Ιωάννης Ε. Βενέτης

ΑΛΓΟΡΙΘΜΙΚΕΣ ΤΕΧΝΙΚΕΣ ΚΑΙ ΕΦΑΡΜΟΓΕΣ

29/01/2019

MPI (II)

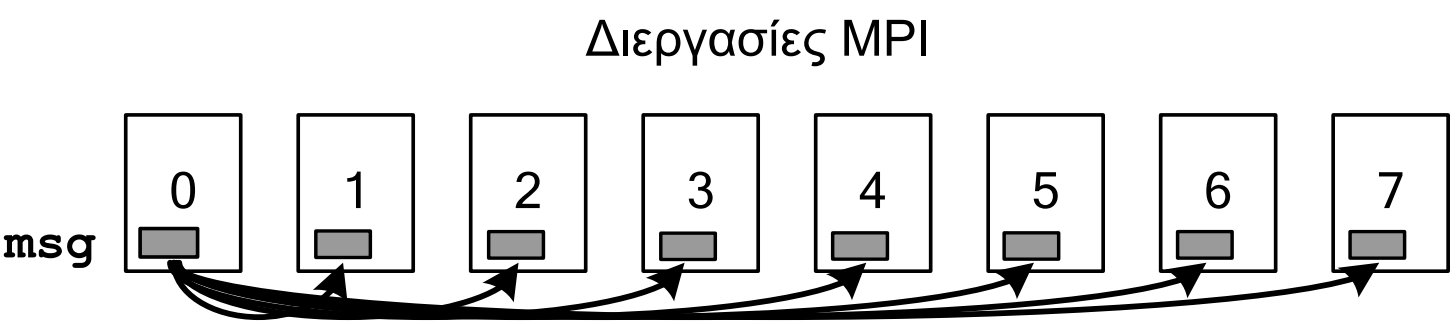
Συλλογική επικοινωνία (collective communication)

- Επιτρέπει την ανταλλαγή δεδομένων μεταξύ διεργασιών ενός προγράμματος MPI που ανήκουν στον ίδιο communicator
  - Broadcast
  - Reduction
  - Gather/Gatherv, Scatter/Scatterv, AllGather/AllGatherv
  - All-to-All
  - Barrier
- Υποστήριξη για τοπολογίες
- Αντιμετώπιση θεμάτων με τους buffers
  - Βελτιστοποίηση περάσματος μηνυμάτων
- Υποστήριξη για διάφορους τύπους δεδομένων

Γιατί συναρτήσεις συλλογικής επικοινωνίας; (1/2)

- Αποστολή μηνύματος από την διεργασία 0 στις διεργασίες 1-7

```
if (rank == 0) {
  for (dest = 1;dest < size; dest++)
       MPI_Send(msg,count,dest,tag,MPI_FLOAT,MPI_COMM_WORLD);
} else {
  MPI_Recv(msg,count,0,tag,MPI_FLOAT,MPI_COMM_WORLD,&status);
}
```

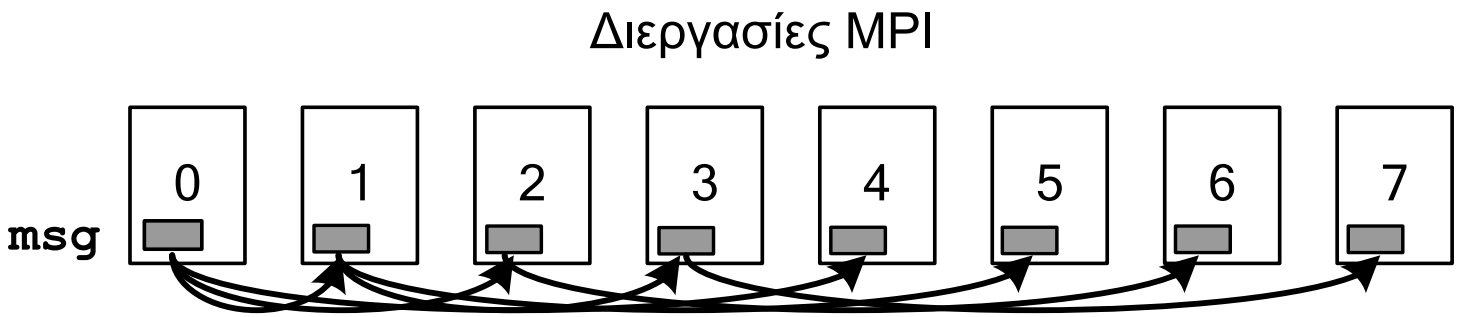


- Για p διεργασίες έχουμε p-1 βήματα επικοινωνίας

Γιατί συναρτήσεις συλλογικής επικοινωνίας; (2/2)

- Αποστολή μηνύματος από την διεργασία 0 στις διεργασίες 1-7

```
MPI_Bcast(msg,count,MPI_FLOAT,0,MPI_COMM_WORLD);
```



- Για p διεργασίες έχουμε $\lceil \log_2 p \rceil$ βήματα επικοινωνίας

Broadcast

- int MPI_Bcast(void *buffer, int count,
  MPI_Datatype datatype, int root,
  MPI_Comm comm);
  - “buffer”: Η διεύθυνση των δεδομένων προς αποστολή
  - “count”: Το πλήθος των στοιχείων που θα αποσταλούν
  - “datatype”: Ο τύπος κάθε στοιχείου
  - “root”: Ο επεξεργαστής ο οποίος αποστέλλει δεδομένα
    - Όλοι οι άλλοι στον communicator θα παραλάβουν
  - “comm”: Communicator

Reduction

- int MPI_Reduce(void *sendbuf, void *recvbuf, int count,
  MPI_Datatype datatype, MPI_Op op,
  int root, MPI_Comm comm);
  - “sendbuf”: Η διεύθυνση των δεδομένων προς αποστολή
  - “recvbuf”: Η διεύθυνση αποθήκευσης των δεδομένων που θα παραληφθούν
  - “count”: Το πλήθος των στοιχείων που θα αποσταλούν
  - “datatype”: Ο τύπος κάθε στοιχείου
  - “op”: Η πράξη που θα εκτελεστεί
  - “root”: Ο επεξεργαστής ο οποίος παραλαμβάνει δεδομένα
    - Όλοι στον communicator αποστέλλουν (και ο “root”)
  - “comm”: Communicator

Πράξεις reduction

- MPI_MAX → Μέγιστη τιμή
- MPI_MIN → Ελάχιστη τιμή
- MPI_SUM → Άθροισμα
- MPI_PROD → Γινόμενο
- MPI_LAND → Λογικό “ΚΑΙ”
- MPI_BAND → “ΚΑΙ” κατά bit
- MPI_LOR → Λογικό “Ή”
- MPI_BOR → “Ή” κατά bit
- MPI_LXOR → Λογικό “ΑΠΟΚΛΕΙΣΤΙΚΟ Ή”
- MPI_BXOR → “ΑΠΟΚΛΕΙΣΤΙΚΟ Ή” κατά bit
- MPI_MAXLOC → Μέγιστη τιμή και θέση
- MPI_MINLOC → Ελάχιστη τιμή και θέση

Υπολογισμός της παράστασης $1^2+2^2+...+N^2$ (1/2)

```
#include <stdio.h>
#include “mpi.h”

int main(int argc, char *argv[])
{
    int my_rank, p, i, res, finres, start, end, num, N;

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

    if (my_rank == 0) {
          printf(“Enter last number: ”);
          scanf(“%d”, &N);
    }
```

Υπολογισμός της παράστασης $1^2+2^2+...+N^2$ (2/2)

```
    MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);

    res   = 0;
    num   = N / p;
    start = (my_rank * num) + 1;
    end   = start + num;
    for (i = start; i < end; i++) {
          res += (i * i);
    }

    printf("\nResult of process %d: %d\n", my_rank, res);

    MPI_Reduce(&res, &finres, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    if (my_rank == 0) {
          printf("\n Total result for N = %d is equal to : %d \n", N, finres);
    }
    MPI_Finalize();
    return(0);
}
```

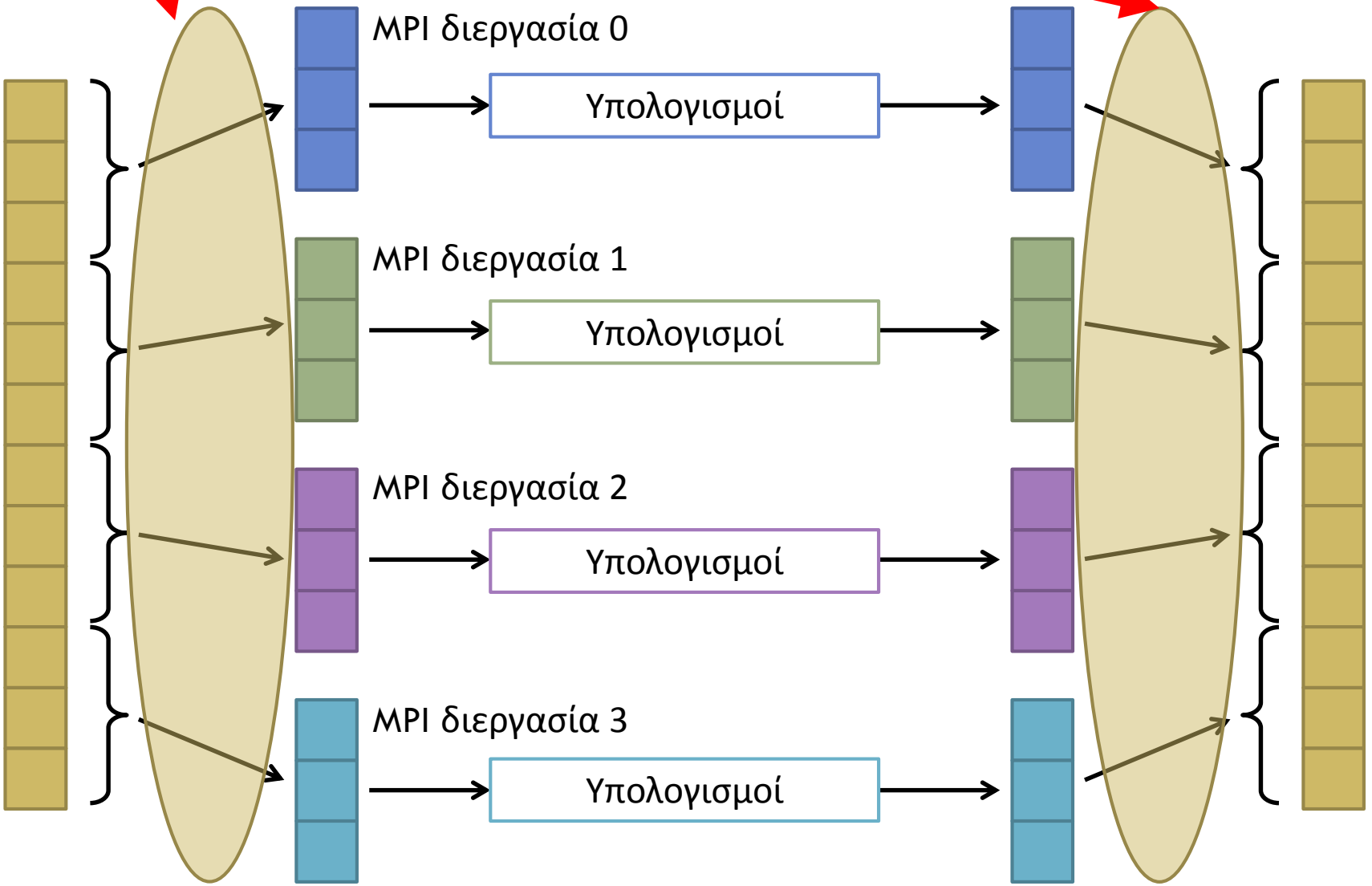
Scatter

- 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 των δεδομένων προς διαμοίραση
    - Στον επεξεργαστή root
  - “sendcount”: Πλήθος στοιχείων που αποστέλλονται προς κάθε επεξεργαστή
  - “sendtype”: Ο τύπος κάθε στοιχείου που αποστέλλεται
  - “recvbuf”: Η διεύθυνση αποθήκευσης των δεδομένων που θα παραληφθούν
    - Σε κάθε επεξεργαστή
  - “recvcount”: Πλήθος στοιχείων που παραλαμβάνονται από κάθε επεξεργαστή
  - “recvtype”: Ο τύπος κάθε στοιχείου που παραλαμβάνεται
  - “root”: Ο επεξεργαστής ο οποίος αποστέλλει δεδομένα
    - Όλοι οι άλλοι στον communicator θα παραλάβουν
  - “comm”: Communicator

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 των δεδομένων προς συγκέντρωση
    - Σε κάθε επεξεργαστή
  - “sendcount”: Πλήθος στοιχείων που αποστέλλονται προς τον επεξεργαστή root από κάθε επεξεργαστή
  - “sendtype”: Ο τύπος κάθε στοιχείου που αποστέλλεται
  - “recvbuf”: Η διεύθυνση αποθήκευσης των δεδομένων που θα παραληφθούν
    - Στον επεξεργαστή root
  - “recvcount”: Πλήθος στοιχείων που παραλαμβάνονται από κάθε επεξεργαστή
  - “recvtype”: Ο τύπος κάθε στοιχείου που παραλαμβάνεται
  - “root”: Ο επεξεργαστής ο οποίος συγκεντρώνει δεδομένα
    - Όλοι οι άλλοι στον communicator θα αποστείλουν
  - “comm”: Communicator

Scatter/Gather



Πολλαπλασιασμός διανύσματος με αριθμό (1/3)

```
#include <stdio.h>
#include “mpi.h”

int main(int argc, char *argv[])
{
   int my_rank, p, i, num, b, size, A[100], local_A[100];

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

   if (my_rank == 0) {
         printf(“Calculating b * A\n\n”);
         printf(“Enter value for b: ”);
         scanf(“%d”, &b);
         printf(“Enter size of vector A:”);
         scanf(“%d”, &size);
         printf(“Enter values of vector elements: ”, size);
         for (i = 0; i < size; i++) {
                  scanf(“%d”, &A[i]);
         }
   }
```

Πολλαπλασιασμός διανύσματος με αριθμό (2/3)

```
MPI_Bcast(&size, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&b, 1, MPI_INT, 0, MPI_COMM_WORLD);

num = size / p;

MPI_Scatter(A, num, MPI_INT, local_A, num, MPI_INT, 0, MPI_COMM_WORLD);

for (i = 0; i < num; i++) {
      local_A[i] *= b;
}

printf("\nLocal results for process %d:\n", my_rank);
for (i = 0; i < num; i++) {
      printf("%d ", local_A[i]);
}
printf("\n\n");
```

Πολλαπλασιασμός διανύσματος με αριθμό (3/3)

```
    MPI_Gather(local_A, num, MPI_INT, A, num, MPI_INT, 0, MPI_COMM_WORLD);

    if (my_rank == 0) {
          printf("\nFinal result:\n");
          for (i = 0; i < size; i++) {
                    printf("%d ", A[i]);
          }
          printf("\n\n");
    }

    MPI_Finalize();

    return(0);
}
```

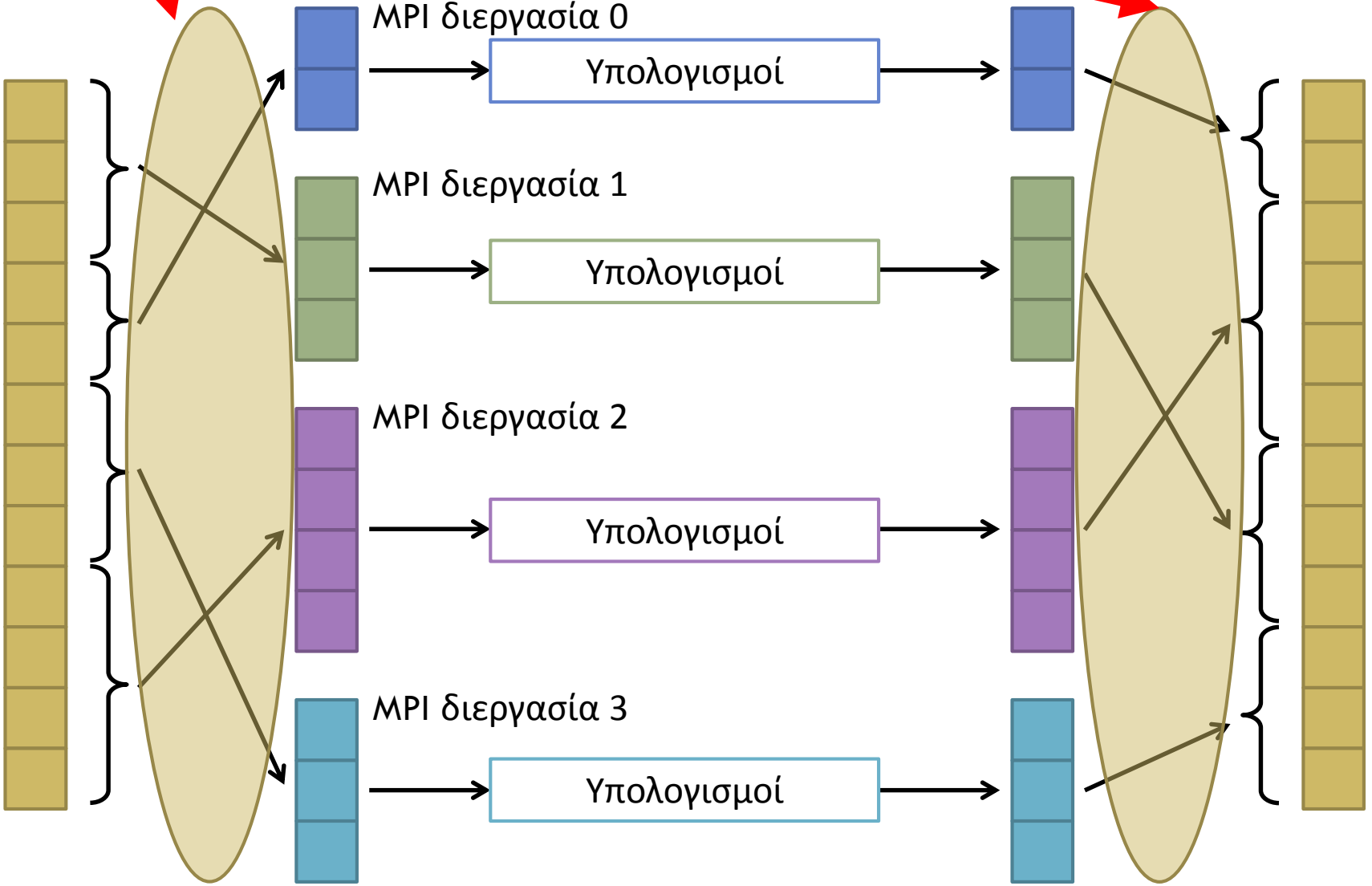
Scatterv

- int MPI_Scatterv(void *sendbuf, int *sendcounts, int *displs,
  MPI_Datatype sendtype, void *recvbuf, int recvcount,
  MPI_Datatype recvtype, int root, MPI_Comm comm);
  - “sendbuf”: Η διεύθυνση των δεδομένων προς διαμοίραση
    - Στον επεξεργαστή root
  - “sendcounts”: Διάνυσμα με το πλήθος στοιχείων που αποστέλλονται προς κάθε επεξεργαστή
  - “displs”: Διάνυσμα με τις μετατοπίσεις από την αρχή του διανύσματος “sendbuf” για κάθε επεξεργαστή
  - “sendtype”: Ο τύπος κάθε στοιχείου που αποστέλλεται
  - “recvbuf”: Η διεύθυνση αποθήκευσης των δεδομένων που θα παραληφθούν
    - Σε κάθε επεξεργαστή
  - “recvcount”: Πλήθος στοιχείων που παραλαμβάνονται από κάθε επεξεργαστή
  - “recvtype”: Ο τύπος κάθε στοιχείου που παραλαμβάνεται
  - “root”: Ο επεξεργαστής ο οποίος αποστέλλει δεδομένα
    - Όλοι οι άλλοι στον communicator θα παραλάβουν
  - “comm”: Communicator

Gatherv

- int MPI_Gatherv(void *sendbuf, int sendcount, MPI_Datatype sendtype,
  void *recvbuf, int *recvcounts, int *displs,
  MPI_Datatype recvtype, int root, MPI_Comm comm);
  - “sendbuf”: Η διεύθυνση των δεδομένων προς συγκέντρωση
    - Σε κάθε επεξεργαστή
  - “sendcount”: Πλήθος στοιχείων που αποστέλλονται προς τον επεξεργαστή root από κάθε επεξεργαστή
  - “sendtype”: Ο τύπος κάθε στοιχείου που αποστέλλεται
  - “recvbuf”: Η διεύθυνση αποθήκευσης των δεδομένων που θα παραληφθούν
    - Στον επεξεργαστή root
  - “recvcounts”: Διάνυσμα με το πλήθος στοιχείων που παραλαμβάνονται από κάθε επεξεργαστή
  - “displs”: Διάνυσμα με τις μετατοπίσεις από την αρχή του διανύσματος “recvbuf” για κάθε επεξεργαστή
  - “recvtype”: Ο τύπος κάθε στοιχείου που παραλαμβάνεται
  - “root”: Ο επεξεργαστής ο οποίος συγκεντρώνει δεδομένα
    - Όλοι οι άλλοι στον communicator θα αποστείλουν
  - “comm”: Communicator

Scatterv/Gatherv



Διαχείριση κάτω τριγωνικού μητρώου (1/4)

```
#include <stdio.h>
#include “mpi.h”

#define MAXPROC 8               /* Max number of procsses */
#define LENGTH  8               /* Size of matrix is LENGTH * LENGTH */

int main(int argc, char *argv[]) {
    int i, j, np, my_rank;
    int x[LENGTH][LENGTH];      /* Send buffer */
    int y[LENGTH];              /* Receive buffer */
    int res[LENGTH][LENGTH];    /* Final receive buffer */
    int *sendcount, *recvcount; /* Arrays for sendcounts and recvcounts */
    int *displs1, *displs2;     /* Arrays for displacements */

    MPI_Init(&argc, &argv);                          /* Initialize MPI */
    MPI_Comm_size(MPI_COMM_WORLD, &np);              /* Get nr of processes */
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);         /* Get own identifier */

    /* Check that we have one process for each row in the matrix */
    if (np != LENGTH) {
           if (my_rank == 0) {
               printf("You have to use %d processes\n", LENGTH);
           }
           MPI_Finalize();
           exit(0);
    }
```

Διαχείριση κάτω τριγωνικού μητρώου (2/4)

```
/* Allocate memory for the sendcount, recvcount and displacements arrays */
sendcount = (int *)malloc(LENGTH*sizeof(int));
recvcount = (int *)malloc(LENGTH*sizeof(int));
displs1   = (int *)malloc(LENGTH*sizeof(int));
displs2   = (int *)malloc(LENGTH*sizeof(int));

/* Should check for errors and inform other processes to terminate */

if (my_rank == 0) {    /* Process 0 does this */
       for (i = 0; i < LENGTH; i++) {
           for (j = 0; j < LENGTH; j++) {
               x[i][j] = i * LENGTH + j;
           }
       }

       printf("The initial matrix is\n");
       for (i = 0; i < LENGTH; i++) {
           for (j = 0; j < LENGTH; j++) {
               printf("%4d ", x[i][j]);
           }
           printf("\n");
       }
}
```

Διαχείριση κάτω τριγωνικού μητρώου (3/4)

```
/* Initialize sendcount and displacements arrays */
for (i = 0; i < LENGTH; i++) {
       sendcount[i] = i + 1;
       displs1[i] = i * LENGTH;
}

/* Scatter the lower triangular part of array x to all proceses, place it in y */
MPI_Scatterv(x, sendcount, displs1, MPI_INT, y, sendcount[my_rank], MPI_INT, 0,
MPI_COMM_WORLD);

if (my_rank == 0) {
       /* Initialize the result matrix res with 0 */
       for (i = 0; i < LENGTH; i++) {
           for (j = 0; j < LENGTH; j++) {
               res[i][j] = 0;
           }
       }

       /* Print out the result matrix res before gathering */
       printf("The result matrix before gathering is\n");
       for (i = 0; i < LENGTH; i++) {
           for (j = 0; j < LENGTH; j++) {
               printf("%4d ", res[i][j]);
           }
           printf("\n");
       }
}
```

Διαχείριση κάτω τριγωνικού μητρώου (4/4)

```
    for (i = 0; i < LENGTH; i++) {
           recvcount[i] = i + 1;
           displs2[i] = i * LENGTH;
    }

    /* Gather the local elements of each process to form a triangular matrix in the root */
    MPI_Gatherv(y, recvcount[my_rank], MPI_INT, res, recvcount, displs2, MPI_INT, 0,
    MPI_COMM_WORLD);

    if (my_rank == 0) {
           /* Print out the result matrix after gathering */
           printf("The result matrix after gathering is\n");
           for (i = 0; i < LENGTH; i++) {
               for (j = 0; j < LENGTH; j++) {
                   printf("%4d ", res[i][j]);
               }
               printf("\n");
           }
    }

    MPI_Finalize();
    exit(0);
}
```