Message Passing Interface (MPI)

Συναρτήσεις Συλλογικής Επικοινωνίας

Συναρτήσεις Συλλογικής Επικοινωνίας

- **Συνάρτηση μετάδοσης δεδομένων από μία διεργασία μιας ομάδας σε όλες τις υπόλοιπες διεργασίες της ομάδας (Broadcast).**
- **Συνάρτηση συγκέντρωσης δεδομένων από όλες τις διεργασίες μιας ομάδας σε μία διεργασία της ομάδας (Gather).**
- **Συνάρτηση διαμοίρασης δεδομένων από μία διεργασία μιας ομάδας σε όλες τις διεργασίες της ομάδας (Scatter).**
- **Συνάρτηση συγκέντρωσης δεδομένων από όλες τις διεργασίες μιας ομάδας σε μία διεργασία όπου και αποθηκεύεται το αποτέλεσμα μιας πράξης (π.χ. γινόμενο, λογικό ή/και, μέγιστο) πάνω στα δεδομένα (Reduce).**
- **Παραλλαγές των παραπάνω συναρτήσεων όπου όλες οι διεργασίες συγκεντρώνουν δεδομένα από όλες τις διεργασίες.**

Μορφές Συναρτήσεων Συλλογικής Επικοινωνίας

1. **Η «απλή» στην οποία όλα τα μηνύματα που στέλνονται ή λαμβάνονται από μία διεργασία, έχουν το ίδιο μέγεθος, και**
2. **η «διανυσματική» στην οποία κάθε μήνυμα μπορεί να έχει διαφορετικό μέγεθος.**

- **Στην απλή έκδοση, αν έχουμε πολλά μηνύματα που στέλνονται ή λαμβάνονται από μία διεργασία, αυτά πρέπει να βρίσκονται σε συνεχόμενες διευθύνσεις στην μνήμη.**
- **Η διανυσματική μορφή επιτρέπει τα μηνύματα να βρίσκονται και σε μη διαδοχικές διευθύνσεις.**

Η συνάρτηση MPI_Barrier

- **Συγχρονίζει όλες τις διεργασίες που συμμετέχουν σε μια συλλογική επικοινωνία**
- **Προκαλεί αναστολή της εκτέλεσης της διεργασίας που την καλεί.**
- **Επιστρέφει μόνον όταν την καλέσουν όλες οι διεργασίες του communicator.**

int MPI_Barrier(MPI_Comm comm)

- **comm : ο communicator της συλλογικής επικοινωνίας.**

✓ **Όταν επιστρέψει, ξέρουμε ότι όλες οι διεργασίες έχουν φτάσει στο ίδιο σημείο στην εκτέλεση του κώδικά τους.**

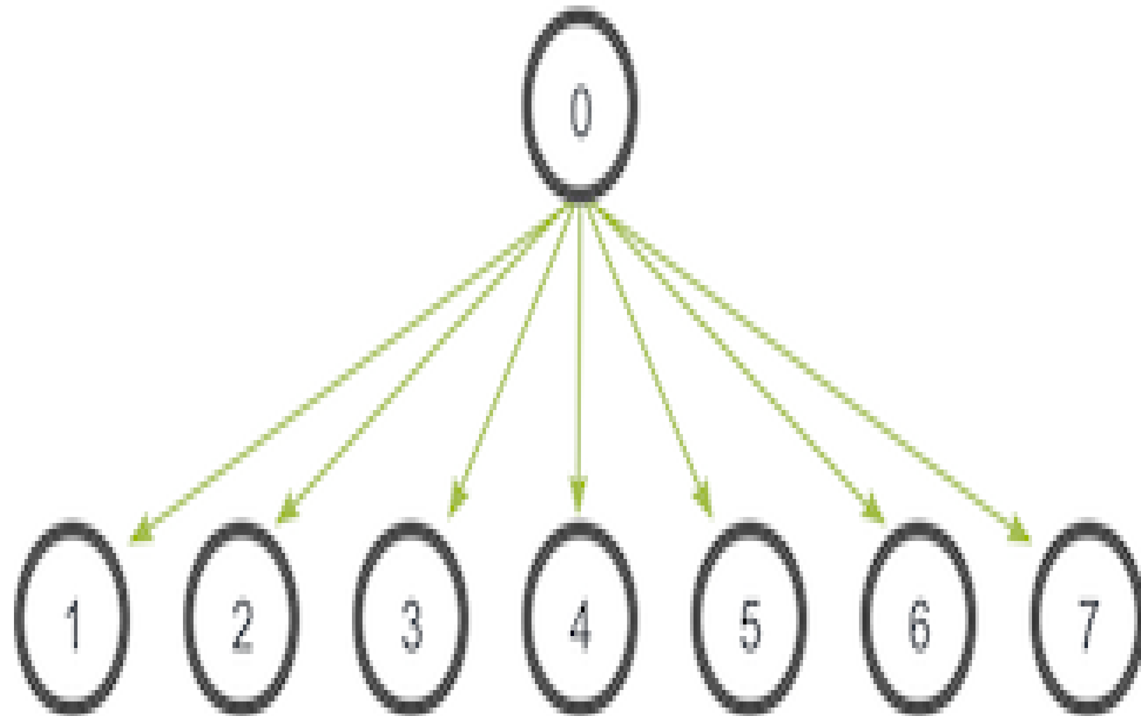
Η συνάρτηση MPI_Bcast

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

- **buffer: Η διεύθυνση της ενδιάμεσης μνήμης που περιέχει το μήνυμα.**
- **count: Το πλήθος των στοιχείων προς αποστολή.**
- **datatype: Ο τύπος των στοιχείων προς αποστολή.**
- **root: Η τάξη της διεργασίας-ρίζας.**
- **comm: Ο communicator της συλλογικής επικοινωνίας.**

- **Η κλήση της κάνει δημόσια εκπομπή (broadcast) των περιεχομένων της ενδιάμεσης μνήμης της ρίζας στην ενδιάμεση μνήμη που έχει οριστεί σε κάθε μια διεργασία του communicator.**
- **Όταν επιστρέψει, όλες οι ενδιάμεσες μνήμες των διεργασιών έχουν τα ίδια περιεχόμενα.**

Η συνάρτηση MPI_Bcast

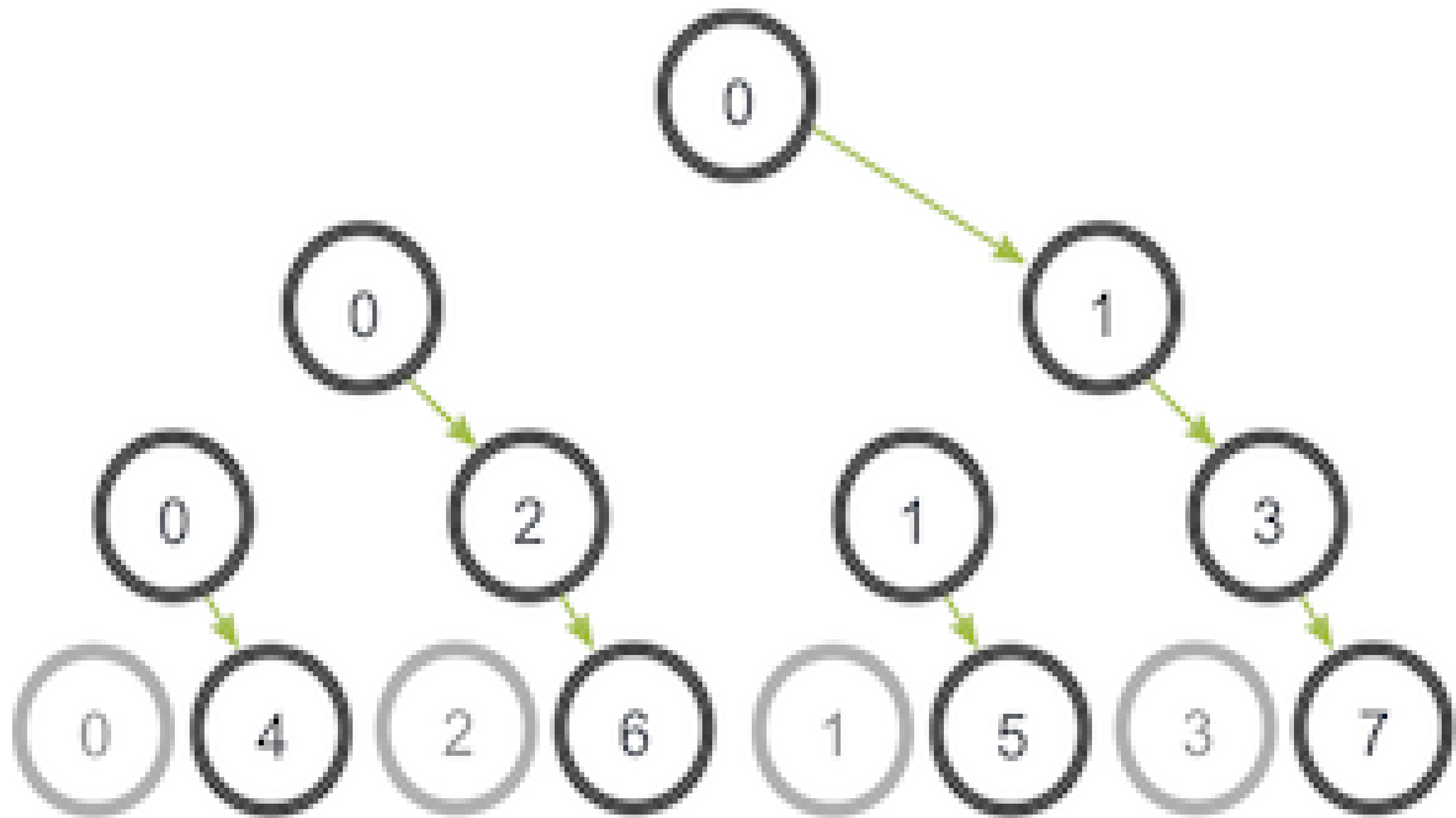


Broadcast με MPI_Send και MPI_Recv

```
#include "mpi.h"
#include <stdio.h>
 int main(argc,argv)
int argc;
char *argv[];
{
   int rank;
   int size, tag=100;
   int  data=20;
   MPI_Status status;
   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
```

```
  if (rank == 0) {
     int i;
     for (i = 1; i < size+1; i++) {
          MPI_Send(&data, 1, MPI_INT, i, tag,   MPI_COMM_WORLD); }
    } }
   else {
     MPI_Recv(&data, 1, MPI_INT, 0, tag, MPI_COMM_WORLD,
                             &status);
      printf("Process %d received data %d from  process\n", rank,
                        data);
 }

 MPI_Finalize();
}
```

Υλοποίηση της MPI_Bcast



Παράδειγμα: MPI_Bcast

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

int main(argc,argv)
int argc;
char *argv[];
{
   int rank;
   double var;
   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   if (rank == 1) var = 10.5;
   MPI_Bcast(&var, 1, MPI_DOUBLE, 1, MPI_COMM_WORLD);
   printf("Process %d var = %f \n", rank, var);

MPI_Finalize(); }
```

Παράδειγμα: MPI_Bcast

Πιθανή έξοδος του ανωτέρω προγράμματος αν εκκινηθούν 6 διεργασίες :

Στη Διεργασία 0 η τιμή της μεταβλητής είναι 10.5

Στη Διεργασία 1 η τιμή της μεταβλητής είναι 10.5

Στη Διεργασία 3 η τιμή της μεταβλητής είναι 10.5

Στη Διεργασία 5 η τιμή της μεταβλητής είναι 10.5

Στη Διεργασία 2 η τιμή της μεταβλητής είναι 10.5

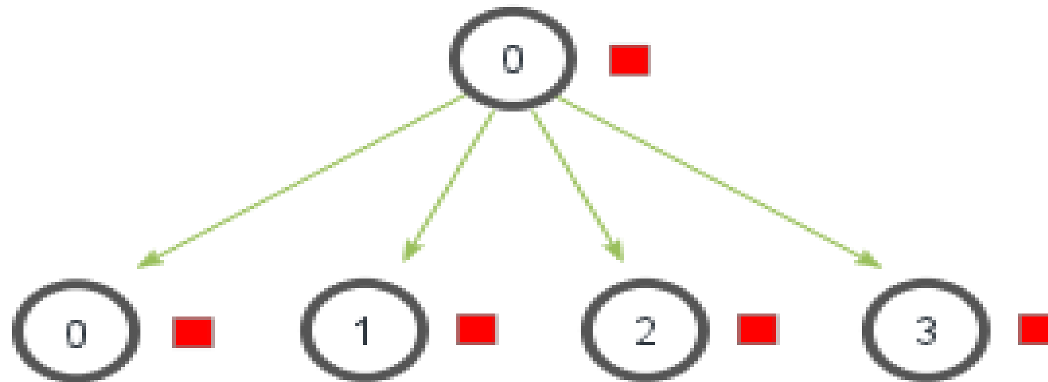
Στη Διεργασία 4 η τιμή της μεταβλητής είναι 10.5

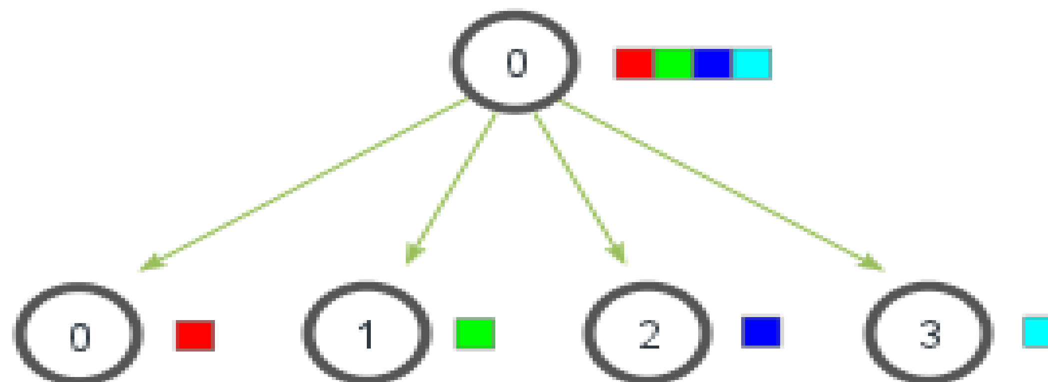
- **Πώς στέλνουμε σε όλες τις διεργασίες με ένα μήνυμα πάνω από ένα στοιχεία (π.χ. έναν πίνακα double αριθμών var[50]);**
- **Αρκεί να δηλώσουμε το πλήθος των στοιχείων στη δεύτερη παράμετρο, και να δηλώσουμε ως απλή μεταβλητή δείκτη το όνομα του πίνακα στην πρώτη παράμετρο:**

Στο προηγούμενο παράδειγμα:

```
MPI_Bcast(var, 50, MPI_DOUBLE, 1, MPI_COMM_WORLD);
```

Η συνάρτηση MPI_Scatter





Η συνάρτηση MPI_Scatter

Η διεργασία-ρίζα διαμοιράζει ένα μήνυμα σε πολλές διεργασίες κάθε μια από τις οποίες, λαμβάνει ένα μέρος του μηνύματος. Το μήνυμα χωρίζεται σε ίσα τμήματα πριν αποσταλεί.

int MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

sendbuf: Η διεύθυνση αρχής της ενδιάμεσης μνήμης αποστολής.

sendcount: Ο αριθμός στοιχείων που αποστέλλονται σε κάθε διεργασία.

sendtype: Ο τύπος των δεδομένων που περιέχονται στην sendbuf.

recvbuf: Η διεύθυνση αρχής της ενδιάμεσης μνήμης λήψης.

recvcount: Ο αριθμός στοιχείων που λαμβάνονται από κάθε διεργασία.

recvtype: Ο τύπος των δεδομένων που περιέχονται στην recvbuf.

root: Η τάξη της διεργασίας-ρίζας

comm: Ο communicator στον οποίο ανήκουν οι διεργασίες.

Η συνάρτηση MPI_Scatter

Η κλήση της MPI_Scatter είναι λειτουργικά ισοδύναμη με *n* κλήσεις

MPI_Send(sendbuf+i*sendcount, sendcount, sendtype, i, …),

για i = 0 έως n – 1 από τη ρίζα, ακολουθούμενη από μια κλήση

MPI_Recv(recvbuf, recvcount, recvtype, root, …)

από κάθε μία από τις υπόλοιπες διεργασίες

- **Η ενδιάμεση μνήμη κάθε διεργασίας περιέχει ένα μέρος του μηνύματος (ιδίου μεγέθους σε όλες τις διεργασίες)**
- **Οι τιμές των sendcount και sendtype στη ρίζα πρέπει να είναι ίδιες με τις recvcount και recvtype στις διεργασίες-παραλήπτες.**
- **Το ίδιο ισχύει και για τις παραμέτρους root και comm.**

Παράδειγμα: MPI_Scatter

Αποστολή 100 ακεραίων από τη ρίζα (sendarray) .

O buffer της ρίζας αρχικοποιείται με τον απαιτούμενο χώρο

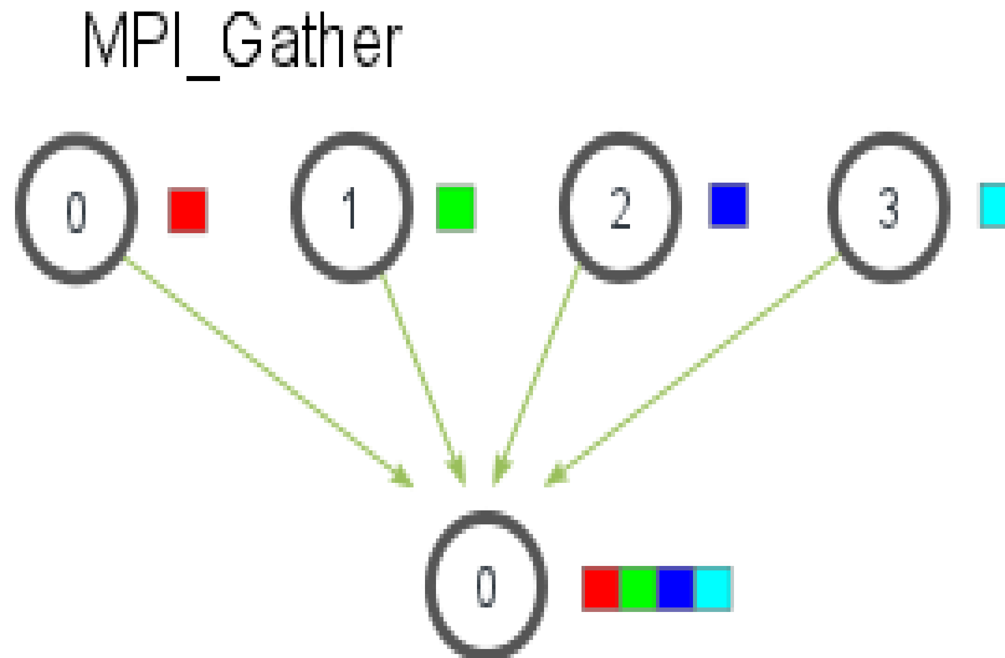
```
#include <stdio.h>
#include "mpi.h"
 main(int argc, char** argv)
{
int size, *sendarray;
int root, rbuf[100];
int i, rank;
 MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
```

Παράδειγμα: MPI_Scatter (συν.)

```
root = 0;

sendarray = (int *) malloc(100*size*sizeof(int));
if (rank==root)
 for (i=0; i<(100*size); i++)
  sendarray[i] = i+1;

MPI_Scatter(sendarray, 100, MPI_INT, rbuf, 100, MPI_INT,
          root, MPI_COMM_WORLD);

MPI_Finalize();
}
```

Η συνάρτηση MPI_Gather

MPI_Gather: **αντίστροφη λειτουργία από την MPI_Scatter. Δηλαδή, κάθε διεργασία (συμπεριλαμβανομένης και της ρίζας) στέλνει τα περιεχόμενα της ενδιάμεσης μνήμης στη ρίζα.**



Η συνάρτηση MPI_Gather

int MPI_Gather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

sendbuf:	**Η διεύθυνση αρχής της ενδιάμεσης μνήμης αποστολής.**
sendcount:	**Ο αριθμός των στοιχείων που αποστέλλονται από κάθε διεργασία.**
sendtype:	**Ο τύπος των στοιχείων που αποστέλλονται από κάθε διεργασία.**
recvbuf:	**Η διεύθυνση αρχής της ενδιάμεσης μνήμης λήψης (στη ρίζα).**
recvcount:	**Ο αριθμός των στοιχείων που λαμβάνονται.**
recvtype:	**Ο τύπος των στοιχείων που λαμβάνονται.**
root:	**Η τάξη της διεργασίας-παραλήπτη (δηλαδή της ρίζας).**
comm:	**Ο communicator στον οποίο ανήκουν οι διεργασίες.**

Η κλήση της MPI_Gather είναι λειτουργικά ισοδύναμη με την κλήση της

MPI_Send(sendbuf, sendcount, sendtype, root, …)

από κάθε μια από τις *n* διεργασίες που αποστέλλουν τα δεδομένα τους, ακολουθούμενη από *n* κλήσεις (για i=0…n-1) της

MPI_Recv(recvbuf+i * recvcount, recvcount, recvtype, i, …)

από τη διεργασία-ρίζα.

Παράδειγμα: MPI_Gather

Η ρίζα λαμβάνει 100 αριθμούς από κάθε μία από τις υπόλοιπες διεργασίες της ομάδας

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

main(int argc, char** argv)
{
int size, sendarray[100];
int root, *rbuf;
int i, rank;

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
```

```
root = 0;

for (i=0; i<100; i++)
  sendarray[i] = i*rank;
if (rank==root)
 rbuf = (int *) malloc(100*size*sizeof(int));

MPI_Gather(sendarray, 100, MPI_INT, rbuf, 100, MPI_INT,  root,
    MPI_COMM_WORLD);

MPI_Finalize();
}
```

O buffer της ρίζας αρχικοποιείται με τον απαιτούμενο συνολικά χώρο μνήμης, δηλαδή για να χωράει συνολικά [(αριθμός διεργασιών)*100] ακεραίους.

Υπολογισμός των 1^2 , 2^2 , ... , $(n-1)^2$, n^2 με broadcast, gather (gather.c)

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

main(int argc, char** argv)
{
int my_rank;
int p, k, count;
int root;
int a1_local;
int a2_local;
int local_num;
int num;
int local_res[50];
int final_res[50];
int namelen;
char proc_name[MPI_MAX_PROCESSOR_NAME];
```

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Get_processor_name(proc_name, &namelen);
if (my_rank == 0)
   {    printf("Dose plithos arithmvn:\n");
        scanf("%d", &num);   }
root = 0;
MPI_Bcast(&num, 1, MPI_INT, root, MPI_COMM_WORLD);
local_num = num/p;
a1_local = (my_rank * local_num) + 1;
a2_local = a1_local + local_num - 1;
count=0;
for (k=a1_local; k<=a2_local; k++)
   {   local_res[count] = (k*k);
         count++;
    }
```

```
printf("\n Process %d on %s : local squares are :  ", my_rank,
    proc_name);
for (k=0; k<local_num; k++) printf("%d ", local_res[k]);
printf("\n\n");

root = 0;
MPI_Gather(local_res, local_num, MPI_INT, final_res, local_num,
    MPI_INT, root, MPI_COMM_WORLD);

if (my_rank == 0)
   {
          printf("\n The %d squares are the following: ", endnum);
          for (k=0; k<num; k++) printf("%d ",final_res[k]);
           printf("\n\n");  }
MPI_Finalize();
}
```

Άσκηση: Υπολογισμός γινομένου αριθμού επί πίνακα (scatter.c)

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

main(int argc, char** argv)
{
int my_rank;
int p, k;
int b, size;
int root;
int matrixA[100];
int loc_num;
int loc_matrix[100];
int loc_res[100];
int final_res[100];
int namelen;
char proc_name[MPI_MAX_PROCESSOR_NAME];
```

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Get_processor_name(proc_name, &namelen);

if (my_rank == 0)
 {
   printf("YPOLOGISMOS THS PARASTASHS b * A \n\n");
   printf("DOSE THN TIMH TOY b:\n");
   scanf("%d", &b);
   printf("DOSE TO MHKOS TOY PINAKA A:\n");
   scanf("%d", &size);
   printf("DOSE TA STOIXEIA TOY PINAKA A MHKOYS %d:\n", size);
   for (k=0; k<size; k++)
        scanf("%d", &matrixA[k]);
  }
root = 0;
MPI_Bcast(&size, 1, MPI_INT, root, MPI_COMM_WORLD);
MPI_Bcast(&b, 1, MPI_INT, root, MPI_COMM_WORLD);
loc_num = size/p;
```

```
MPI_Scatter(matrixA, loc_num, MPI_INT, loc_matrix, loc_num, MPI_INT, root,
    MPI_COMM_WORLD);

for (k=0; k<loc_num; k++)
   loc_res[k] = b*loc_matrix[k];

printf("\n Process %d on %s : local results are :  ", my_rank, proc_name);
for (k=0; k<loc_num; k++) printf("%d ", loc_res[k]);
printf("\n\n");

root = 0;
MPI_Gather(loc_res, loc_num, MPI_INT, final_res, loc_num, MPI_INT, root,
    MPI_COMM_WORLD);

if (my_rank == 0)
 {
   printf("\n TELIKO APOTELESMA  %d *  A =\n", b);
   for (k=0; k<size; k++) printf("%d ", final_res[k]);
   printf("\n\n");
  }
MPI_Finalize();
}
```