Message Passing Interface (MPI)

Μη-Αναστέλλουσα Επικοινωνία

Μη-Αναστέλλουσα Επικοινωνία

- **Διαχωρισμός της έναρξης μιας διαδικασίας αποστολής ή παραλαβής μηνυμάτων από την ολοκλήρωσή της, παρέχοντας τη δυνατότητα κλήσης δύο διαφορετικών συναρτήσεων αποστολής ή παραλαβής.**
- **Η κλήση της πρώτης συνάρτησης αρχίζει τη διαδικασία αποστολής ή παραλαβής, ενώ η κλήση της δεύτερης ελέγχει εάν η διαδικασία αυτή έχει *ολοκληρωθεί*.**
- **Μεταξύ των δύο κλήσεων δεν αναστέλλεται η εκτέλεση της διεργασίας-αποστολέα ή της διεργασίας παραλήπτη.**

Μη-Αναστέλλουσα Επικοινωνία

Δύο βασικοί λόγοι χρήσης μη-αναστέλλουσας επικοινωνίας είναι οι ακόλουθοι:

1. **επίτευξη μικρότερων χρόνων επικοινωνίας**
2. **αποφυγή πρόκλησης αδιεξόδων (deadlocks)**

Μη-Αναστέλλουσα Επικοινωνία

- **Προκειμένου μία διεργασία η οποία ξεκινάει μία διαδικασία αποστολής ή παραλαβής, να αναφέρεται σε αυτήν τη διαδικασία, το MPI χρησιμοποιεί χειριστές σε αντικείμενα request (request handles).**
- **Το αντικείμενο request είναι μια δομή δεδομένων που δεσμεύει το MPI όταν γίνεται μια μη αναστέλλουσα επικοινωνία.**
- **Τα αντικείμενα βρίσκονται στη μνήμη συστήματος του MPI και δεν μπορούν να προσπελαστούν άμεσα από το πρόγραμμα του χρήστη (αδιαφανή αντικείμενα - opaque objects). Η προσπέλασή τους γίνεται με τη βοήθεια των χειριστών (handles).**

Η συνάρτηση MPI_Isend

int MPI_Isend (void *buf, int count, MPI_Datatype datatype,
int dest, int tag, MPI_Comm comm, MPI_Request *request)

- **buf: Η διεύθυνση αρχής της ενδιάμεσης μνήμης όπου βρίσκεται το μήνυμα.**
- **count: Ο αριθμός (πλήθος) των προς αποστολή στοιχείων.**
- **datatype: Ο τύπος των προς αποστολή στοιχείων**
- **dest: Η τάξη της διεργασίας-παραλήπτη.**
- **tag: Η ετικέτα του μηνύματος.**
- **comm: Ο communicator στον οποίο λαμβάνει χώρα η επικοινωνία.**
- **request: Χειριστής προς ένα αντικείμενο request.**

Η συνάρτηση MPI_Isend

- Στην παράμετρο request επιστρέφεται ο χειριστής του αντικειμένου request που δημιουργεί το MPI για τη συγκεκριμένη επικοινωνία.
- Ακόμη και όταν επιστρέψει η MPI_Isend, η διεργασία-αποστολέας δεν πρέπει να προσπελάσει την ενδιάμεση μνήμη και τις άλλες παραμέτρους της MPI_Isend, μέχρι να επιστρέψει η συνάρτηση ολοκλήρωσης της διαδικασίας αποστολής.

Η συνάρτηση MPI_Irecv

int MPI_Irecv (void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)

- **buf:** Η διεύθυνση της ενδιάμεσης μνήμης όπου θα αποθηκευθεί το μήνυμα.
- **count:** Ο μέγιστος αριθμός στοιχείων προς λήψη.
- **datatype:** Ο τύπος των στοιχείων που θα παραληφθούν.
- **source:** Η τάξη της διεργασίας-αποστολέα.
- **tag:** Η ετικέτα του μηνύματος.
- **comm:** Ο communicator στον οποίο λαμβάνει χώρα η επικοινωνία.
- **request:** Χειριστής προς ένα αντικείμενο request.

Η συνάρτηση MPI_Irecv

Ακόμη και όταν επιστρέψει η συνάρτηση MPI_Irecv, η διεργασία-παραλήπτης δεν πρέπει να προσπελάσει την ενδιάμεση μνήμη λήψης και τις υπόλοιπες παραμέτρους της MPI_Irecv, μέχρι να επιστρέψει η συνάρτηση ολοκλήρωσης της διαδικασίας παραλαβής.

Έλεγχος Ολοκλήρωσης μη-αναστέλλουσας επικοινωνίας

1. **Έλεγχος *με αναμονή*. MPI_Wait και παραλλαγές της:** αναστέλλουν την εκτέλεση της διεργασίας που τις καλεί.

   Μη αναστέλλουσα επικοινωνία ακολουθούμενη αμέσως από έλεγχο με αναμονή, είναι ισοδύναμη με μια αναστέλλουσα επικοινωνία.

2. **Έλεγχος *χωρίς αναμονή*. MPI_Test και παραλλαγές της:** επιστρέφουν true ή false ανάλογα με το αν έχει ολοκληρωθεί η επικοινωνία - **δεν** αναστέλλουν την εκτέλεση της διεργασίας που τις καλεί.

- **MPI_Wait και MPI_Test *ολοκληρώνουν* τις μη αναστέλλουσες διαδικασίες αποστολής και λήψης μηνυμάτων.**

- **Ολοκλήρωση αποστολής:** ο αποστολέας είναι ελεύθερος να προσπελάσει την ενδιάμεση μνήμη αποστολής.
- **Ολοκλήρωση λήψης:** ο παραλήπτης γνωρίζει ότι η ενδιάμεση μνήμη λήψης περιέχει το μήνυμα, ότι είναι ελεύθερος να το προσπελάσει, και ότι τα πεδία της δομής status έχουν έγκυρες τιμές.

int MPI_Wait (MPI_Request *request, MPI_Status *status)

- ***αντικείμενο 'request':*** **Χειριστής προς ένα αντικείμενο request (ο οποίος δημιουργήθηκε και επεστράφη κατά την έναρξη της διαδικασίας αποστολής ή λήψης μέσω των MPI_Isend, MPI_Irecv).**
- ***αντικείμενο 'status':*** **Στην περίπτωση μιας διαδικασίας παραλαβής, περιέχει πληροφορίες σχετικά με το μήνυμα που παρελήφθη. Στην περίπτωση μιας διαδικασίας αποστολής, μπορεί να περιέχει έναν κωδικό λάθους.**

- **Η παράμετρος request προσδιορίζει μία μη αναστέλλουσα διαδικασία αποστολής (posting send) ή παραλαβής (posting receive) που έχει κληθεί προηγουμένως.**
- **Η MPI_Wait επιστρέφει όταν ολοκληρωθεί η επικοινωνία που χαρακτηρίζεται από την request. Στη συνέχεια το αντικείμενο request καταστρέφεται και η παράμετρος request παίρνει την τιμή MPI_REQUEST_NULL.**

nonblock0.c

```
int main(int argc, char **argv) {
int i, my_rank, x, y;
MPI_Status status;
MPI_Request request;
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);
if (my_rank == 0) { /* P0 */
  x=10;
  MPI_Isend(&x,1,MPI_INT,1,0,MPI_COMM_WORLD,&request);
  MPI_Recv(&y,1,MPI_INT,1,0,MPI_COMM_WORLD,&status);
  MPI_Wait(&request,&status);
} else if (my_rank == 1) { /* P1 */
  y=20;
  MPI_Isend(&y,1,MPI_INT,0,0,MPI_COMM_WORLD,&request);
  MPI_Recv(&x,1,MPI_INT,0,0,MPI_COMM_WORLD,&status);
  MPI_Wait(&request,&status);
 }
MPI_Finalize();}
```

_Η συνάρτηση MPI_Test

Ελέχγει αν ολοκληρώθηκε η αποστολή ή η παραλαβή ενός μηνύματος.

int MPI_Test (MPI_Request *request, int *flag, MPI_Status *status)

- **request:** **Χειριστής προς το αντικείμενο request.**
- **flag:** **Γίνεται αληθής αν έχει ολοκληρωθεί η επικοινωνία.**
- **status:** **Επιστρέφει ένα αντικείμενο status.**

Επιστρέφει αμέσως μετά την κλήση της. Η request προσδιορίζει μία μη αναστέλλουσα διαδικασία αποστολής /παραλαβής.

Μη αναστέλλουσα παραλαβή: αν η flag *αληθής*, οι τιμές των source, tag και count των δεδομένων που ελήφθησαν, είναι διαθέσιμες μέσω της status.

Μη αναστέλλουσα αποστολή: αν η flag *αληθής*, η status μπορεί να περιέχει έναν κωδικό λάθους που επιστρέφει η διαδικασία αποστολής (όχι η MPI_Test).

Επειδή σε ένα MPI πρόγραμμα μπορεί να έχουμε πολλές μη αναστέλλουσες διαδικασίες επικοινωνίας που περιμένουν να ολοκληρωθούν, υπάρχουν συναρτήσεις που ελέγχουν την ολοκλήρωση πολλών μαζί επικοινωνιών.

Υπάρχουν τρία είδη τέτοιων συναρτήσεων. Αυτές που

- **ελέγχουν την ολοκλήρωση όλων των διαδικασιών επικοινωνίας,**
- **αυτές που ελέγχουν αν ολοκληρώθηκαν κάποιες διαδικασίες,**
- **αυτές που ελέγχουν αν ολοκληρώθηκε τουλάχιστον μία διαδικασία.**

Κάθε είδος συνάρτησης παρέχεται σε μορφή με και χωρίς αναμονή.

Η συνάρτηση MPI_Waitany

Αναστέλλει την εκτέλεση της διεργασίας που την καλεί, μέχρι να ολοκληρωθεί μία τουλάχιστον από τις διαδικασίες επικοινωνίας σε εκκρεμότητα.

Αν περισσότερες από μία μπορούν να ολοκληρωθούν, επιλέγεται μία αυθαίρετα.

Η συνάρτηση MPI_Waitany

int MPI_Waitany (int count, MPI_Request *array_of_requests, int *index, MPI_Status *status)

- **count:** Το πλήθος των διαδικασιών που ελέγχονται για ολοκλήρωση.
- **array_of_requests:** Ο πίνακας που περιέχει τους χειριστές request των προς έλεγχο διαδικασιών επικοινωνίας.
- **index:** Η θέση του πίνακα που αντιστοιχεί στο χειριστή request της επικοινωνίας που ολοκληρώθηκε.
- **status:** Δείκτης προς μια δομή status.

Η συνάρτηση επιστρέφει στη μεταβλητή index τη θέση του πίνακα του αντικειμένου request που αντιστοιχεί στην επικοινωνία που ολοκληρώθηκε, καθώς και στη μεταβλητή status την κατάσταση της επικοινωνίας.

Η συνάρτηση MPI_Testany

Ελέγχει αν έχει ολοκληρωθεί τουλάχιστον μια μη αναστέλλουσα διαδικασία επικοινωνίας σε εκκρεμότητα. Η κλήση της είναι:

```
int MPI_Testany (int count, MPI_Request  *array_of_requests, int *index,
                 int *flag, MPI_Status *status)
```

- **count:** Το πλήθος των διαδικασιών επικοινωνίας που ελέγχονται για ολοκλήρωση.
- **array_of_requests:** Ο πίνακας που περιέχει τους χειριστές request των προς έλεγχο διαδικασιών επικοινωνίας.
- **index:** Η θέση του πίνακα που αντιστοιχεί στο χειριστή request της επικοινωνίας που ολοκληρώθηκε.
- **flag:** Επιστρέφει true αν έχει ολοκληρωθεί τουλάχιστον μία επικοινωνία.
- **status:** Δείκτης προς μια δομή status.

- **Αν τουλάχιστον μία από τις επικοινωνίες έχει ολοκληρωθεί, η τιμή της flag γίνεται true, στη μεταβλητή index επιστρέφεται η θέση του πίνακα του αντικειμένου request που αντιστοιχεί στην επικοινωνία που ολοκληρώθηκε, και στην παράμετρο status η κατάσταση της επικοινωνίας.**

- **Το αντικείμενο request παύει να υπάρχει και ο χειριστής (παράμετρος) request παίρνει την τιμή της σταθεράς MPI_REQUEST_NULL.**

- **Αν καμία επικοινωνία δεν έχει ολοκληρωθεί, η τιμή της flag γίνεται false και η δομή status είναι ακαθόριστη. Σε αυτήν την περίπτωση στην index επιστρέφεται η τιμή της MPI_UNDEFINED.**

Η MPI_Testany (count, array_of_requests, &index, &flag, &status)

έχει το ίδιο αποτέλεσμα με την επαναληπτική κλήση της

MPI_Test (&array_of_requests[i], &flag, &status)

για i = 0, 1, … , count - 1, μέχρι κάποια κλήση να επιστρέψει flag = true ή μέχρι να αποτύχουν όλες οι κλήσεις της MPI_Test.

Στην πρώτη περίπτωση, η μεταβλητή index παίρνει την τελευταία τιμή του i και στη δεύτερη, παίρνει την τιμή MPI_UNDEFINED.

Η συνάρτηση MPI_Waitall

Αναστέλλει την εκτέλεση της διεργασίας που την καλεί, μέχρι να ολοκληρωθούν όλες οι διαδικασίες επικοινωνίας σε εκκρεμότητα.

int MPI_Waitall (int count, MPI_Request *array_of_requests,
MPI_Status *array_of_statuses)

- **count:** **Το πλήθος των διαδικασιών επικοινωνίας που ελέγχονται για ολοκλήρωση.**
- **array_of_requests:** **Πίνακας που περιέχει χειριστές request.**
- **array_of_statuses:** **Πίνακας που περιέχει δομές τύπου status.**

Το στοιχείο i του πίνακα array_of_requests παίρνει ως τιμή το αποτέλεσμα της i-οστής διαδικασίας επικοινωνίας.

Η κλήση

MPI_Waitall (count, &array_of_requests, &array_of_statuses)

είναι ισοδύναμη με την κλήση

MPI_Wait (&array_of_requests[i], &array_of_statuses[I]),

για i = 0, 1, … , count - 1.

Η συνάρτηση MPI_Testall

Επιστρέφει true αν έχουν ολοκληρωθεί όλες οι διαδικασίες επικοινωνίας σε εκκρεμότητα. Είναι μια τοπική συνάρτηση.

int MPI_Testall (int count, MPI_Request *array_of_requests, int *flag, MPI_Status *array_of_statuses)

- **count:** **Το πλήθος των διαδικασιών επικοινωνίας που ελέγχονται για ολοκλήρωση.**
- **array_of_requests:** **Πίνακας που περιέχει χειριστές request.**
- **flag:** **Αληθής αν έχουν ολοκληρωθεί όλες οι επικοινωνίες.**
- **array_of_statuses:** **Πίνακας που περιέχει δομές τύπου status.**

Αν έχουν ολοκληρωθεί όλες οι προς έλεγχο διαδικασίες επικοινωνίας, η τιμή της μεταβλητής flag γίνεται true, οι αντίστοιχες θέσεις του πίνακα status ενημερώνονται με τις αλλαγές

Παράδειγμα: αμφίδρομη ανταλλαγή μηνυμάτων σε τοπολογία δακτυλίου

- **Θεωρώντας ότι οι διεργασίες είναι οργανωμένες σε τοπολογία δακτυλίου , ζητείται να γράψουμε κώδικα για να στέλνει και να παραλαμβάνει κάθε διεργασία ένα μήνυμα στην/από την προηγούμενή της και ένα στην/από την επόμενή της. Πιο συγκεκριμένα, θα πρέπει,**
- **αρχικά να ορίσουμε την τοπολογία δακτυλίου (ring) καθορίζοντας ποια είναι η προηγούμενη και ποια η επόμενη της κάθε διεργασίας, και**
- **στη συνέχεια να εισάγουμε τις κατάλληλες συναρτήσεις μη-αναστέλλουσας αποστολής και παραλαβής ώστε να μη δημιουργείται αδιέξοδο**

nonblock.c

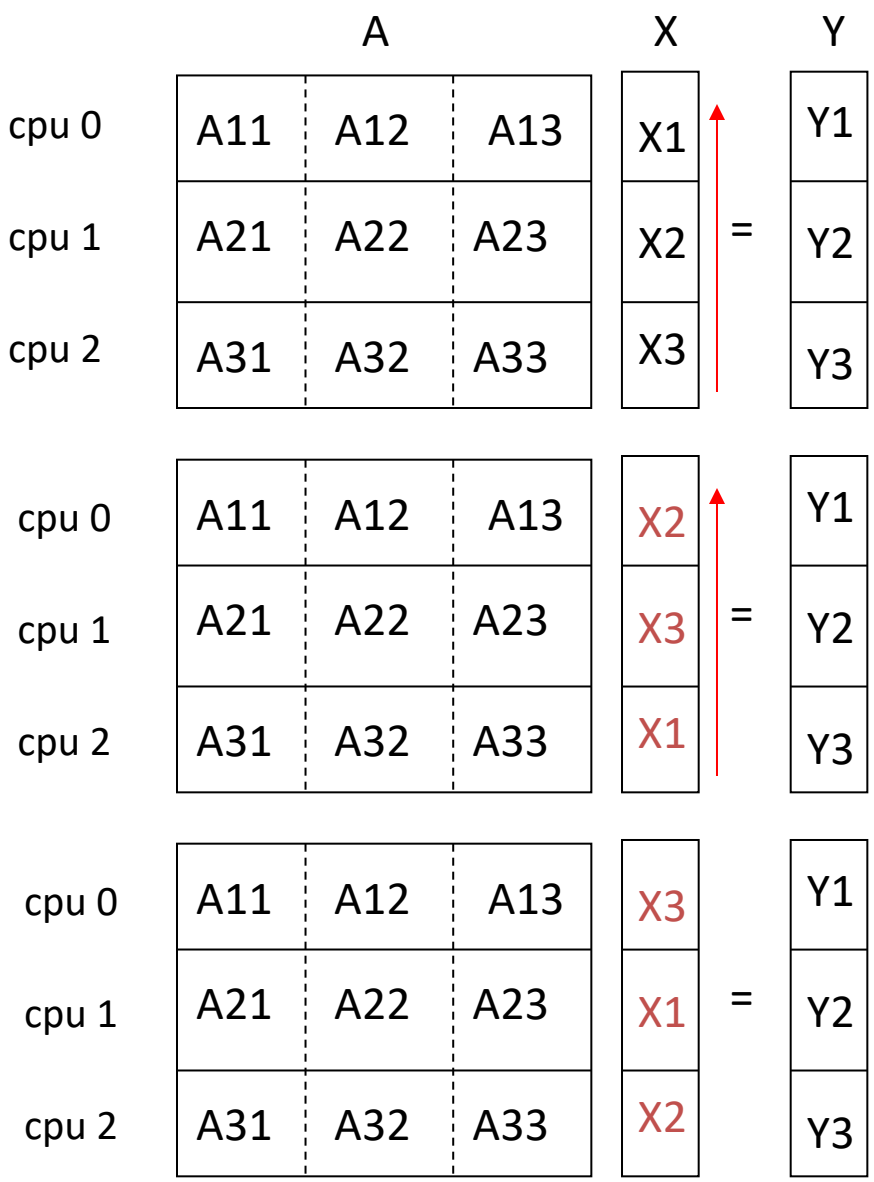
```
#include <stdio.h>
#include "mpi.h"
 int main(argc,argv)
int argc;
char *argv[];
{
 int numtasks, rank, next, prev;
int buf[2], tag1=1, tag2=2;
 /* δηλώνουμε κατάλληλα τα αντικείμενα τύπου Request και Status
    που θα χρειαστούν για τον τελικό έλεγχο της επιτυχούς ή όχι
    ολοκλήρωσης του συνόλου των επικοινωνιών*/
MPI_Request reqs[4];
MPI_Status stats[4];
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
```

```
prev = rank-1;
next = rank+1;
 if (rank == 0)  prev = numtasks - 1;
if (rank == (numtasks - 1))  next = 0;

 MPI_Irecv(&buf[0], 1, MPI_INT, prev, tag1, MPI_COMM_WORLD,
        &reqs[0]);
MPI_Irecv(&buf[1], 1, MPI_INT, next, tag2, MPI_COMM_WORLD,
        &reqs[1]);

MPI_Isend(&rank, 1, MPI_INT, prev, tag2, MPI_COMM_WORLD,
        &reqs[2]);
MPI_Isend(&rank, 1, MPI_INT, next, tag1, MPI_COMM_WORLD,
        &reqs[3]);
 MPI_Waitall(4, reqs, stats);
 MPI_Finalize();
}
```

Παράδειγμα: Πολλαπλασιασμός πίνακα-διανύσματος



AX=Y
A : NxN πίνακας
X,Y : διανύσματα διάστασης N

Y1 = A11*X1 + A12*X2 + A13*X3
Y2 = A21*X1 + A22*X2 + A23*X3
Y3 = A31*X1 + A32*X2 + A33*X3

Y1 = A11*X1 + A12*X2 + A13*X3
Y2 = A21*X1 + A22*X2 + A23*X3
Y3 = A31*X1 + A32*X2 + A33*X3

Y1 = A11*X1 + A12*X2 + A13*X3
Y2 = A21*X1 + A22*X2 + A23*X3
Y3 = A31*X1 + A32*X2 + A33*X3

Παράδειγμα: Πολλαπλασιασμός πίνακα-διανύσματος

Δεδομένα στη Διεργασία 0: **[A11 A12 A13] ← N/3 x N πίνακας**
X1 ← διάνυσμα μήκους N/3
Y1 ← διάνυσμα μήκους N/3

Δεδομένα στη Διεργασία 1: **[A21 A22 A23] ← N/3 x N πίνακας**
X2 ← διάνυσμα μήκους N/3
Y2 ← διάνυσμα μήκους N/3

Δεδομένα στη Διεργασία 2: **[A31 A32 A33] ← N/3 x N πίνακας**
X3 ← διάνυσμα μήκους N/3
Y3 ← διάνυσμα μήκους N/3

Θα πρέπει σε όλες τις διεργασίες να σταλούν τα X1, X2, X3
#shifts = np-1

```
A[i][j] = i+j
X[i] = i
```

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

#define DIM 20 // logical A[DIM][DIM], X[DIM], Y[DIM]

int main(int argc, char **argv)
{
  int np, my_rank, left_neighbor, right_neighbor, tag=100;
  int Nx, Ny; // Ny=DIM, Nx=DIM/np, on each process:A[Nx][Ny],X[Nx],Y[Nx]
  MPI_Request req_sr[2];
  MPI_Status stat_sr[2];
  double **A, *X, *Y, *Temp;

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

  if(DIM%np != 0) { // DIM is divided by np
    if(my_rank==0) printf("DIM cannot be divided by np\n");
    MPI_Finalize();
    return -1;
  }
```

```
Nx = DIM/np; // on each process: A[Nx][Ny]
Ny = DIM;

 left_neighbor = (my_rank-1 + np)%np; // left neighbor
 right_neighbor = (my_rank+1)%np; //  right neighbor

 A =  // allocate memory,
 X =
 Temp =  // Temp – temporary array for receiving data from right neighbor
 Y =

int i,j;
 for(i=0;i<Nx;i++) { // initialize A, X
   for(j=0;j<Ny;j++)
          A[i][j] = (my_rank*Nx+i) + j;
   X[i] = my_rank*Nx+i;
 }

 int count; // loop counter
 int index, curr_block;
 memset(Y, '\0', sizeof(double)*Nx);
```

```
for(count=0;count<np;count++){
    if(count < np-1) {
      MPI_Irecv(Temp, Nx, MPI_DOUBLE, right_neighbor, tag,
                    MPI_COMM_WORLD,&req_sr[0]);
                    // receive from bottom neighbor
      MPI_Isend(X, Nx, MPI_DOUBLE, left_neighbor, tag,
                     MPI_COMM_WORLD, &req_sr[1]);
                   // send to top neighbor
    }
    // compute on current data
    curr_block = (my_rank+count)%np;
    index = curr_block*Nx;//starting index of A[i][index+0:index+Nx-1]
    for(i=0;i<Nx;i++)
      for(j=0;j<Nx;j++)
         Y[i] += A[i][index+j]*X[j];

    // complete communication
    if(count<np-1) {
      MPI_Waitall(2, req_sr, stat_sr); // data now in Temp
      memcpy(X, Temp, sizeof(double)*Nx); // copy data from Temp to X
    }
  }
    MPI_Finalize();
}
```