Message Passing Interface (MPI)

Συναρτήσεις Συλλογικής Επικοινωνίας (συνέχεια)

Υπολογισμός του average τυχαίων αριθμών με χρήση των MPI_Scatter & MPI_Gather (avg.c)

- **Δημιουργούμε στη ρίζα (διεργασία 0) ένα διάνυσμα τυχαίων αριθμών στο διάστημα [0,1]. Ο αριθμός των στοιχείων του διανύσματος προσδιορίζεται από το χρήστη.**
- **Διαμοιράζουμε (scatter) το διάνυσμα σε όλες τις διεργασίες (ίσος αριθμός στοιχείων/διεργασία).**
- **Κάθε διεργασία υπολογίζει το μέσο όρο (average) των αριθμών που της αντιστοιχούν.**
- **Συγκεντρώνουμε (gather) όλους τους μέσους όρους στη ρίζα (διεργασία 0). Η ρίζα υπολογίζει το μέσο όρο όλων των αριθμών.**

Υπολογισμός του average

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <assert.h>
int main(int argc, char** argv)
  {
   MPI_Init(&argc,&argv);
  int my_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  int p;
  MPI_Comm_size(MPI_COMM_WORLD, &p);
  int num_elements;
  int num_elements_per_proc;
```

```
 if (my_rank == 0)
 {
   printf("Dose plithos arithmvn:\n");
   scanf("%d", &num_elements);
  }
MPI_Bcast(&num_elements, 1, MPI_INT, 0, MPI_COMM_WORLD);
num_elements_per_proc = num_elements/p;

float *rand_nums = (float *)malloc(sizeof(float) * num_elements);

if (my_rank == 0) {
  int i;
  for (i = 0; i < num_elements; i++) {
    rand_nums[i] = (rand() / (float)RAND_MAX);
printf("%f\n",rand_nums[i]) ;}}
```

```
float *sub_rand_nums = (float *)malloc(sizeof(float) *
    num_elements_per_proc);
 MPI_Scatter(rand_nums, num_elements_per_proc, MPI_FLOAT,
    sub_rand_nums, num_elements_per_proc, MPI_FLOAT, 0,
    MPI_COMM_WORLD);


 float sum = 0.0;
 float sub_avg;
   int l;
 for (l = 0; l < num_elements_per_proc; l++) {
   sum += sub_rand_nums[l];}
   sub_avg = sum/ (float)num_elements_per_proc ;
 float *sub_avgs ;
 if (my_rank == 0) {
   sub_avgs = (float *)malloc(sizeof(float) * p);
 }
```

```
MPI_Gather(&sub_avg, 1, MPI_FLOAT, sub_avgs, 1, MPI_FLOAT, 0,
    MPI_COMM_WORLD);

 if (my_rank == 0) {
   float avg;
   float tot_sum = 0.0;
   int j;
   for (j = 0; j < p; j++) {
   tot_sum += sub_avgs[j];}
   avg = tot_sum/ (float) p;
   printf("Avg of all elements is %f\n", avg);
  }

MPI_Finalize();
}
```

Πολλαπλασιασμός Πίνακα με Διάνυσμα (mv.c)

Δίνεται δισδιάστατος πίνακας B(NxN) και διάνυσμα C(Nx1) και θέλουμε να υπολογίσουμε παράλληλα το διάνυσμα

A(Nx1) = B(NxN) * C(Nx1)

- **Θεωρούμε ότι έχουμε στη διάθεσή μας τόσες διεργασίες όση και η διάσταση των πινάκων, δηλ. *p=N***

Πολλαπλασιασμός Πίνακα με Διάνυσμα (mv.c)

Κάθε διεργασία αναλαμβάνει τον υπολογισμό ενός στοιχείου του πίνακα αποτελέσματος Α. Για να επιτευχθεί αυτό κάθε διεργασία θα πρέπει να παραλάβει:

- **Μία γραμμή του πίνακα Β. Άρα ο πίνακας Β πρέπει να διαμοιραστεί μέσω της MPI_Scatter.**
- **Ολόκληρο το διάνυσμα C. Άρα το διάνυσμα C πρέπει να αποσταλεί σε όλες τις διεργασίες μέσω της MPI_Bcast.**

Στη συνέχεια

- **κάθε διεργασία με βάση τα δεδομένα που έχει λάβει υπολογίζει το στοιχείο του πίνακα Α που της αναλογεί, και**
- **τα στοιχεία από όλες τις διεργασίες μαζεύονται στη διεργασία-ρίζα μέσω της MPI_Gather.**

Πολλαπλασιασμός Πίνακα με Διάνυσμα (mv.c)

```
#include <stdio.h>
#include <mpi.h>
#define N 4

main(int argc, char **argv) {
      int j,k,A_loc;
      int rank,size,root;
      int A[N];
      int B_loc[N];
      int C[N];
      int B[N][N];

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

root = 0;
```

```
/* Αρχικοποίησε τους πίνακες B και C */

if (rank == root)
 {

   for (k=0; k<N; k++)
     for (j=0; j<N; j++)
        B[k][j] = k+j;

   for (k=0; k<N; k++)
        C[k] = k;
 }

/* Διαμοίρασε τον πίνακα B κατά γραμμές με χρήση της MPI_Scatter */

MPI_Scatter(B, N, MPI_INT, B_loc, N, MPI_INT, root, MPI_COMM_WORLD);

/* Στείλε σε όλες το διάνυσμα C με χρήση της MPI_Bcast */

MPI_Bcast(C, N, MPI_INT, root, MPI_COMM_WORLD);
```

```
/* Υπολόγισε το  στοιχείο του πίνακα A που σου αναλογεί */

for(j=0; j<N; j++)
  A_loc += C[j]*B_loc[j];

/* και συγκέντρωσε τα τελικά αποτελέσματα απλά με μία MPI_Gather */
 MPI_Gather(&A_loc, 1, MPI_INT, A, 1, MPI_INT, root,
     MPI_COMM_WORLD);

/* τύπωσε το τελικό αποτέλεσμα */
if (rank == 0)
 {
  for (k=0; k<N; k++)
    printf("A[%d]=%d\n", k, A[k]);
 }

MPI_Finalize();
}
```

Ασκήσεις

- Πρόσθεση δύο πινάκων B, C

- mpi/examples/max.c

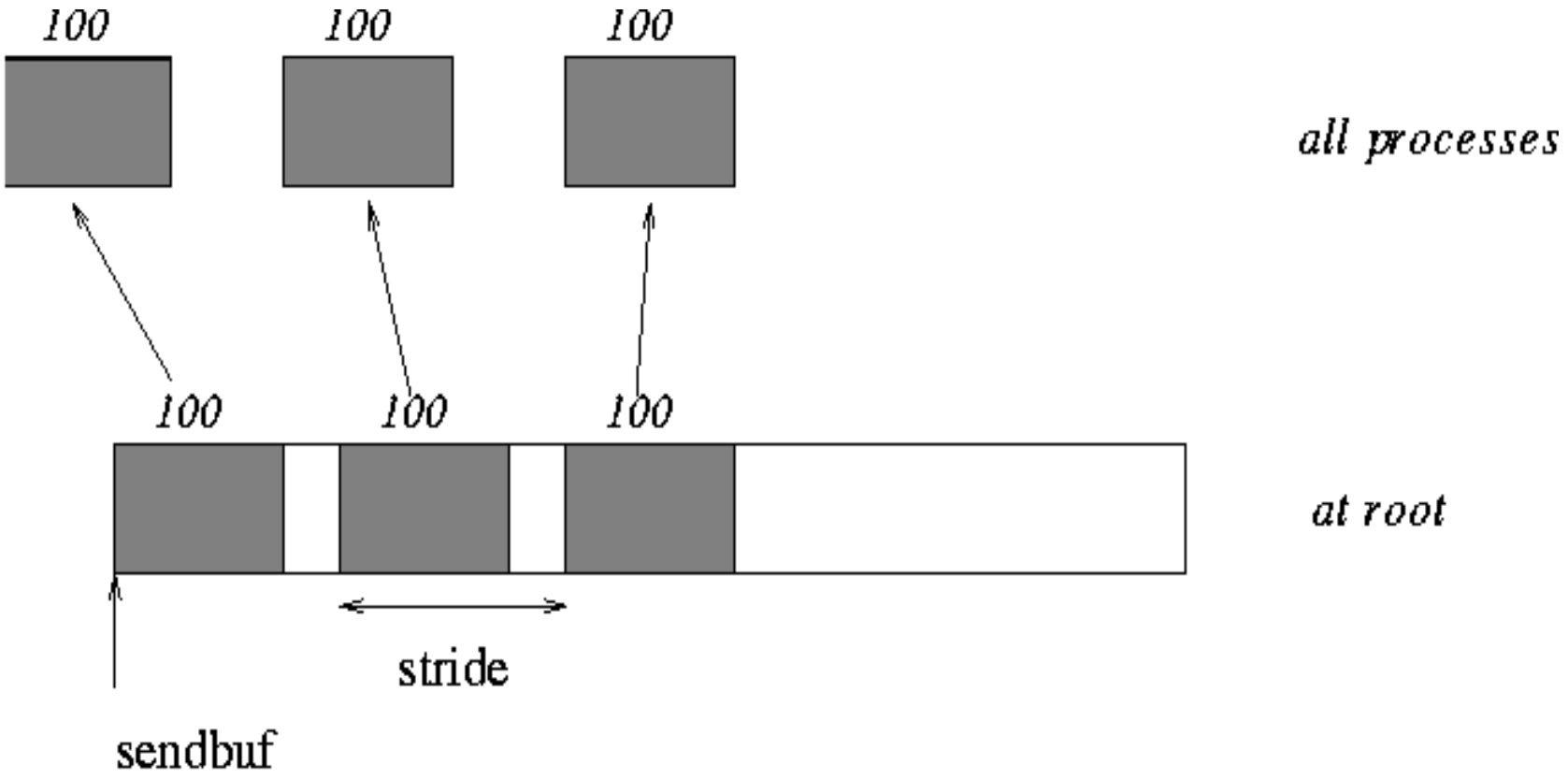
Η συνάρτηση MPI_Scatterv

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

sendbuf : Η διεύθυνση μνήμης αρχής (σημείο εκκίνησης αποστολής)
sendcounts: Πίνακας που δείχνει πόσα στοιχεία θα σταλούν σε κάθε διεργασία.
displs: Πίνακας που δείχνει το offset του μπλοκ στοιχείων που θα σταλεί σε κάθε διεργασία από το σημείο εκκίνησης.
sendtype: Ο τύπος δεδομένων των στοιχείων που θα αποσταλούν.
recvbuf: Η θέση μνήμης όπου θα αποθηκευτούν τα στοιχεία σε κάθε παραλήπτη.
recvcount: Ο αριθμός στοιχείων που θα παραλάβει κάθε διεργασία.
recvtype: Ο τύπος δεδομένων των στοιχείων που θα παραληφθούν.
root: Η ταυτότητα της διεργασίας που θα στείλει τα στοιχεία.
comm: Ο communicator όπου θα επενεργήσει η συνάρτηση.

Παράδειγμα - MPI_SCATTERV

Η ρίζα διαμοιράζει σύνολα 100 ακεραίων στις άλλες διεργασίες αλλά τα σύνολα των 100 ακεραίων απέχουν *stride* ακεραίους στον buffer παραλαβής. Υποθέτουμε ότι $stride \geq 100$



Παράδειγμα - MPI_SCATTERV

```
int np,*sendbuf;
 int root, rbuf[100], i, *displs, *scounts;  ...

MPI_Comm_size(MPI_COMM_WORLD, &np);
   sendbuf = (int *)malloc(np*stride*sizeof(int));
   ...
   displs = (int *)malloc(np*sizeof(int));
   scounts = (int *)malloc(np*sizeof(int));

   for (i=0; i<np; ++i) {
      displs[i] = i*stride;
      scounts[i] = 100;
   }
   MPI_Scatterv( sendbuf, scounts, displs, MPI_INT, rbuf, 100, MPI_INT, root,
MPI_COMM_WORLD);
```

Η συνάρτηση 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;)

sendbuf: Η θέση μνήμης απ' όπου θα σταλούν τα στοιχεία κάθε διεργασίας

sendcount: Ο αριθμός στοιχείων που θα στείλει κάθε διεργασία.

sendtype: Ο τύπος δεδομένων των στοιχείων που θα αποσταλούν.

recvbuf : Η διεύθυνση μνήμης αρχής (σημείο εκκίνησης για την αποθήκευση των παραληφθέντων στοιχείων).

recvcounts: Ο πίνακας που δείχνει πόσα στοιχεία θα παραληφθούν στη ρίζα από κάθε διεργασία.

displs: Ο πίνακας που δείχνει το offset από το σημείο εκκίνησης όπου θα αποθηκευτεί το μπλοκ που θα παραληφθεί από κάθε διεργασία.

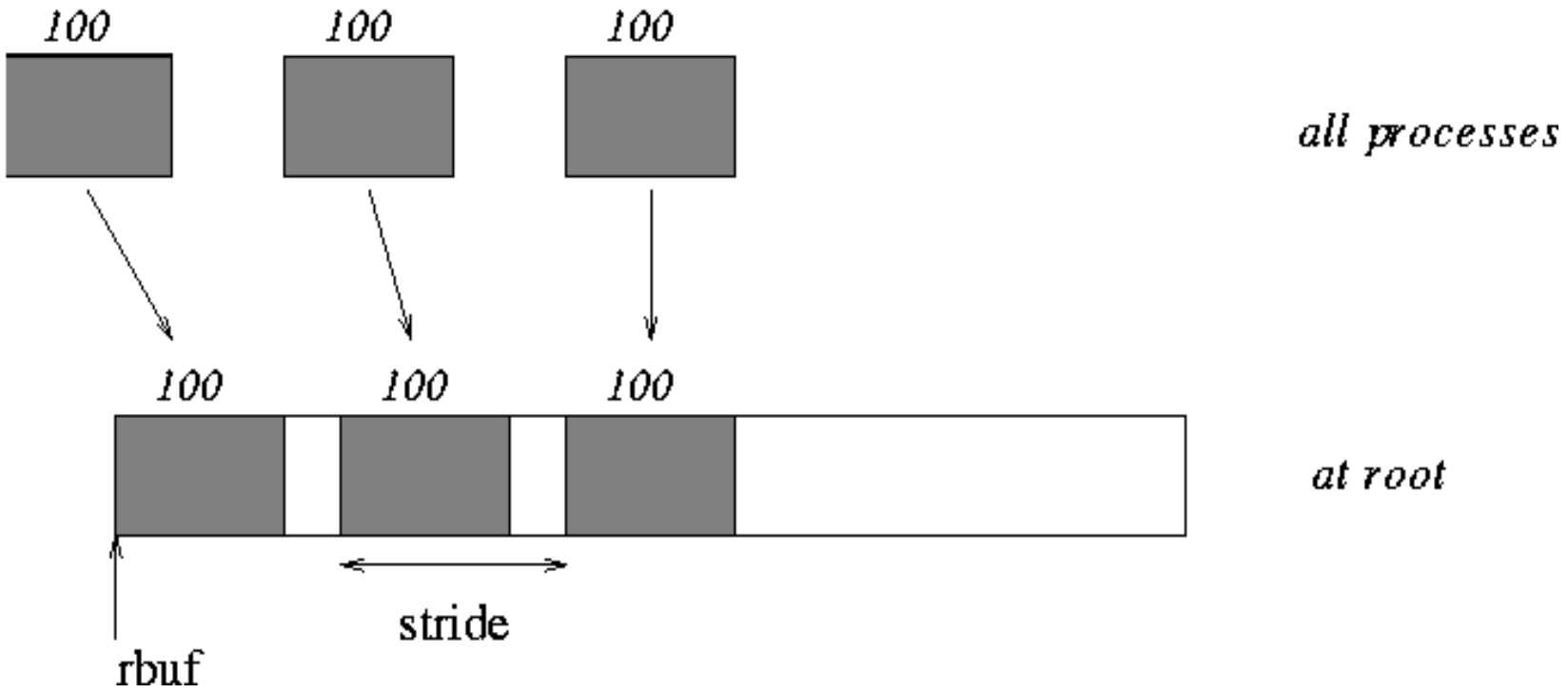
recvtype: Ο τύπος δεδομένων των στοιχείων που θα παραληφθούν.

root: Η ταυτότητα της διεργασίας που θα συγκεντρώσει τα στοιχεία.

comm: Ο communicator όπου θα επενεργήσει η συνάρτηση.

Παράδειγμα - MPI_GATHERV

Κάθε διεργασία στέλνει 100 ακεραίους στη ρίζα, αλλά κάθε σύνολοτων 100 ακεραίων τοποθετείται *stride* ακεραίους μακριά στον buffer παραλαβής. Υποθέτουμε ότι $stride \geq 100$



Παράδειγμα - MPI_GATHERV

```
int np, sendarray[100];
   int root, *rbuf, stride;
   int *displs,i,*rcounts;
 ...
 MPI_Comm_size(MPI_COMM_WORLD, &p);

   rbuf = (int *)malloc(np*stride*sizeof(int));
   displs = (int *)malloc(np*sizeof(int));
   rcounts = (int *)malloc(np*sizeof(int));

   for (i=0; i<np; ++i) {
      displs[i] = i*stride;
      rcounts[i] = 100;
   }
   MPI_Gatherv( sendarray, 100, MPI_INT, rbuf, rcounts, displs, MPI_INT, root,
                    MPI_COMM_WORLD);
```

gatherv_ex.c

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
/* Each processor sends a different amount of data to the root.  We use
    MPI_Gather first to tell the root how much data is going to be sent.
*/
int numnodes,myid;
#define mpi_root 0
int main(int argc,char *argv[]){
    int *myray,*displacements,*counts,*allray;
    int size,mysize,i;

    MPI_Init(&argc,&argv);
     MPI_Comm_size( MPI_COMM_WORLD, &numnodes );
     MPI_Comm_rank(MPI_COMM_WORLD, &myid);
```

```
mysize=myid+1;
    myray=(int*)malloc(mysize*sizeof(int));
    for(i=0;i<mysize;i++)
            myray[i]=myid+1;

/* counts and displacement arrays are only required on the root */
    if(myid == mpi_root){
            counts=(int*)malloc(numnodes*sizeof(int));
            displacements=(int*)malloc(numnodes*sizeof(int));
    }

/* we gather the counts to the root */
    MPI_Gather(myray,1,MPI_INT,counts,1,MPI_INT,mpi_root,
                    MPI_COMM_WORLD);
```

```
/* calculate displacements and the size of the recv array */
    if(myid == mpi_root){
            displacements[0]=0;
            for( i=1;i<numnodes;i++)
                    displacements[i]=counts[i-1]+displacements[i-1];
            size=0;
            for(i=0;i< numnodes;i++)
                    size=size+counts[i];
            allray=(int*)malloc(size*sizeof(int));
    }
/* different amounts of data from each processor   is gathered to the root */
     MPI_Gatherv(myray,mysize,MPI_INT,allray,counts,displacements,
    MPI_INT, mpi_root,MPI_COMM_WORLD);
     if(myid == mpi_root){
            for(i=0;i<size;i++)
                    printf("%d ",allray[i]);
            printf("\n");   }
  MPI_Finalize();  }
```

Αποστολή και παραλαβή τριγωνικού πίνακα (gatherv.c)

- **Ορίζεται ένας τετραγωνικός πίνακας NxN με τυχαία στοιχεία.**
- **Στη συνέχεια διαμοιράζεται (MPI_Scatterv) μόνο το κάτω τριγωνικό μέρος αυτού στις διεργασίες του συστήματος (υποθέτουμε ότι έχουμε p=N=8 διεργασίες).**
- **Δηλαδή, αποστέλεται το πρώτο στοιχείο της πρώτης γραμμής στη διεργασία 0, τα δύο πρώτα στοιχεία της δεύτερης γραμμής στη διεργασία 1, τα τρία πρώτα στοιχεία της τρίτης γραμμής στη διεργασία 2 κοκ.**

- **Αποστέλλεται διαφορετικός αριθμός στοιχείων σε κάθε διεργασία και**
- **τα διαφορετικά μπλοκς στοιχείων που αποστέλονται δεν είναι συνεχόμενα στη μνήμη αποστολής.**

Αποστολή και παραλαβή τριγωνικού πίνακα

- **Στη συνέχεια συγκεντρώνονται (MPI_Gatherv) ξανά τα μεταβλητά μπλοκ δεδομένων που έλαβε η κάθε διεργασία στη διεργασία-ρίζα, και**
- **αποθηκεύονται σε έναν πίνακα που έχει αρχικοποιηθεί με μηδενικά, έτσι ώστε να έχει στο τέλος τη μορφή ενός κανονικού κάτω-τριγωνικού πίνακα (με μηδενικά δηλαδή τα στοιχεία άνω της διαγωνίου).**

- **Συγκεντρώνεται στη διεργασία-ρίζα διαφορετικός αριθμός στοιχείων από κάθε διεργασία και**
- **αποθηκεύονται σε μη συνεχόμενες θέσεις στη μνήμη παραλαβής.**

```
#include <stdio.h>
#include "mpi.h"
#define N 8
main(int argc, char* argv[]) {
int i, j, np, me;
int root = 0;
  int x[N][N];
int y[N];
int res[N][N];

int *sendcnt, *recvcnt;
int *offsets1, *offsets2;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &np);
MPI_Comm_rank(MPI_COMM_WORLD, &me);
```

```
sendcnt = (int *)malloc(N*sizeof(int));
offsets1 = (int *)malloc(N*sizeof(int));

/* Αρχικοποίησε τους πίνακες sendcnt και offsets1 που δείχνουν πόσα στοιχεία
θα σταλούν στην κάθε διεργασία και με τι offset από το σημείο εκκίνησης */
for (i=0; i<N; i++) {
  sendcnt[i] = i+1;
  offsets1[i] = i*N;
}
if (me == 0) {
/* Αρχικοποίηση του πίνακα με τιμές 0 … N*N-1 */
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      x[i][j] = i*N+j;
}

/* Διαμοίρασε το κάτω τριγωνικό μέρος του πίνακα, γραμμή προς γραμμή */
MPI_Scatterv(&x, sendcnt, offsets1, MPI_INT, &y, sendcnt[me], MPI_INT, root,
MPI_COMM_WORLD);
```

```
if (me==0) {
/* Αρχικοποίησε τον πίνακα-αποτέλεσμα με μηδενικά */
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
      res[i][j] = 0;
}

recvcnt = (int *)malloc(N*sizeof(int));
offsets2 = (int *)malloc(N*sizeof(int));

/* Αρχικοποίησε τους πίνακες recvcnt και offsets2 οι οποίοι
δείχνουν πόσα στοιχεία θα παραληφθούν από κάθε διεργασία
και με τι offset θα αποθηκευτούν */

for (i=0; i<N; i++) {
   recvcnt[i] = i+1;
   offsets2[i] = i*N;
}
```

```
/* Συγκέντρωσε τα στοιχεία έτσι ώστε ο πίνακας-αποτέλεσμα στη ρίζα να πάρει τη μορφή κάτω τριγωνικού πίνακα */

MPI_Gatherv(&y, recvcnt[me], MPI_INT, &res, recvcnt, offsets2, MPI_INT, root, MPI_COMM_WORLD);
 if (me == 0) {
/* Τύπωσε τον πίνακα-αποτέλεσμα */
  printf("The result matrix after gathering is\n");
  for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
      printf("%4d ", res[i][j]);
    }
    printf("\n");  }
}
MPI_Finalize(); }
```