rank == root)
36.     {
37.         minval = outmin.value;
38.         minrank = outmin.index / sendCounts[my_rank];
39.         minindex = outmin.index % sendCounts[my_rank];
40.
41.         int globalindexmin = minrank * sendCounts[my_rank] + minindex;
42.
43.         maxval = outmax.value;
44.         maxrank = outmax.index / sendCounts[my_rank];
45.         maxindex = outmax.index % sendCounts[my_rank];
46.
47.         int globalindexmax = maxrank * sendCounts[my_rank] + maxindex;
48.
49.         printf("The min val is %f\n", minval);
50.         printf("The rank that has the min val is rank = %d\n", minrank);
```

```
51.         printf("The index is at %d\n", minindex);
52.         printf("The global index is at %d\n", globalindexmin);
53.
54.         printf("The max val is %f\n", maxval);
55.         printf("The rank that has the max val is rank = %d\n", maxrank);
56.         printf("The index is at %d\n", maxindex);
57.         printf("The global index is at %d\n", globalindexmax);
58.     }
59. }
```

Επεξήγηση

Έχοντας ορίσει τη struct info και τις global μεταβλητές `inmax`, `inmin`, `outmax`, `outman` η κάθε διεργασία υπολογίζει τη global θέση τιμή της και όχι τον τοπικό τους δείκτη για τα ζητούμενα της εργασίας.

Αποτελέσματα για διαφορετικά τρεξίματα

```
mpiexec -n 4 ./test
```

```
Give numbers for arrayX
arrayX[0]: 5
arrayX[1]: 6
arrayX[2]: 7
arrayX[3]: 0
arrayX[4]: -5
arrayX[5]: 97
arrayX[6]: 4
arrayX[7]: 3
```

```
Enter Your Menu Choice: 3

Choice 3
arrayD[0] = 9.803921568627452
arrayD[1] = 10.784313725490197
arrayD[2] = 11.764705882352940
arrayD[3] = 4.901960784313726
arrayD[4] = 0.000000000000000
arrayD[5] = 100.000000000000000
arrayD[6] = 8.823529411764707
arrayD[7] = 7.843137254901960
```

```
Enter Your Menu Choice: 4

Choice 4
The min val is 0.000000
The rank that has the min val is rank = 0
The index is at 0
The global index is at 0
The max val is 100.000000
The rank that has the max val is rank = 3
The index is at 1
The global index is at 7
```

```
mpiexec -n 3 ./test
```

```
Give length of array X: 6

Give numbers for arrayX
arrayX[0]: 5
arrayX[1]: 98
arrayX[2]: 4
arrayX[3]: 22
arrayX[4]: 3
arrayX[5]: 61
```

```
Enter Your Menu Choice: 3

Choice 3
arrayD[0] = 2.105263157894737
arrayD[1] = 100.000000000000000
arrayD[2] = 1.052631578947368
arrayD[3] = 20.000000000000000
arrayD[4] = 0.000000000000000
arrayD[5] = 61.052631578947370
```

```
Choice 4
The min val is 0.000000
The rank that has the min val is rank = 1
The index is at 0
The global index is at 2
The max val is 100.000000
The rank that has the max val is rank = 0
The index is at 1
The global index is at 1
```

ΣΥΜΠΕΡΑΣΜΑΤΑ

Έχοντας πλέον διεκπεραιώσει την εργασία 2 παρατηρήσαμε τις αλλαγές που έχουν οι συναρτήσεις συλλογικής επικοινωνίας με τις συναρτήσεις των point-to-point επικοινωνία.

Θεωρητικά, για την πραγματοποίηση της επικοινωνίας πολλών διεργασιών, μπορούν να χρησιμοποιηθούν οι συναρτήσεις send και receive που χρησιμοποιούνται στην από κόμβο σε κόμβο επικοινωνία. Με άλλα λόγια, μπορούν να χρησιμοποιηθούν «πολλές από κόμβο σε κόμβο επικοινωνίες». Όμως, έτσι αυξάνεται η πολυπλοκότητα του κώδικα. Αν π.χ., πρέπει μία διεργασία να στείλει το ίδιο μήνυμα σε 100 άλλες διεργασίες, θα χρειαστούν 100 κλήσεις από κόμβο σε κόμβο επικοινωνίας. Θα ήταν πολύ πιο εύκολο αν ήταν δυνατόν η αποστολή του μηνύματος στις 100 διεργασίες να γίνει με την κλήση μιας μόνο συνάρτησης. (1)

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. **Βασίλειος Μάμαλης, Γραμματή Πάντζιου, Αλέξανδρος Τομαράς.** *Εισαγωγή στον Παράλληλο Υπολογισμό (Πρότυπα, Αλγόριθμοι, Προγραμματισμός) .* Αθήνα : Νέων Τεχνολογιών, 2013.

ΠΑΡΑΡΤΗΜΑ

Εκφώνηση

ΕΡΓΑΣΤΗΡΙΟ ΜΑΘΗΜΑΤΟΣ «ΕΙΣΑΓΩΓΗ ΣΤΟΝ ΠΑΡΑΛΛΗΛΟ ΥΠΟΛΟΓΙΣΜΟ» ΑΣΚΗΣΗ-ΙΙ 2020-2021 (10%)

Σας ζητείται να φτιάξετε ένα MPI πρόγραμμα (σε γλώσσα C), το οποίο δοθέντος ενός διανύσματος X (μήκους n στοιχείων x_i | i=0…n-1), **να υπολογίζει παράλληλα** σε περιβάλλον 'p' επεξεργαστών και να τυπώνει στην οθόνη (ως έξοδο) τα ακόλουθα (όπου *m*, x_{min}, x_{max} είναι η μέση τιμή, το ελάχιστο και το μέγιστο στοιχείο του διανύσματος X αντίστοιχα):

(α) Πόσα στοιχεία του διανύσματος X έχουν μικρότερη και πόσα μεγαλύτερη τιμή από τη μέση τιμή (*m*) αυτού.

(β) Τη διασπορά των στοιχείων του διανύσματος X:

$$var = ((x_0 - m)^2 + (x_1 - m)^2 + (x_2 - m)^2 + \ldots + (x_{n-1} - m)^2) / n$$

(γ) Ένα νέο διάνυσμα Δ όπου κάθε στοιχείο δ_i θα ισούται με την ποσοστιαία σχέση του αντίστοιχου στοιχείου (x_i) του διανύσματος X με τη διαφορά μεγίστου-ελαχίστου των τιμών όλου του διανύσματος X:

$$\delta_i = ((x_i - x_{min}) / (x_{max} - x_{min})) * 100$$

(δ) Ποια είναι η μεγαλύτερη τιμή του διανύσματος Δ και για ποιο στοιχείο x_i συγκεκριμένα παρατηρείται (θα πρέπει να τυπώνεται η θέση *i* του στοιχείου στο διάνυσμα, η τιμή του στοιχείου και το δ_i του).

(ε) Το διάνυσμα των προθεμάτων (prefix sums) των στοιχείων του διανύσματος X.

Προσπαθήστε να χρησιμοποιήσετε στην υλοποίησή σας κατά το δυνατόν **μόνο συναρτήσεις συλλογικής επικοινωνίας**.

Το σύνολο του απαιτούμενου υπολογιστικού φόρτου θα πρέπει να ισοκατανεμηθεί στους 'p' επεξεργαστές του παράλληλου περιβάλλοντός. Επίσης, κάθε επεξεργαστής θα πρέπει να λαμβάνει (κατέχει) στην τοπική του μνήμη μόνο τα δεδομένα εισόδου που χρησιμοποιεί για τοπικούς (δικούς του) υπολογισμούς.

Θεωρείστε αρχικά ότι το 'n' είναι ακέραιο πολλαπλάσιο του 'p'. Στη συνέχεια, προσπαθήστε να επεκτείνετε το πρόγραμμά σας έτσι ώστε να συμπεριφέρεται σωστά για οποιονδήποτε συνδυασμό τιμών 'n' και 'p' (με χρήση των συναρτήσεων Scatterv/Gatherv - μελετήστε σχετικά το παράδειγμα «mpi_scatterv.doc»).

Σχετικά με τα ζητούμενα του υποερωτήματος (δ) (και ειδικότερα για τον υπολογισμό της θέσης) μελετήστε το παράδειγμα «mpi_groups_plus.doc» (τελ. σελίδα για τη χρήση της MPI_Reduce με τελεστές maxloc/minloc).

Σχετικά με τα ζητούμενα του υποερωτήματος (ε) ζητείται να το υλοποιήσετε μόνο για την περίπτωση n=p (αναζητώντας στο διαδίκτυο σχετική συνάρτηση συλλογικής επικοινωνίας που μπορεί να σας βοηθήσει σε αυτό. Για την περίπτωση n>p ζητείται να περιγράψετε μόνο με λόγια (δίνοντας με σαφήνεια τα απαιτούμενα βήματα) πως θα μπορούσε να υλοποιηθεί.

Το πρόγραμμά σας θα πρέπει να δουλεύει επαναληπτικά με menu επιλογών.

Τρόπος και Ημερομηνία Παράδοσης:

Η Άσκηση θα πρέπει να παραδοθεί ηλεκτρονικά (μέσω της πλατφόρμας του Eclass) μέχρι και την **Τετάρτη 23/12/2020.**

Παραδοτέα: Ο κώδικας σχολιασμένος, τεκμηρίωση, και ενδεικτικά τρεξίματα.

Ανάπτυξη κώδικα

```c
/*********************************************************
 *                      ΠΑΟΛΑ ΒΕΛΑΣΚΟ
 *                   cs161020, 9ο εξάμηνο
 * *******************************************************
 *              Πανεπιστήμιο Δυτικής Αττικής
 *       Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών
 * -------------------------------------------------------
 *            Εισαγωγή στον Παράλληλο Υπολογισμό
 *            Εργαστήριο Τμήμα: Ε2 (Δευτέρα 1-2)
 *        Καθηγητές: κος Β. Μάμαλης, κος Μ. Ιορδανάκης
 * -------------------------------------------------------
 *        ΕΡΓΑΣΙΑ 2 - MPI in C (Συλλογική Επικοινωνία)
 *********************************************************/

#include <stdio.h>
#include "mpi.h"
#include <stdlib.h>

int main_menu();
void initArrayX(int arrayX[], int arrayX_size);
int menu();
void greaterLess(int countGreater, int countLess, int sendCounts[], int my_rank,
 int localArrayX[], float avg, int *totalCountGreater, int root, int *totalCount
Less);
void var(float numerator, int sendCounts[], int my_rank, int localArrayX[], floa
t avg, float *totalNumerators, int root, int arrayX_size);

void freeSpace(double arrayD[], int arrayX[], double localArrayD[], int localArr
ayX[], int sendCounts[], int displacements[], int my_rank);

struct info
{
    float value;
    int index;
};

/*global variables in, out type struct info*/
struct info inmin;
struct info outmin;
struct info inmax;
struct info outmax;

int main(int argc, char *argv[])
{
    int numtasks; // Αριθμός των διεργασιών
    int my_rank;  // Η ταυτότητα της διεργασίας
    int root = 0; // Η ρίζα

    int i;
    int rc;

    int choice_mainmenu;
    int choice;

    int *arrayX;     // Αρχικός πίνακας X
    int arrayX_size; // μήκος του αρχικού πίνακα X

```

```
54.     int *localArrayX;   // Τοπικός πίνακας που θα έχει στοιχεία όσα του ανατεθού
    ν
55.     int *sendCounts;    // Πίνακας που περιέχει το μήκος των πινάκων κάθε διεργα
    σίας
56.     int *displacements; // Πίνακας offset
57.
58.     float sum;
59.     float tot_sum;
60.     float avg; // Final average result
61.
62.     int local_min; // Ελάχιστη τιμή στον τοπικό πίνακα κάθε διεργασίας
63.     int local_max; // Μέγιστη τιμή στον τοπικό πίνακα κάθε διεργασίας
64.     int min;       // Ελάχιστη τιμή στον πίνακα X
65.     int max;       // Μέγιστη τιμή στον πίνακα X
66.
67.     int countGreater;      // a counter that keeps records of how many elements
    are greater than the average's value in each process
68.     int countLess;         // a counter that keeps records of how many elements
    are less than the average's value in each process
69.     int totalCountGreater; // Τελικό αποτέλεσμα για πόσες τιμές είναι μεγελύτερε
    ς από τη μέση τιμή
70.     int totalCountLess;    // Τελικό αποτέλεσμα για πόσες τιμές είναι μικρότερες
     από τη μέση τιμή
71.
72.     float numerator;       // Υπολογισμός αριθμητή της κάθε διεργάσιας για τη δι
    ασπορά
73.     float totalNumerators; // Συνολικός αριθμητής για τη διασπορά
74.     float totalVar;        // Διασπορά του πίνακα X
75.
76.     double *arrayD;      // Πίνακας arrayD
77.     double *localArrayD; // Τοπικός πίνακας arrayD
78.
79.     int minrank, minindex;
80.     float minval;
81.
82.     int maxrank, maxindex;
83.     float maxval;
84.
85.     /***********************************************ΠΡΟΕΤΟΙΜΑΣΙΑ ΠΡΟΓΡΑΜΜΑΤΟΣ**
    *****************************************/
86.
87.     // start the parallelism
88.     rc = MPI_Init(&argc, &argv);
89.     if (rc != 0)
90.     {
91.         printf("MPI initialization error\n");
92.         MPI_Abort(MPI_COMM_WORLD, rc);
93.     }
94.     // function that returns the number of processes
95.     MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
96.     // function that returns the rank of processes
97.     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
98.
99.     // Main menu loop
100.          do
101.          {
102.
103.              if (my_rank == 0)
104.              {
105.                  choice_mainmenu = main_menu(); //choose an operation from th
    e main menu
```

```
106.                }
107.
108.                // Broadcast επιλογή από την main menu
109.                MPI_Bcast(&choice_mainmenu, 1, MPI_INT, 0, MPI_COMM_WORLD);
110.
111.                // Εισαγωγή νέων δεδομένων
112.                if (choice_mainmenu == 1)
113.                {
114.
115.                    // Μήκος του πίνακα από το πληκτρολόγιο
116.                    if (my_rank == root)
117.                    {
118.                        printf("Give length of array X: ");
119.                        scanf("%d", &arrayX_size);
120.                    }
121.
122.                    // Αποστολή το μήκος του πίνακα με MPI_Bcast στους υπόλοιπου
    ς επεξεργαστές
123.                    MPI_Bcast(&arrayX_size, 1, MPI_INT, root, MPI_COMM_WORLD);
124.
125.                    // Υπολογισμός: πόσα νούμερα θα λάβει η κάθε διεργασία
126.
127.                    // Create an array that keeps how many numbers should be giv
    en to each process
128.                    sendCounts = (int *)malloc(sizeof(int) * numtasks);
129.                    // Check if malloc was successfully completed
130.                    if (!sendCounts)
131.                    {
132.                        printf("Error: Not available memory\n");
133.                        exit(EXIT_FAILURE);
134.                    }
135.
136.                    // dividing how many numbers should be given to each process
     & save it to array sendCounts[]
137.                    int mod;
138.                    mod = arrayX_size % numtasks;
139.                    for (i = 0; i < numtasks; i++)
140.                    {
141.
142.                        if (i < mod)
143.                        {
144.                            *(sendCounts + i) = arrayX_size / numtasks + 1;
145.                        }
146.                        else
147.                        {
148.                            *(sendCounts + i) = arrayX_size / numtasks;
149.                        }
150.                    }
151.
152.                    /*debugging: sendCounts[]
153.                    if (my_rank == 0)
154.                    {
155.                        for (i = 0; i < numtasks; i++)
156.                        {
157.                            printf("sendCounts[%d] = %d\n", i, sendCounts[i]);
158.                        }
159.                    }
160.                    */
161.
162.                    // Create an array that keeps the offsets
163.                    displacements = (int *)malloc(sizeof(int) * numtasks);
```

```
164.                    // Check if malloc was successfully completed
165.                    if (!displacements)
166.                    {
167.                        printf("Failure");
168.                        exit(0);
169.                    }
170.
171.                    // Αρχικοποίηση του πίνακα displacements
172.                    int index = 0;
173.                    for (i = 0; i < numtasks; i++)
174.                    {
175.
176.                        if (i == 0)
177.                        {
178.                            displacements[i] = index;
179.                        }
180.                        else
181.                        {
182.                            index += *(sendCounts + (i - 1));
183.                            displacements[i] = index;
184.                        }
185.                    }
186.
187.                    /*debugging: displacements[]
188.                    if (my_rank == 0)
189.                    {
190.                        for (i = 0; i < numtasks; i++)
191.                        {
192.                            printf("displacements[%d] = %d\n", i, displacements[
    i]);
193.                        }
194.                    }
195.                    */
196.
197.                    // allocate memory for localArrayX
198.                    localArrayX = (int *)malloc(sizeof(int) * sendCounts[my_rank
    ]);
199.                    // Check if malloc was successfully completed
200.                    if (localArrayX == NULL)
201.                    {
202.                        printf("Error: Not available memory\n");
203.                        exit(EXIT_FAILURE);
204.                    }
205.
206.                    /****************************************ΤΕΛΟΣ ΠΡΟΕΤΟΙΜΑΣΙΑ
    ΠΡΟΓΡΑΜΜΑΤΟΣ*********************************************/
207.
208.                    /* Δημιουργία πίνακα X στη διεργασί 0 & αρχικοποίησή του με
    τιμές από πληκτρολόγιο */
209.                    if (my_rank == 0)
210.                    {
211.                        // allocate memory for array X
212.                        arrayX = (int *)malloc(sizeof(int) * arrayX_size); // re
    turns the adress that memory has provided to the array
213.                        // Check if malloc was successfully completed
214.                        if (arrayX == NULL)
215.                        {
216.                            printf("Error: Not available memory\n");
217.                            exit(EXIT_FAILURE);
218.                        }
219.
```

```
220.                         // Call function initArrayX to initialize arrayX
221.                         initArrayX(arrayX, arrayX_size);
222.
223.                         /* debugging: print array X
224.                         printf("\nHello I am process: %d ", my_rank);
225.                         printf("Contents of array X\n");
226.                         for (i = 0; i < arrayX_size; i++)
227.                         {
228.                             printf("arrayX[%d] = %d\n", i, arrayX[i]);
229.                         }
230.                         */
231.                     }
232.
233.                     // Διαμοιρασμός πίνακα arrayX  και στέλνονται οι κατάλληλοι αριθμοί σε κάθε διεργασία
234.                     MPI_Scatterv(arrayX, sendCounts, displacements, MPI_INT, localArrayX, sendCounts[my_rank], MPI_INT, root, MPI_COMM_WORLD);
235.                     // printf(" Hello I am process %d and I have received number = %d", my_rank, number);
236.
237.                     /* debugging: local array X
238.                     printf("\nHello I am process: %d\nContents of local array X\n", sendCounts[my_rank]);
239.                     for (i = 0; i < sendCounts[my_rank]; i++)
240.                     {
241.                         printf("localArrayX[%d] = %d\n", i, localArrayX[i]);
242.                     }*/
243.
244.                     /*************************************************************/
245.                     /*                         Υπολογισμός Μέσης Τιμής                 */
246.
247.                     sum = 0;
248.
249.                     for (i = 0; i < sendCounts[my_rank]; i++)
250.                     {
251.                         sum += localArrayX[i];
252.                     }
253.
254.                     // Reducte & MPI_SUM the results of each process & calculate the average
255.                     MPI_Reduce(&sum, &tot_sum, 1, MPI_FLOAT, MPI_SUM, root, MPI_COMM_WORLD);
256.                     if (my_rank == root)
257.                     {
258.                         avg = tot_sum / (float)arrayX_size;
259.                         printf("The avg of all elements equalts to, avg=%.2f\n", avg);
260.                     }
261.
262.                     // Αποστολή από την ρίζα την τιμή της μέσης τιμής
263.                     MPI_Bcast(&avg, 1, MPI_FLOAT, root, MPI_COMM_WORLD);
264.
265.                     /* debugging: Αν έχουν λάβει όλοι την τιμή του MO
266.                     printf("Hello, I am process: %d and I have received from %d the value of avg = %.2f\n", my_rank, root, avg);
267.                     */
268.
269.                     /*************************************************************/
```

```
270.                    /*                              MIN & MAX
            */
271.
272.                    // find min & max of localArrayX of each process
273.                    local_min = localArrayX[0];
274.                    local_max = localArrayX[0];
275.
276.                    for (i = 0; i < sendCounts[my_rank]; i++)
277.                    {
278.                        if (localArrayX[i] < local_min)
279.                        {
280.                            local_min = localArrayX[i];
281.                        }
282.
283.                        if (localArrayX[i] > local_max)
284.                        {
285.                            local_max = localArrayX[i];
286.                        }
287.                    }
288.
289.                    // REDUCE & MPI_MIN: Find the minimum and send it to root
290.                    MPI_Reduce(&local_min, &min, 1, MPI_INT, MPI_MIN, root, MPI_
    COMM_WORLD);
291.
292.                    // REDUCE & MPI_MAX: Find the maximum and send it to root
293.                    MPI_Reduce(&local_max, &max, 1, MPI_INT, MPI_MAX, root, MPI_
    COMM_WORLD);
294.
295.                    if (my_rank == root)
296.                    {
297.                        printf("The minimum element value in array X is: %d\n",
    min);
298.                        printf("The maximum element value in array X is: %d\n",
    max);
299.                    }
300.
301.                    // Αποστολή από την ρίζα τις τιμές min & max
302.                    MPI_Bcast(&min, 1, MPI_FLOAT, root, MPI_COMM_WORLD);
303.                    MPI_Bcast(&max, 1, MPI_FLOAT, root, MPI_COMM_WORLD);
304.
305.                    /* debugging: Αν έχουν λάβει όλοι την τιμή του MO
306.                    printf("Hello, I am process: %d and I have received from %d
    the value of min = %d\n", my_rank, root, min);
307.                    printf("Hello, I am process: %d and I have received from %d
    the value of max = %d\n", my_rank, root, max);
308.                    */
309.
310.                    // Inner menu loop of main menu: για τα υποερωτήματα της άσκ
    ησης
311.                    int flag = 1;
312.                    do
313.                    {
314.                        if (my_rank == 0)
315.                        {
316.                            choice = menu(); //choose an operation from the menu
     (για τα υποερωτήματα της άσκησης)
317.                        }
318.
319.                        // Broadcast την επιλογή μενού για τα υποερωτήματα της ά
    σκησης
320.                        MPI_Bcast(&choice, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

```
321.
322.                         if (choice == 1) // ΕΡΩΤΗΣΗ Α
323.                         {
324.                             greaterLess(countGreater, countLess, sendCounts, my_
    rank, localArrayX, avg, &totalCountGreater, root, &totalCountLess);
325.
326.                             if (my_rank == 0)
327.                             {
328.                                 printf("Total elements that are greater than the
     average value: %d\n", totalCountGreater);
329.                                 printf("Total elements that are less than the av
    erage value: %d\n", totalCountLess);
330.                             }
331.                         }
332.                         else if (choice == 2) // ΕΡΩΤΗΣΗ Β
333.                         {
334.
335.                             var(numerator, sendCounts, my_rank, localArrayX, avg
    , &totalNumerators, root, arrayX_size);
336.                             if (my_rank == 0)
337.                             {
338.
339.                                 totalVar = totalNumerators / arrayX_size;
340.
341.                                 printf("Total var of elements: var = %.2f\n", to
    talVar);
342.                             }
343.                         }
344.                         else if (choice == 3) // ΕΡΩΤΗΣΗ Γ
345.                         {
346.                             // Allocate memory for localArrayD
347.                             localArrayD = (double *)malloc(sizeof(double) * send
    Counts[my_rank]);
348.                             //Check if malloc was executed successfully
349.                             if (localArrayD == NULL)
350.                             {
351.                                 printf("Error: Not available memory\n");
352.                                 exit(EXIT_FAILURE);
353.                             }
354.
355.                             // Αρχικοποίηση πίνακα localArrayX της κάθε διεργασί
    ας
356.                             for (i = 0; i < sendCounts[my_rank]; i++)
357.                             {
358.                                 localArrayD[i] = ((localArrayX[i] - min) / (doub
    le)(max - min)) * 100.0;
359.                                 //printf("\nI am process %d with localArrayD[%d]
     = %.15f\n", my_rank, i, localArrayD[i]);
360.                             }
361.
362.                             if (my_rank == root)
363.                             {
364.                                 // Allocate memory for arrayD
365.                                 arrayD = (double *)malloc(sizeof(double) * array
    X_size);
366.                                 //Check if malloc was executed successfully
367.                                 if (arrayD == NULL)
368.                                 {
369.                                     printf("Error: Not available memory\n");
370.                                     exit(EXIT_FAILURE);
371.                                 }
```

```
372.                             }
373.
374.                             // Send localArrayD of each process to the root and
    store it in array arrayD
375.                             MPI_Gatherv(localArrayD, sendCounts[my_rank], MPI_DO
    UBLE, arrayD, sendCounts, displacements, MPI_DOUBLE, root, MPI_COMM_WORLD);
376.
377.                             // Εκτύπωση πίνακα arrayD από την ρίζα
378.                             if (my_rank == root)
379.                             {
380.                                 for (i = 0; i < arrayX_size; i++)
381.                                 {
382.                                     printf("arrayD[%d] = %.15f\n", i, arrayD[i])
    ;
383.                                 }
384.                             }
385.                         }
386.                         else if (choice == 4) // ΕΡΩΤΗΣΗ Δ
387.                         {
388.                             if (localArrayD == NULL)
389.                             {
390.                                 if (my_rank == 0)
391.                                     printf("Question 3 should be first executed\
    n");
392.                             }
393.                             else
394.                             {
395.
396.                                 inmin.value = localArrayD[0];
397.                                 inmax.value = localArrayD[0];
398.                                 inmin.index = 0;
399.                                 inmax.index = 0;
400.
401.                                 // Υπολογισμός & εύρεση θέσης μικρότερης & μεγαλ
    ύτερης τιμής σε κάθε τοπικό πίνακα Δ της κάθε διεργασίας
402.                                 for (i = 1; i < sendCounts[my_rank]; i++)
403.                                 {
404.                                     if (inmin.value > localArrayD[i])
405.                                     {
406.                                         inmin.value = localArrayD[i];
407.                                         inmin.index = i;
408.                                     }
409.
410.                                     if (inmax.value < localArrayD[i])
411.                                     {
412.                                         inmax.value = localArrayD[i];
413.                                         inmax.index = i;
414.                                     }
415.                                 }
416.
417.                                 inmin.index = my_rank * sendCounts[my_rank] + in
    min.index;
418.                                 inmax.index = my_rank * sendCounts[my_rank] + in
    max.index;
419.
420.                                 MPI_Reduce(&inmin, &outmin, 1, MPI_FLOAT_INT, MP
    I_MINLOC, root, MPI_COMM_WORLD);
421.                                 MPI_Reduce(&inmax, &outmax, 1, MPI_FLOAT_INT, MP
    I_MAXLOC, root, MPI_COMM_WORLD);
422.
423.                                 if (my_rank == root)
```

```
424.                                 {
425.                                     minval = outmin.value;
426.                                     minrank = outmin.index / sendCounts[my_rank]
    ;
427.                                     minindex = outmin.index % sendCounts[my_rank
    ];
428.
429.                                     int globalindexmin = minrank * sendCounts[my
    _rank] + minindex;
430.
431.                                     maxval = outmax.value;
432.                                     maxrank = outmax.index / sendCounts[my_rank]
    ;
433.                                     maxindex = outmax.index % sendCounts[my_rank
    ];
434.
435.                                     int globalindexmax = maxrank * sendCounts[my
    _rank] + maxindex;
436.
437.                                     printf("The min val is %f\n", minval);
438.                                     printf("The rank that has the min val is ran
    k = %d\n", minrank);
439.                                     printf("The index is at %d\n", minindex);
440.                                     printf("The global index is at %d\n", global
    indexmin);
441.
442.                                     printf("The max val is %f\n", maxval);
443.                                     printf("The rank that has the max val is ran
    k = %d\n", maxrank);
444.                                     printf("The index is at %d\n", maxindex);
445.                                     printf("The global index is at %d\n", global
    indexmax);
446.                                 }
447.                             }
448.                         }
449.                         else if (choice == 5)
450.                         {
451.                             // Call function freeSpace to deallocate memory
452.                             freeSpace(arrayD, arrayX, localArrayD, localArrayX,
    sendCounts, displacements, my_rank);
453.
454.                             localArrayD = NULL;
455.
456.                             flag = 0;
457.                         }
458.                         else if (choice == 6)
459.                         {
460.                             exit(0);
461.                         }
462.                         else
463.                         {
464.                             if (my_rank == 0)
465.                                 printf("WRONG CHOICE\n");
466.                         }
467.
468.                     } while (flag);
469.                 }
470.                 else if (choice_mainmenu == 2) // ΕΠΙΛΟΓΗ 2 από την main menu
471.                 {
472.                     if (my_rank == 0)
473.                     {
```

```
474.                         printf("\ncs161020\nPaola Velasco\nSemester 9th\n");
475.                     }
476.                 }
477.                 else if (choice_mainmenu == 3) // ΕΠΙΛΟΓΗ 3 από την main menu
478.                 {
479.                     if (my_rank == 0)
480.                     {
481.                         printf("Exiting Program...\n");
482.                         exit(0);
483.                     }
484.                 }
485.             } while (1);
486.
487.             // end of parallelism
488.             MPI_Finalize();
489.
490.             return 0;
491.         }
492.
493.         // Συνάρτηση: εμφάνιση main menu & επιστροφή της τιμής της επιλογής
494.         int main_menu()
495.         {
496.             int choice_mainmenu;
497.             while (1)
498.             {
499.                 printf("\n\nProgramme Main Menu\n");
500.                 printf("1.) Εισαγωγή νέων δεδομένων \n");
501.                 printf("2.) Εμφάνιση στοιχείων της φοιτήτριας\n");
502.                 printf("3.) Exit\n");
503.
504.                 printf("\nEnter Your Menu Choice: ");
505.                 scanf("%d", &choice_mainmenu);
506.                 switch (choice_mainmenu)
507.                 {
508.
509.                 case 1:
510.                     printf("\nΕπιλογή 1\n\n");
511.                     return 1;
512.                     break;
513.
514.                 case 2:
515.                     printf("\nΕπιλογή 2\n");
516.                     return 2;
517.                     break;
518.
519.                 case 3:
520.                     printf("\nΕπιλογή 3\n");
521.                     return 3;
522.                     break;
523.
524.                 default:
525.                     printf("\nThis is not a valid Menu Option! Please Select Ano
    ther\n\n");
526.                     break;
527.                 }
528.             }
529.         }
530.
531.         // Συνάρτηση: αρχικοποίηση αρχικού πίνακα X
532.         void initArrayX(int arrayX[], int arrayX_size)
533.         {
```

```
534.            int i;
535.            // initialize array X
536.            printf("\nGive numbers for arrayX\n");
537.            for (i = 0; i < arrayX_size; i++)
538.            {
539.                printf("arrayX[%d]: ", i);
540.                scanf("%d", &arrayX[i]);
541.            }
542.        }
543.
544.        // Συνάρτηση: εμφάνιση menu υποερωτήματα της άσκησης & επιστροφή της τιμ
    ής της επιλογής
545.        int menu()
546.        {
547.            int choice;
548.            while (1)
549.            {
550.                printf("\n\nProgramme Main Menu\n");
551.                printf("1.) Πόσα στοιχεία του X έχουν μικρότερη και πόσα μεγαλύτ
    ερη τιμή από τη μέση τιμή; \n");
552.                printf("2.) Διασπορά των στοιχείων του διανύσματος X\n");
553.                printf("3.) Υπολόγισε τη ποσοστιάια σχέση των στοιχείων του X με
     τη διαφορά μεγίστου-ελαχίστου\n");
554.                printf("4.) Ποιά είναι η μεγαλύτερη τιμη του διανύσματος Δ και γ
    ια ποιό στοιχείο\n");
555.                printf("5.) Go back to main menu\n");
556.                printf("6.) Exit\n");
557.
558.                printf("\nEnter Your Menu Choice: ");
559.                scanf("%d", &choice);
560.                switch (choice)
561.                {
562.
563.                case 1:
564.                    printf("\nChoice 1\n");
565.                    return 1;
566.                    break;
567.
568.                case 2:
569.                    printf("\nChoice 2\n");
570.                    return 2;
571.                    break;
572.
573.                case 3:
574.                    printf("\nChoice 3\n");
575.                    return 3;
576.                    break;
577.
578.                case 4:
579.                    printf("\nChoice 4\n");
580.                    return 4;
581.                    break;
582.
583.                case 5:
584.                    printf("\nChoice 5\n");
585.                    return 5;
586.                    break;
587.
588.                case 6:
589.                    printf("\nChoice 6\n");
590.                    return 6;
```

```
591.                    break;
592.
593.                default:
594.                    printf("\nThis is not a valid Menu Option! Please Select Ano
    ther\n\n");
595.                    break;
596.                }
597.            }
598.        }
599.
600.        // Συνάρτηση: ερωτήματος Α ( Υπολογισμός για πόσες τιμές έχουν μεγαλύτερ
    η και πόσες μικρότερη από τη μέση τιμή )
601.        void greaterLess(int countGreater, int countLess, int sendCounts[], int
    my_rank, int localArrayX[], float avg, int *totalCountGreater, int root, int *to
    talCountLess)
602.        {
603.
604.            int i = 0;
605.
606.            // Έλεγχος σχέσης των στοιχείων localArrayX με την τιμή της μέσης τι
    μής
607.            countGreater = 0;
608.            countLess = 0;
609.            for (i = 0; i < sendCounts[my_rank]; i++)
610.            {
611.                if (localArrayX[i] > avg)
612.                    countGreater++;
613.                else if (localArrayX[i] < avg)
614.                    countLess++;
615.                else
616.                {
617.                    printf("localArray[%d] = %d is equal with avg = %.2f\n", i,
    localArrayX[i], avg);
618.                }
619.            }
620.
621.            //printf("Hello I am proccess %d and countGreater = %d while countLe
    ss = %d\n", my_rank, countGreater, countLess);
622.
623.            // Send the information to the root
624.            MPI_Reduce(&countGreater, totalCountGreater, 1, MPI_INT, MPI_SUM, ro
    ot, MPI_COMM_WORLD);
625.            MPI_Reduce(&countLess, totalCountLess, 1, MPI_INT, MPI_SUM, root, MP
    I_COMM_WORLD);
626.        }
627.
628.        // Συνάρτηση: ερώτημα Β ( Υπολογισμός Διασποράς )
629.        void var(float numerator, int sendCounts[], int my_rank, int localArrayX
    [], float avg, float *totalNumerators, int root, int arrayX_size)
630.        {
631.            int i;
632.            numerator = 0.0;
633.
634.            for (i = 0; i < sendCounts[my_rank]; i++)
635.            {
636.                numerator += (localArrayX[i] - avg) * (localArrayX[i] - avg);
637.            }
638.
639.            // Reduce & MPI_SUM  numerator to the root and store it to totalNume
    rators
```

```
640.            MPI_Reduce(&numerator, totalNumerators, 1, MPI_FLOAT, MPI_SUM, root,
    MPI_COMM_WORLD);
641.        }
642.
643.        void freeSpace(double arrayD[], int arrayX[], double localArrayD[], int
    localArrayX[], int sendCounts[], int displacements[], int my_rank)
644.        {
645.            if (my_rank == 0)
646.            {
647.                free(arrayD);
648.                free(arrayX);
649.            }
650.
651.            free(localArrayD);
652.            free(localArrayX);
653.            free(sendCounts);
654.            free(displacements);
655.        }
```