Προηγμένες αρχιτεκτονικές υπολογιστών και προγραμματισμός παραλλήλων συστημάτων

LAB-06. MPI (Μέρος Γ') – Οι βασικές συναρτήσεις του MPI

Οι 6 βασικές συναρτήσεις του MPI

- Το MPI περιλαμβάνει 125 συναρτήσεις.
- Παρόλα αυτά 6 μόνο συναρτήσεις αρκούν για να υλοποιηθούν τα περισσότερα προγράμματα:
  - `MPI_Init`
  - `MPI_Finalize`
  - `MPI_Comm_size`
  - `MPI_Comm_rank`
  - `MPI_Send`
  - `MPI_Recv`

MPI_Init

- Εκκινεί το περιβάλλον εκτέλεσης του MPI.
- Μπορεί να κληθεί μόνο από το κυρίως thread.
- Αυτό θα είναι και το thread που θα καλέσει την `MPI_Finalize()`.
- Δέχεται δύο ορίσματα:
  - `argc`: Pointer προς το πλήθος των ορισμάτων της εφαρμογής.
  - `argv`: Pointer προς τον πίνακα ορισμάτων της εφαρμογής.
  - Μπορεί να περαστεί `NULL` και για τα δύο.

MPI_Finalize

- Τερματίζει το περιβάλλον εκτέλεσης του MPI.
- Δεν δέχεται ορίσματα.
- Πρέπει να καλείται από το κυρίως thread όλων των διεργασιών του MPI.
- Μετά την κλήση της, ο αριθμός των διεργασιών που εκτελούνται είναι απροσδιόριστος.
- Δεν συστήνεται η διενέργεια εργασιών μετά την κλήση της.

MPI_Comm_size

- Καθορίζει το μέγεθος μιας ομάδας διεργασιών που σχετίζεται με ένα communicator.
- Δέχεται δύο ορίσματα:
  - `comm`: ένας pointer προς τον communicator (αντικείμενο τύπου `MPI_comm`).
  - `size`: το μέγεθος της ομάδας (ακέραιος pointer – `int *size`).
- Είναι thread-safe: Μπορεί να κληθεί με ασφάλεια από πολλαπλά threads.

MPI_Comm_rank

- Καθορίζει την τάξη της διεργασίας που εκτελείται μέσα στην ομάδα ενός communicator.
- Δέχεται δύο ορίσματα:
  - `comm`: ένας pointer προς τον communicator (αντικείμενο τύπου `MPI_comm`).
  - `rank`: τάξη της ομάδας μέσα στην ομάδα του `comm` (ακέραιος pointer – `int *rank`).
- Είναι thread-safe: Μπορεί να κληθεί με ασφάλεια από πολλαπλά threads.

Ένα απλό πρόγραμμα

- Όλες οι μη-MPI ρουτίνες εκτελούνται τοπικά σε κάθε διεργασία. Έτσι, πχ, το `printf` εκτελείται σε κάθε διεργασία.

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

int main (int argc, char **argv) {
        int rank, size;
        MPI_Init (&argc, &argv);
        MPI_Comm_rank (MPI_COMM_WORLD, &rank);
        MPI_Comm_size (MPI_COMM_WORLD, &size);
        printf ("Hello world! I'm %d of %d\n", rank, size);
        MPI_Finalize ();
        return 0;
}
```

Πέρασμα μηνυμάτων

- Αποστολή μηνύματος point-to-point:
  - `send(dest, type, address, length)`
    - `dest`: ένας ακέραιος που αναπαριστά τη διεργασία (rank) που θα λάβει το μήνυμα.
    - `type`: ένας θετικός ακέραιος τον οποίο ο παραλήπτης μπορεί να χρησιμοποιήσει για να επιλέξει μηνύματα.
    - (`address, length`) περιγράφει ένα ενιαίο χώρο στη μνήμη που περιέχει το μήνυμα προς αποστολή.

- Αποστολή μηνύματος προς όλους (global):
  - `broadcast(type, address, length)`

Buffer

- Η αποστολή και η λήψη μόνο μιας συνεχούς σειράς από bytes έχει μειονεκτήματα:
- Κρύβεται από το hardware η πραγματική δομή των δεδομένων.
- Απαιτείται πακετάρισμα διάσπαρτων δεδομένων.
  - Π.χ. γραμμές πίνακα αποθηκευμένου κατά στήλες.
  - Συλλογές διαφορετικών τύπων δομών δεδομένων.
- Εμποδίζει την επικοινωνία μεταξύ μηχανών με διαφορετικές αναπαραστάσεις των τύπων των δεδομένων (πχ το μήκος ή το alignment ενός `int`, ενός `double`, κλπ).

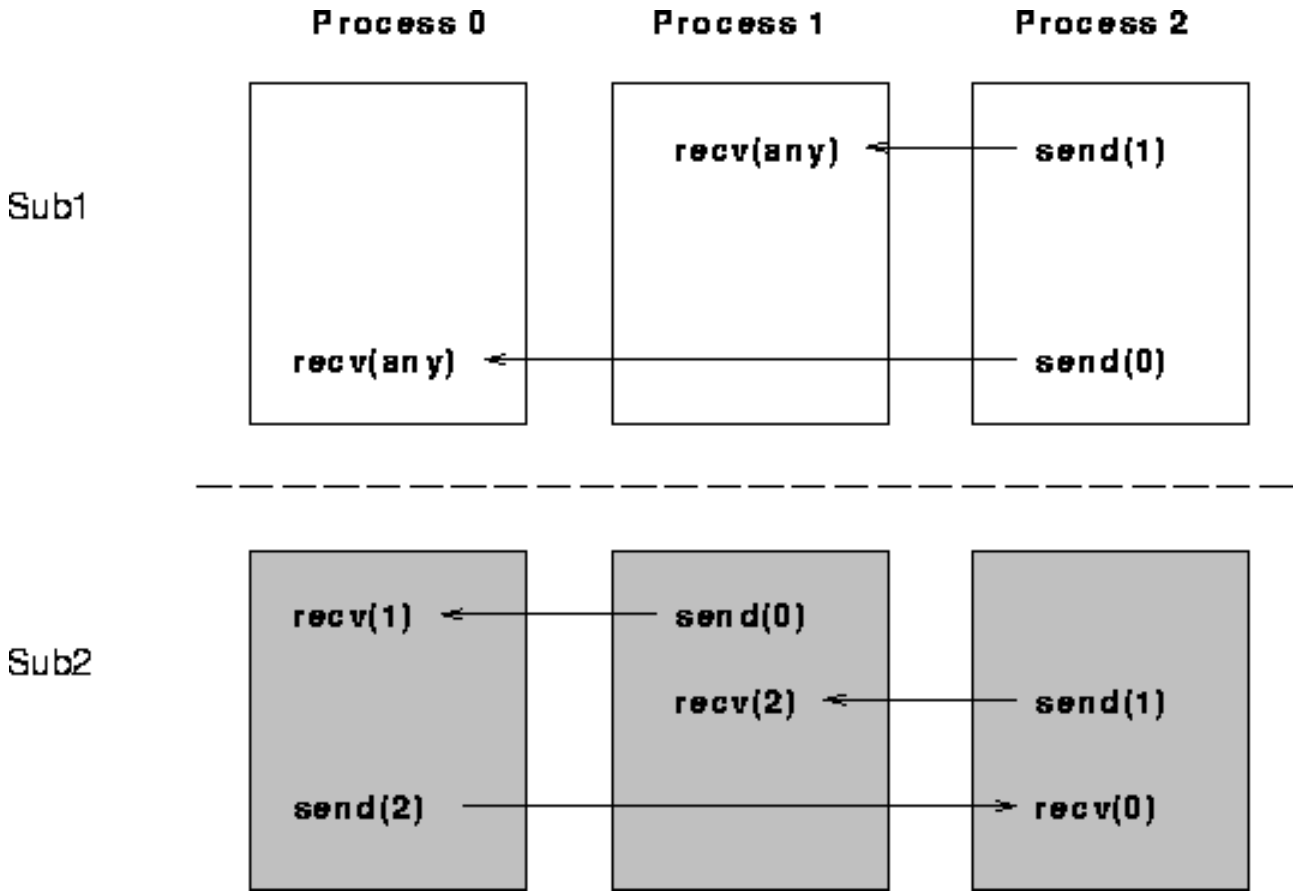
MPI Buffer

- Στο MPI ο buffer ορίζεται από starting address, datatype, και count.
- Datatype =
  - elementary (όλοι τα primitive datatypes της C ή Fortran).
  - πίνακας από datatypes.
  - Blocks από datatypes.
  - Δεικτοδοτημένο (indexed) array με blocks από datatypes.
  - Γενική δομή.
- Τα Datatypes δημιουργούνται αναδρομικά.
- Οι ορισμοί των βασικών datatypes επιτρέπουν ετερογενή επικοινωνία.

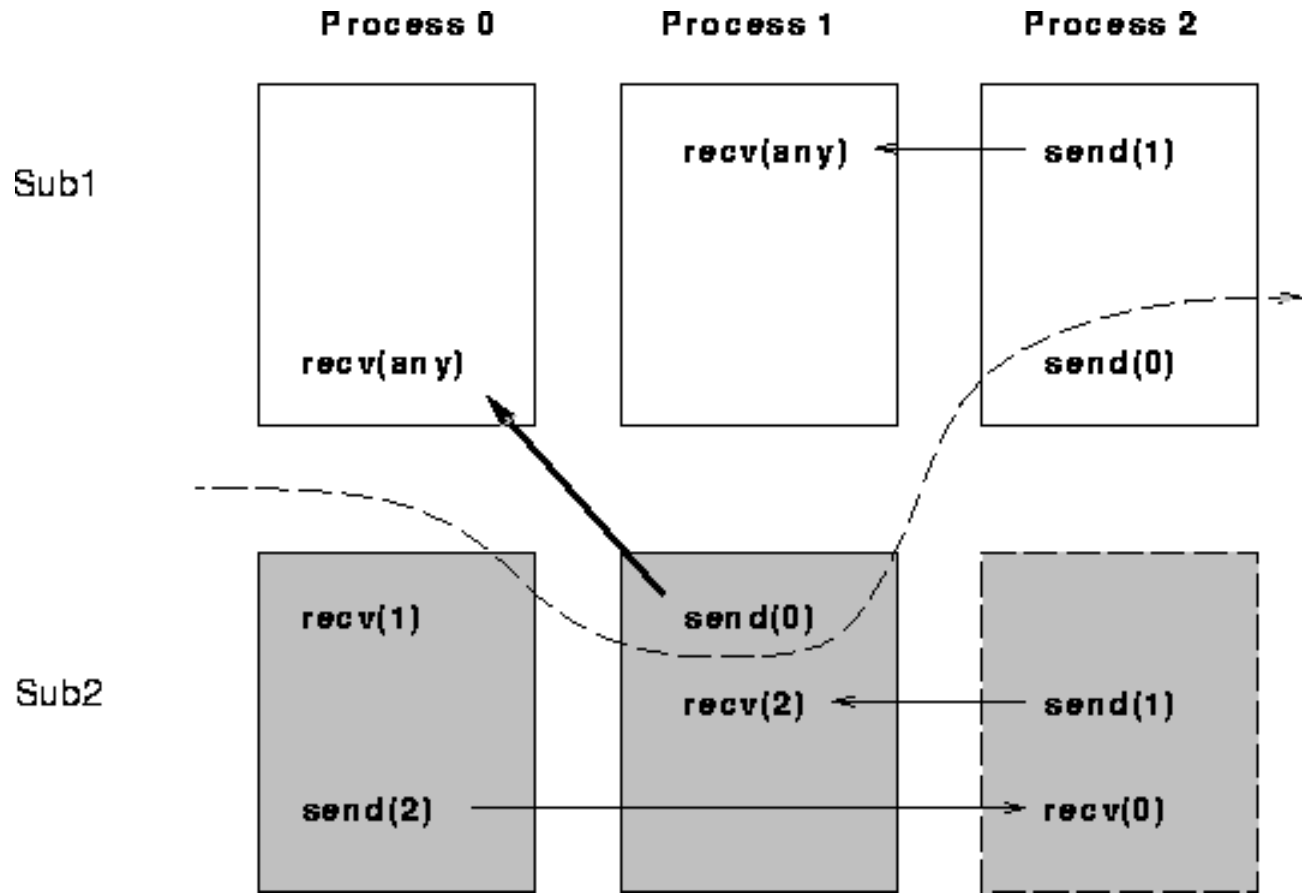
Τύποι δεδομένων

- Ο απλός τύπος είναι πολύ περιοριστικός.
- Συχνά χρησιμοποιούμε overloading για να επιτύχουμε την απαραίτητη ευελιξία.
- Προβλήματα:
  - κάτω από τον έλεγχο του χρήστη.
  - επιτρέπονται wildcards (`MPI_ANY_TAG`).
  - συγκρούσεις χρήσης της βιβλιοθήκης με άλλους χρήστες ή άλλες βιβλιοθήκες.

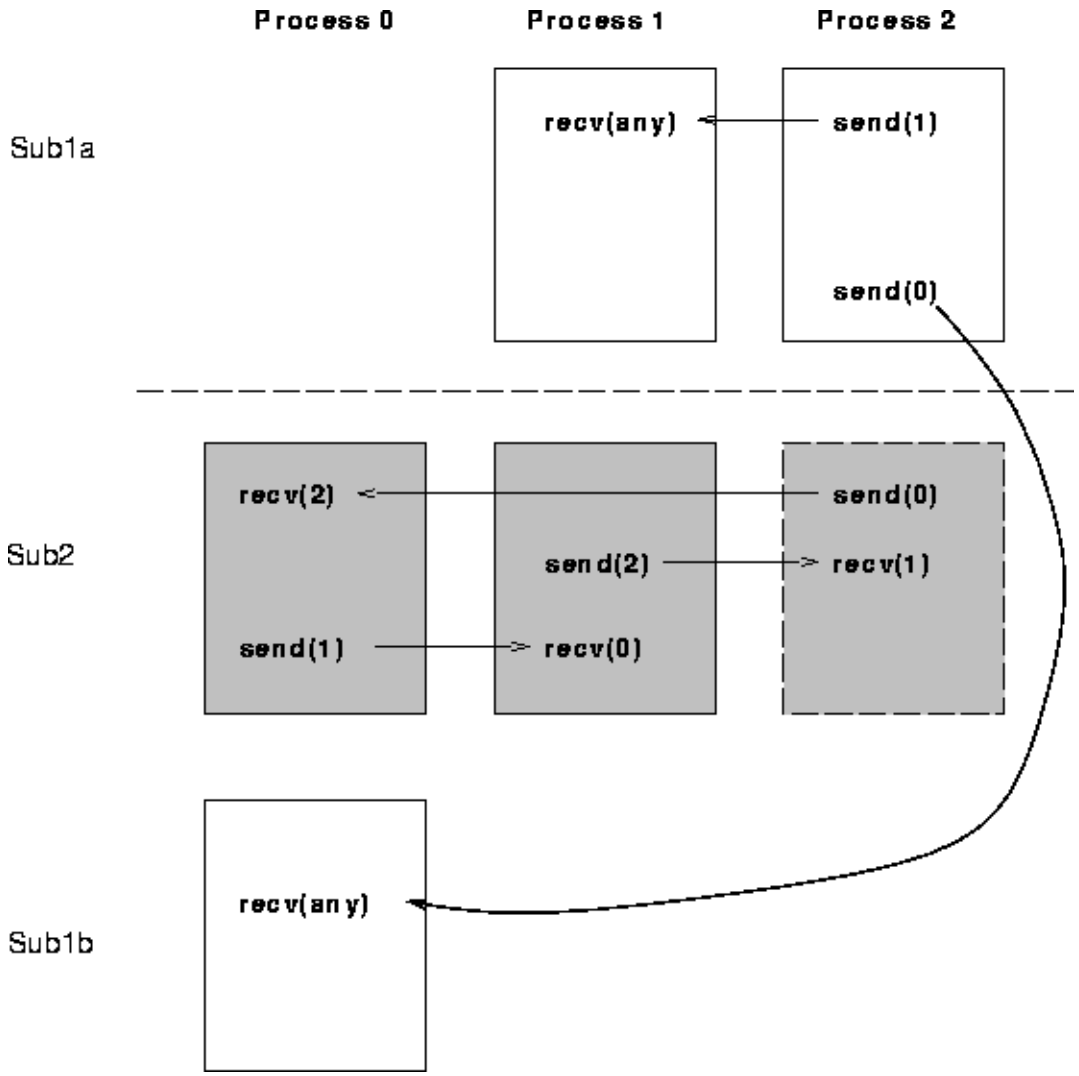
Ορθή επικοινωνία



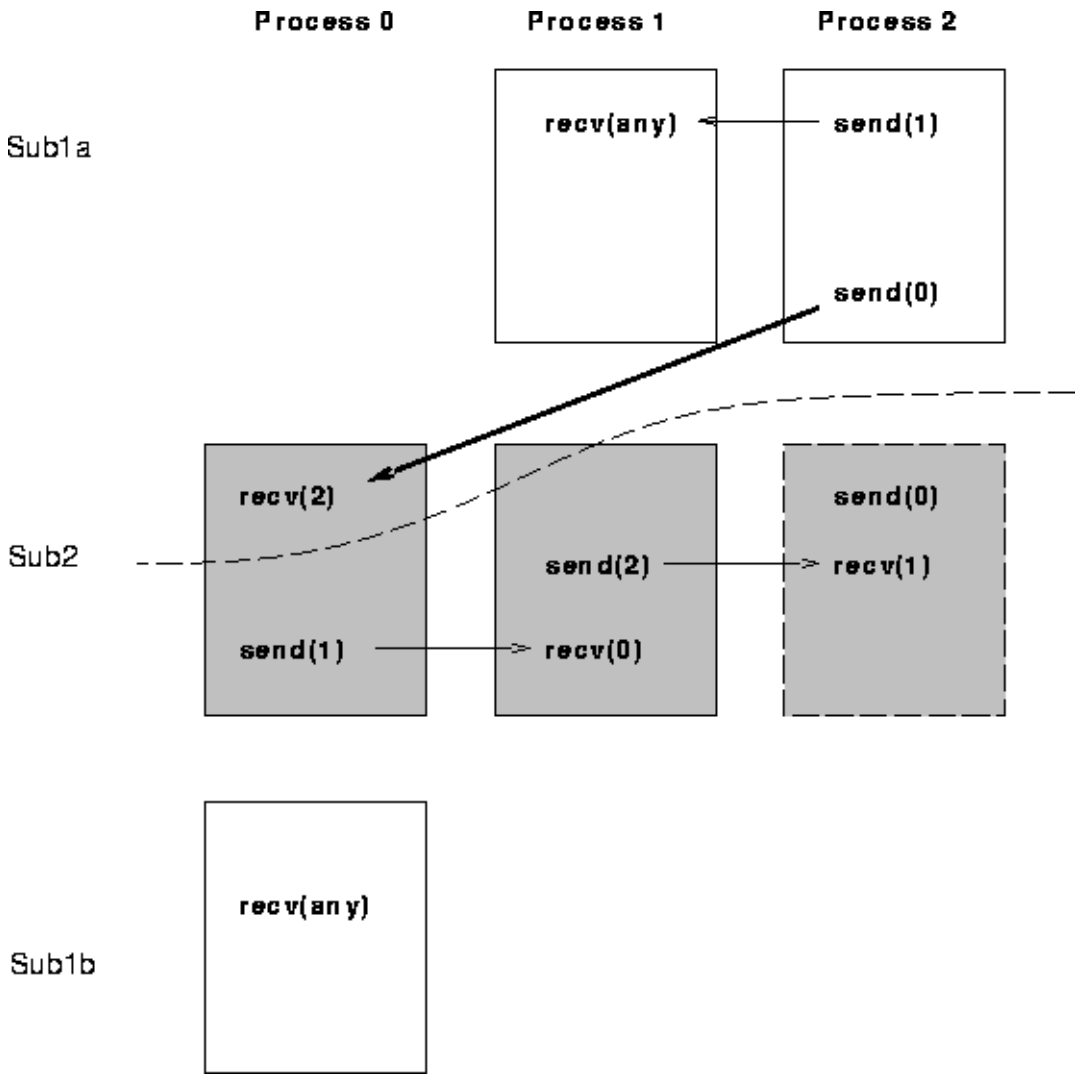
Εσφαλμένη επικοινωνία



Επικοινωνία με αναμονή



Εσφαλμένη επικοινωνία με αναμονή



Λύση στο πρόβλημα του τύπου

- Διαφορετικό context επικοινωνίας για κάθε οικογένεια μηνυμάτων. Χρησιμοποιείται για την τοποθέτηση μηνυμάτων στην ουρά (queueing) και το ταίριασμα (matching) των send και receive.
- Απαγόρευση wildcards, για ασφάλεια.
- Οι τύποι (tags, στο MPI) διατηρούνται για κανονική χρήση (wildcards OK).

Περιορισμοί στην επικοινωνία

- Ξεχωριστά γκρουπ διεργασιών δουλεύουν σε ξεχωριστά υπο-προβλήματα.
- Επιθυμητές οι τοπικές μεταβλητές στις διεργασίες.
- Ο κοινός χώρος διευθύνσεων δυσκολεύει το modularity.
- Τα μηνύματα της εφαρμογής διαχωρίζονται από τα εσωτερικά μηνύματα της βιβλιοθήκης.
- Η γνώση των τύπων μηνυμάτων της βιβλιοθήκης είναι ανεπιθύμητη.
- Ο συγχρονισμός πριν και μετά από τις κλήσεις στις συναρτήσεις βιβλιοθήκης είναι ανεπιθύμητος.

MPI_Send

- Η βασική συνάρτηση αποστολής μηνυμάτων.
- **`int MPI_Send(const void *buf, int count, MPI_Datatype dt, int dst, int tag, MPI_Comm comm)`**
  - `buf`: αρχική διεύθυνση του buffer που θα σταλεί.
  - `count`: πλήθος στοιχείων στον buffer.
  - `dt`: ο τύπος δεδομένων κάθε στοιχείου στον buffer.
  - `dst`: η τάξη (rank) της διεργασίας που θα λάβει τα δεδομένα.
  - `tag`: ακέραια ετικέτα μηνύματος.
  - `comm`: ο communicator.

MPI_Send (2)

- Επιστρέφεται `MPI_SUCCESS` αν η αποστολή ήταν επιτυχής, ή ακέραιος κωδικός σφάλματος.
- Η `MPI_Send` είναι blocking συνάρτηση.
- Η διεργασία θα μπλοκαριστεί μέχρι να ολοκληρωθεί η αποστολή (και η λήψη από την άλλη πλευρά).
- Είναι thread-safe: Μπορεί να κληθεί με ασφάλεια από πολλαπλά threads.

MPI_Recv

- Η βασική συνάρτηση παραλαβής μηνυμάτων:
- **`int MPI_Recv(void * buf, int count, MPI_Datatype dt, int source, int tag, MPI_Comm comm, MPI_Status * status)`**
  - `buf`: αρχική διεύθυνση του buffer που θα ληφθεί.
  - `count`: το *μέγιστο* πλήθος στοιχείων που θα ληφθούν.
    - Το πραγματικό πλήθος στοιχείων λαμβάνεται με την `MPI_Get_count`.
  - `dt`: ο τύπος δεδομένων κάθε στοιχείου στον buffer.
  - `source`: η τάξη (rank) της διεργασίας που έστειλε τα δεδομένα.
  - `tag`: ακέραια ετικέτα μηνύματος.
  - `comm`: ο communicator.
  - `status`: αντικείμενο κατάστασης.

MPI_Recv (2)

- Επιστρέφεται `MPI_SUCCESS` αν η λήψη ήταν επιτυχής, ή ακέραιος κωδικός σφάλματος.
- Η `MPI_Recv` είναι blocking συνάρτηση.
- Η διεργασία θα μπλοκαριστεί μέχρι να ολοκληρωθεί η λήψη των δεδομένων.
- Είναι thread-safe: Μπορεί να κληθεί με ασφάλεια από πολλαπλά threads.

MPI Message Tags (1)

- Τα tags μηνυμάτων είναι προαιρετικά.
- Μπορούν να λάβουν αυθαίρετες ακέραιες τιμές που φανερώνουν τη σημασιολογία των μηνυμάτων.
- Προσθέτουν πληροφορία σχετικά με τη φύση του μηνύματος.
- Σε περίπτωση που δύο διεργασίες P_0 και P_1 ανταλλάσσουν συχνά μηνύματα πολλαπλών τύπων, τα tags μπορούν να διαφοροποιήσουν τη φύση αυτών των μηνυμάτων.
- Διευκολύνεται έτσι η επεξεργασία στη λήψη.
  - Παράδειγμα: Αποστολή δύο διανυσμάτων από την P_0 στην P_1.
  - tag=1: Η P_1 πρέπει να προσθέσει τα διανύσματα.
  - tag=2: Η P_1 πρέπει να υπολογίσει το εσωτερικό τους γινόμενο.
  - tag=3: Η P_1 πρέπει να υπολογίσει το εξωτερικό τους γινόμενο.

MPI Message Tags (2)

- Γίνεται ταίριασμα των tags κατά τη λήψη.
- Μπορεί όμως να χρησιμοποιηθεί ως tag το wildcard `MPI_ANY_TAG`: Τότε ο παραλήπτης θα ταιριάξει αυθαίρετα tags.
- Ο παραλήπτης μπορεί να μάθε το tag που έβαλε σε ένα μήνυμα ο αποστολέας μέσω της συνάρτησης `MPI_Probe()`.

Η δομή MPI_Status

- Περιέχει πληροφορία για την κατάσταση του παραληφθέντος μηνύματος.

```
struct MPI_Struct {
    int MPI_SOURCE;
    int MPI_TAG;
    int MPI_ERROR;
    int cancelled;
    size_t count;
} MPI_Status;
```

- `MPI_Source`: Ο αποστολέας του μηνύματος.
- `MPI_TAG`: Το tag του μηνύματος.
- `MPI_ERROR`: Κωδικός σφάλματος σχετικός με το μήνυμα.
- `cancelled`: Ένδειξη ακύρωσης της άιτησης ή όχι.
- `count`: Πλήθος οντοτήτων που παρελήφθησαν.

Συλλογική επικοινωνία: MPI_Bcast

- Αποστολή μηνυμάτων από την διεργασία με τάξη root προς όλες τις άλλες διεργασίες του comm.
- **int MPI_Bcast(void *buffer, int count, MPI_Datatype dt, int root, MPI_Comm comm)**
  - buf: αρχική διεύθυνση του buffer.
  - count: πλήθος στοιχείων στον buffer.
  - dt: ο τύπος δεδομένων κάθε στοιχείου στον buffer.
  - root: η τάξη (rank) της διεργασίας που πραγματοποιεί την αποστολή.
  - comm: ο communicator.

Συλλογική επικοινωνία: MPI_Reduce

- Συνδυάζει δεδομένα από όλες τις διεργασίες και επιστρέφει το αποτέλεσμα σε μια μόνο διεργασία.
- **`int MPI_Reduce(const void *sendbuf, void *recvbuf, int count, MPI_Datatype dt, MPI_Op op, int root, MPI_Comm comm)`**
  - `sendbuf`: αρχική διεύθυνση του buffer που θα σταλεί.
  - `recvbuf`: αρχική διεύθυνση του buffer που θα παραληφθεί από τη διεργασία με τάξη `root`.
  - `count`: πλήθος στοιχείων στον buffer αποστολής.
  - `dt`: ο τύπος δεδομένων κάθε στοιχείου στον buffer.
  - `op`: η πράξη επί των δεδομένων.
  - `root`: η τάξη της διεργασίας που θα λάβει το αποτέλεσμα της εκτέλεσης της `op`.
  - `comm`: ο communicator.

Αλγόριθμος υπολογισμού π

- Υπάρχουν δεκάδες μέθοδοι στην επίσημη βιβλιογραφία για τον υπολογισμό του π.
- Μία εξ' αυτών:

$$\pi = \frac{1}{n}\sum_{i=1}^{n}\frac{4}{1+x_i^2}$$

- Όπου:
- n: η επιθυμητή ακρίβεια (το πλήθος των δεκαδικών ψηφίων του π).

$$x_i = \frac{2i-1}{2n} = \frac{i-0.5}{n}$$

Υπολογισμός του π με το MPI (1)

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

int main (int argc, char ** argv) {
   int done = 0, n, myid, numprocs, i, rc;
   double PI25DT = 3.141592653589793238462643;
   double mypi, pi, h, sum, x, a;

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

   while (!done) {
      // Embed the code of next slide
   }
   MPI_Finalize();
}
```

Υπολογισμός του π με το MPI (2)

```
if (myid == 0) {
   printf("Enter number of intervals: (0 quits)"); scanf("%d",&n);
}
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
if (n == 0) { break; }

h = 1.0 / (double) n;
sum = 0.0;

for (i = myid + 1; i <= n; i += numprocs) {
   x = h * ((double)i - 0.5);
   sum += 4.0 / (1.0 + x * x);
}
mypi = h * sum;
MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
if (myid == 0) {
   printf("pi is about %.16f, Error=%.16f\n", pi, fabs(pi-PI25DT));
}
```

Ανάθεση υπολογισμών στις διεργασίες

```
for (i = myid + 1; i <= n; i += numprocs) {
    x = h * ((double)i - 0.5);
    sum += 4.0 / (1.0 + x * x);
}
```

Για numprocs = 5:

myid=0	myid=1	myid=2	myid=3	myid=4
i=1	i=2	i=3	i=4	i=5
i=6	i=7	i=8	i=9	i=10
i=11	i=12	i=13	i=14	i=15
i=16	i=17	i=18	i=19	i=20

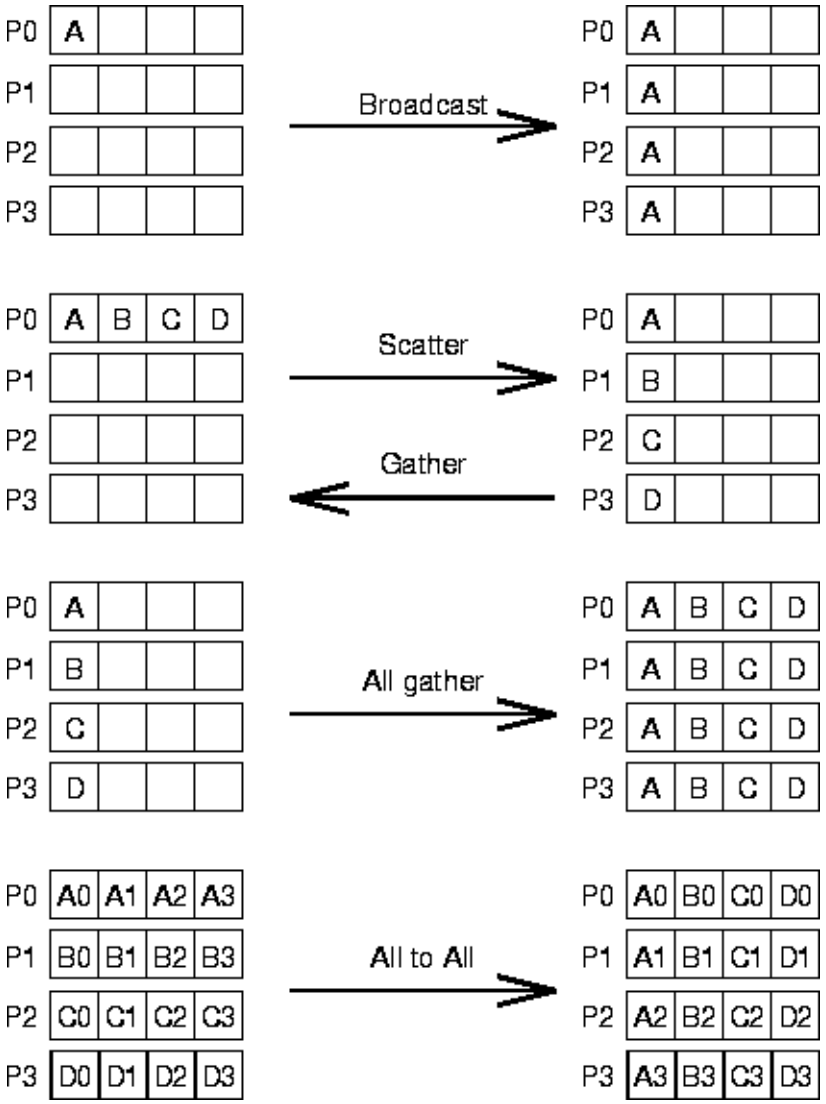
Συλλογικές επικοινωνίες στο MPI

- Η επικοινωνία γίνεται στο εσωτερικό μιας ομάδας διεργασιών.
- Τα γκρουπ φτιάχνονται χειροκίνητα. Χρήση συναρτήσεων MPI group-manipulation ή ρουτίνες MPI topology-definition.
- Δε χρησιμοποιούνται message tags. Αντ' αυτού γίνεται χρήση διαφορετικών communicators.
- Καμία συλλογική επικοινωνία non-blocking.
- Τρία είδη συλλογικής επικοινωνίας:
  - Συγχρονισμός.
  - Μετακίνηση δεδομένων.
  - Συλλογικές πράξεις.

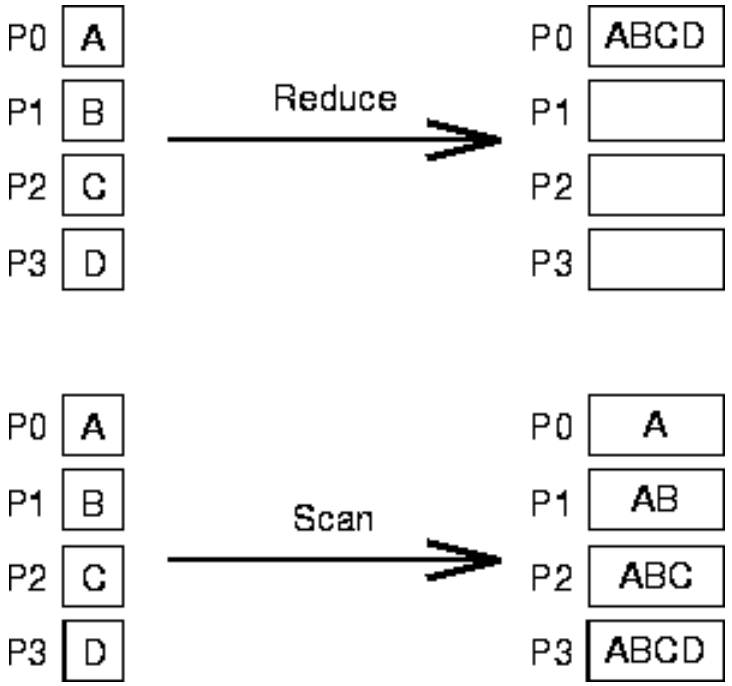
Συγχρονισμός: `MPI_Barrier`

- Η συνάρτηση αυτή μπλοκάρει τη διεργασία που την καλεί μέχρι να την καλέσουν όλες οι διεργασίες που ανήκουν στον communicator `comm`.
- `int MPI_Barrier (MPI_Comm comm)`

Τύποι συλλογικής επικοινωνίας



Τύποι συλλογικών υπολογισμών



Συλλογικές συναρτήσεις MPI_*

- Εδώ: * wildcard = αντικαθίσταται από οποιοδήποτε string.

Allgather	Allgatherv	Allreduce
Alltoall	Alltoallv	Bcast
Gather	Gatherv	Reduce
ReducedScatter	Scan	Scatter
Scatterv		

Συλλογικές συναρτήσεις MPI_* (2)

- Οι συναρτήσεις `MPI_All*` δίνουν δεδομένα σε όλες τις διεργασίες.
- Οι συναρτήσεις `MPI_*v` επιτρέπουν διαφορετικού μεγέθους δεδομένα.
- Οι συναρτήσεις `MPI_Allreduce`, `MPI_Reduce`, `MPI_ReduceScatter`, και `MPI_Scan` παίρνουν τόσο built-in όσο και user-defined συναρτήσεις συνδυασμού.

Συλλογικές συναρτήσεις υπολογισμού

MPI Name	Λειτουργία	MPI name	Λειτουργία
MPI_MAX	maximum	MPI_LXOR	Λογικό XOR
MPI_MIN	minimum	MPI_BAND	Bitwise AND
MPI_PROD	γινόμενο	MPI_BOR	Bitwise OR
MPI_SUM	άθροισμα	MPI_BXOR	Bitwise XOR
MPI_LAND	Λογικό AND	MPI_MAXLOC	Max & Θέση
MPI_LOR	Λογικό OR	MPI_MINLOC	Min & Θέση

User-defined συλλογικές πράξεις

- `int MPI_Op_create(MPI_User_function * ufn, int commute, MPI_Op * op)`
  - `ufn(invec, inoutvec, len, datatype)`
  - Η συνάρτηση ufn εκτελεί:

```
for i from 0 to len - 1
  inoutvec[i] = invec[i] op inoutvec[i];
```

- `int MPI_Op_free(MPI_Op * op)`