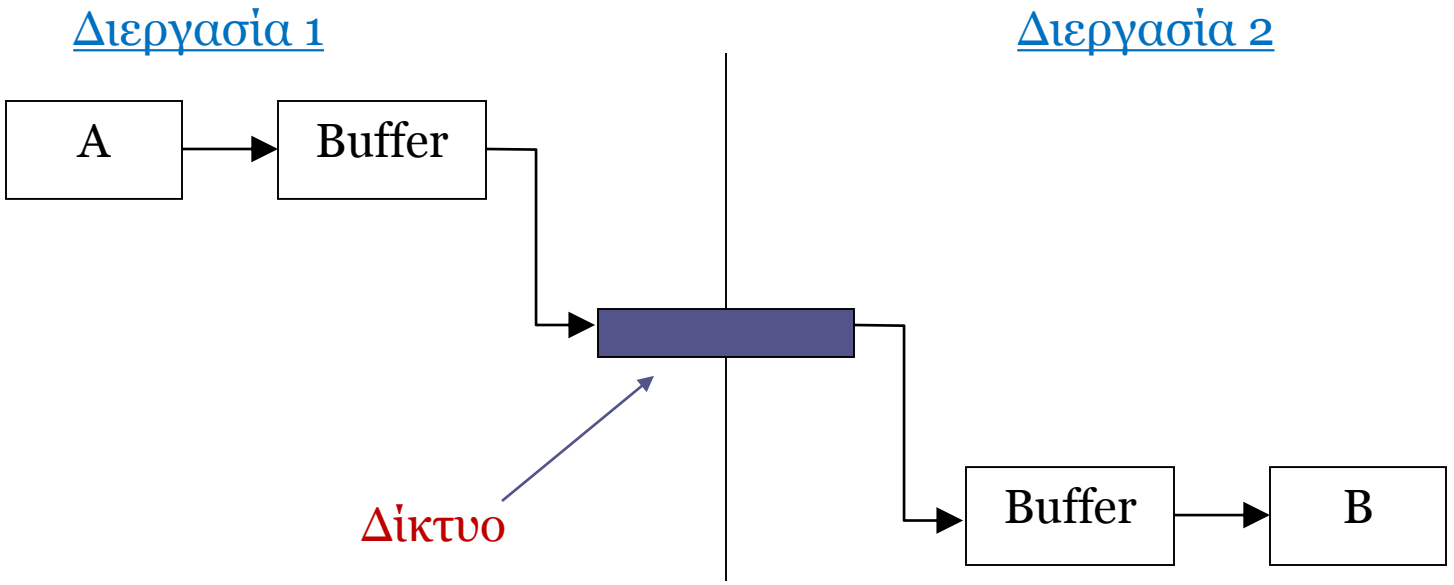
Προηγμένες αρχιτεκτονικές υπολογιστών και προγραμματισμός παραλλήλων συστημάτων

LAB-09. MPI (Μέρος Ε') – Non-blocking επικοινωνίες

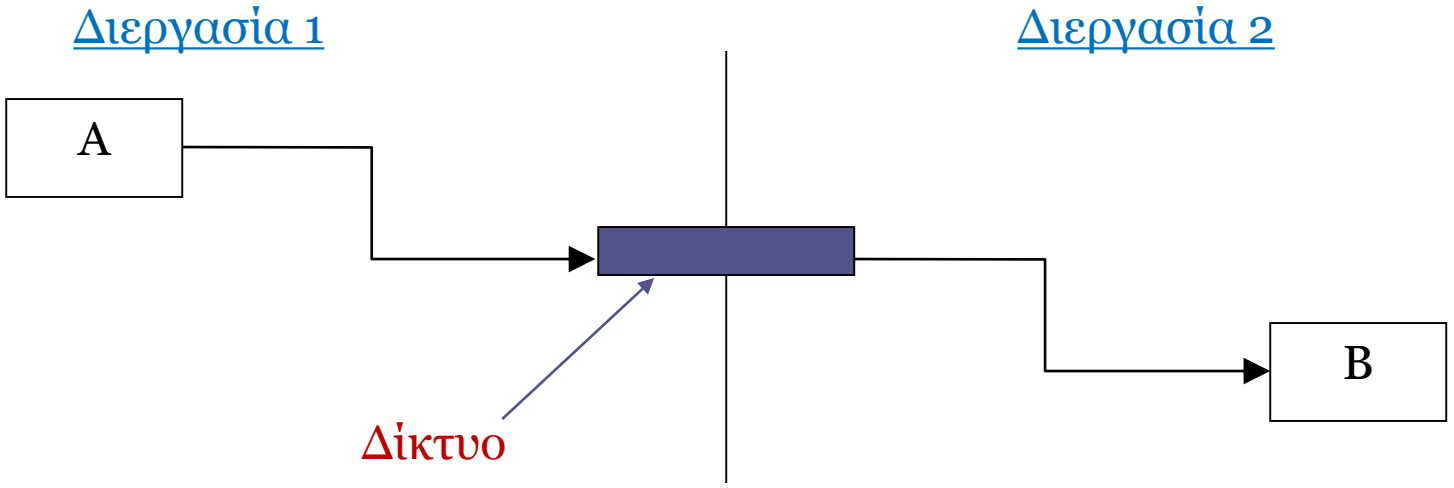
Η χρήση buffer

- Πώς μεταδίδονται τα δεδομένα μετά την send;



Χωρίς buffer

- Δύο επιλογές:
  - Το send δεν επιστρέφει αν δεν παραδοθούν τα δεδομένα.
  - Το send επιστρέφει πριν παραδοθούν τα δεδομένα. Έλεγχος παράδοσης αργότερα.



Blocking/Non-blocking επικοινωνίες

- Blocking επικοινωνίες: Η διεργασία μπλοκάρεται μέχρι να ολοκληρωθεί η αντίστοιχη λειτουργία.
  - `MPI_Send`: δεν ολοκληρώνεται μέχρι ο buffer να αδειάσει (έτοιμος για να ξαναχρησιμοποιηθεί).
  - `MPI_Recv`: δεν ολοκληρώνεται μέχρι ο buffer να γεμίσει (έτοιμος για να χρησιμοποιηθεί).
- Non-blocking επικοινωνίες:
  - `MPI_Isend`
  - `MPI_Irecv`

Σκοπός non-blocking επικοινωνιών

- Παροχή ενός μηχανισμού εναλλαγής μεταξύ επικοινωνιών και χρήσιμων υπολογισμών.
  - Οι επικοινωνίες και οι υπολογισμοί μπορούν να πραγματοποιούνται ταυτόχρονα.
  - Αποκρύπτονται οι (όποιες) συνέπειες του (όποιου) latency.
- Αποφυγή deadlocks.
- Αποφυγή buffering και memory-to-memory αντιγραφών.
  - Βελτίωση επιδόσεων.

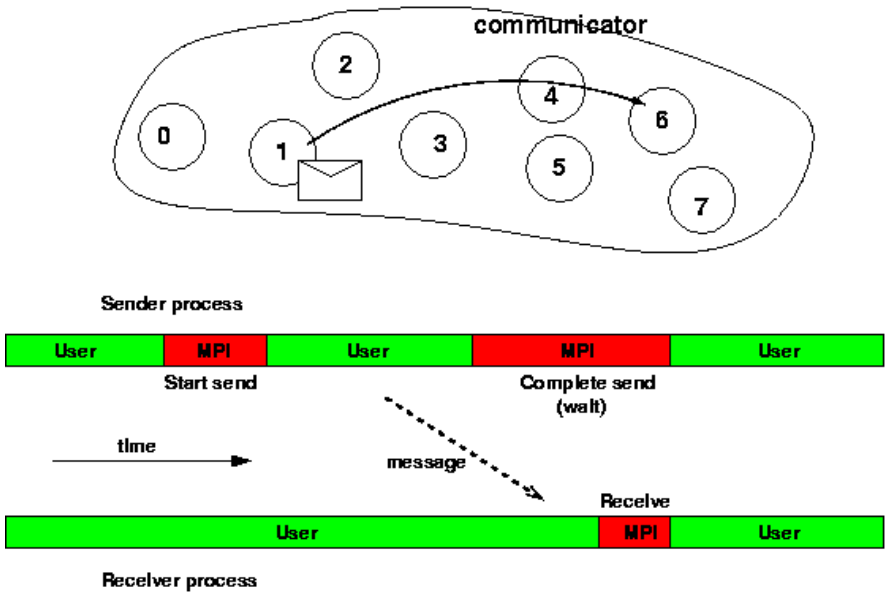
Non-blocking επικοινωνίες

- Non-blocking επικοινωνίες: Η διεργασία δεν μπλοκάρεται μέχρι να ολοκληρωθεί η αντίστοιχη λειτουργία.
- Η εκτέλεση του κώδικα συνεχίζεται κανονικά, χωρίς να έχει ολοκληρωθεί η κληθείσα λειτουργία.
- Οι non-blocking λειτουργίες επιστρέφουν αμέσως ένα αντικείμενο διαχείρισης αίτησης (request handle).
- Μια διεργασία μπορεί να κάνει ερώτηση (query) ή αναμονή (wait) σε ένα request κάνοντας χρήση του αντίστοιχου handle.

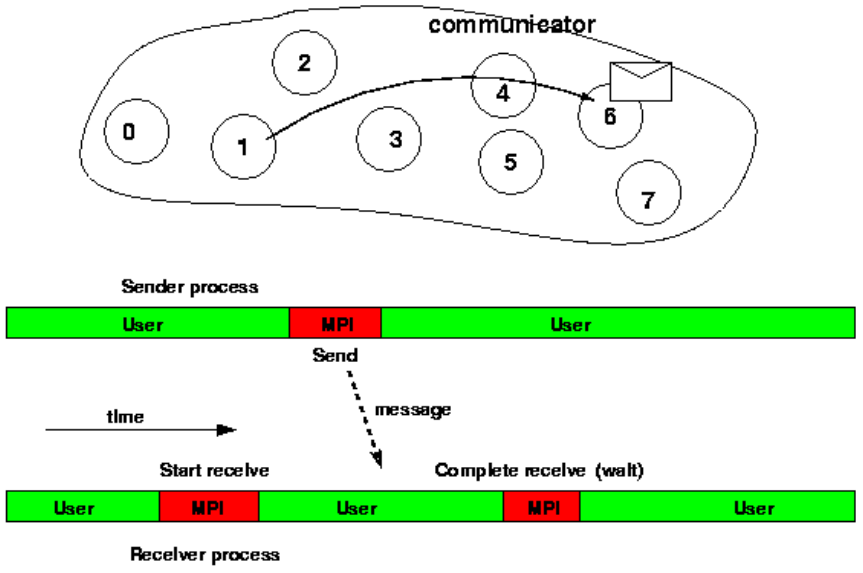
Δομή non-blocking επικοινωνιών (1)

1. Useful work (Program Execution)
2. Non-blocking operation (MPI_Isend, MPI_Irecv,…)
3. Useful work (Program Execution)
4. Complete communication call (MPI_Wait, MPI_Test, …)
5. Useful work (Program Execution)

Δομή non-blocking επικοινωνιών (2)



Non-blocking send



Non-blocking receive

MPI_Request (Request Handle)

- Αντικείμενο-διαχειριστής non-blocking αιτήσεων.
- Μπορεί να χρησιμοποιηθεί από τις συναρτήσεις `wait` και `test` για ενημέρωση της ολοκλήρωσης της non-blocking λειτουργίας.
  - `MPI_Wait, MPI_Waitall, MPI_Waitany, MPI_Waitsome`
  - `MPI_Test, MPI_Testall, MPI_Testany, MPI_Testsome`

MPI_Isend

- Εκκίνηση μιας non-blocking αποστολής μηνύματος.
- **`int MPI_Isend(const void *buf, int count, MPI_Datatype dt, int dst, int tag, MPI_Comm comm, MPI_Request *request)`**
  - `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.
  - `request` (output): Ο διαχειριστής του request.

MPI_Irecv

- Εκκίνηση μιας non-blocking λήψης μηνύματος.
- **`int MPI_Irecv(void *buf, int count, MPI_Datatype dt, int source, int tag, MPI_Comm comm, MPI_Request * request)`**
  - `buf`: αρχική διεύθυνση του buffer που θα ληφθεί.
  - `count`: το πλήθος στοιχείων στον receive buffer.
  - `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.
  - `request` (output): Ο διαχειριστής του request.

MPI_Wait

- Αναμονή ολοκλήρωσης μιας αίτησης MPI.
- **`int MPI_Wait(MPI_Request * request, MPI_Status * status)`**
  - `request` (input): Ο διαχειριστής του request.
  - `status` (output): Αντικείμενο κατάστασης της αντίστοιχης λειτουργίας υπό αναμονή.

MPI_Test

- Έλεγχος ολοκλήρωσης μιας αίτησης MPI.
- **`int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)`**
  - `request` (input): Ο διαχειριστής του request.
  - `flag` (output): Τίθεται ίση με true αν η αίτηση έχει ολοκληρωθεί.
  - `status` (output): Αντικείμενο κατάστασης της αντίστοιχης λειτουργίας υπό έλεγχο.

Παράδειγμα

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

int main (int argc, char ** argv) {
   /// Number of processes, Rank of current process
   int NumProcs, ProcRank;

   /// Request handle
   MPI_Request request;

   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &NumProcs);
   MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);

   if (NumProcs != 2) {
      printf("This appl is meant to be run with 2 processes.\n");
      MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
   }
```

```c
if (ProcRank == 0) {
  int buffer_sent = 12345;

  printf("Process %d sends value %d.\n", ProcRank, buffer_sent);
  MPI_Isend(&buffer_sent,1,MPI_INT,1,0,MPI_COMM_WORLD, &request);

  printf("Process %d is doing useful work here...\n", ProcRank);
  ///  Some useful code here

  MPI_Wait(&request, MPI_STATUS_IGNORE);

} else if (ProcRank == 1) {
  int received;

  printf("Process %d starts receiving data...\n", ProcRank);
  MPI_Irecv(&received,1, MPI_INT, 0, 0, MPI_COMM_WORLD, &request);

  printf("Process %d is doing useful work here...\n", ProcRank);
  ///  Useful code here (received buffer cannot be accessed yet!!)

  MPI_Wait(&request, MPI_STATUS_IGNORE);
  printf("Process %d received value %d.\n", ProcRank, received);
}
```

Πολλαπλές αναμονές

- Είναι συχνά επιθυμητό να περιμένουμε την ολοκλήρωση πολλών request.
- Για παράδειγμα, σε ένα κατανεμημένο περιβάλλον με ύπαρξη master διεργασίας, ο master περιμένει ένα ή περισσότερους slaves να του στείλουν μήνυμα.
- Το MPI διαθέτει συναρτήσεις υποστήριξης τέτοιων πολλαπλών αναμονών.

MPI_Waitall

- Δέχεται πίνακα από request handles και αναμένει την ολοκλήρωση **όλων** των αντίστοιχων λειτουργιών.
- **`int MPI_Waitall(int count, MPI_Request requests[], MPI_Status statuses[])`**
  - `count`: Πλήθος των λειτουργιών των οποίων την ολοκλήρωση αναμένουμε.
  - `requests[]`: Πίνακας μεγέθους `count` που περιέχει τους request handles των λειτουργιών υπό αναμονή.
  - `statuses[]`: Πίνακας αντικειμένων status μεγέθους `count`.

MPI_Waitany

- Δέχεται πίνακα από request handles και αναμένει την ολοκλήρωση **μίας οποιασδήποτε** αντίστοιχης λειτουργίας.
- **`int MPI_Waitany(int count, MPI_Request requests[], int *idx, MPI_Status *status)`**
  - `count`: Πλήθος των λειτουργιών των οποίων την ολοκλήρωση αναμένουμε.
  - `requests[]`: Πίνακας μεγέθους `count` που περιέχει τους request handles των λειτουργιών υπό αναμονή.
  - `idx`: Δείκτης στον πίνακα requests που δείχνει ποια λειτουργία ολοκληρώθηκε. Τιμές [0, `count` – 1].
  - `status`: Αντικείμενο status της λειτουργίας.

MPI_Waitsome

- Δέχεται πίνακα από request handles και αναμένει την ολοκλήρωση **κάποιων** αντίστοιχων λειτουργιών.
- **`int MPI_Waitsome(int incount, MPI_Request requests[], int *outcount, int indices[], MPI_Status statuses[])`**
  - `incount`: Πλήθος των λειτουργιών των οποίων την ολοκλήρωση αναμένουμε.
  - `requests[]`: Πίνακας μεγέθους `incount` που περιέχει τους request handles των λειτουργιών υπό αναμονή.
  - `outcount`: Πλήθος των λειτουργιών που ολοκληρώθηκαν.
  - `indices[]`: Πίνακας μεγέθους `outcount` με τους δείκτες στον πίνακα requests των ολοκληρωμένων λειτουργιών.
  - `statuses[]`: Πίνακας αντικειμένων status.

Ταχείες επικοινωνίες

- Η αναμονή λήψης μεγάλου αριθμού μηνυμάτων με blocking τρόπο μπορεί να οδηγήσει στο μπλοκάρισμα μιας διεργασίας για μεγάλο χρονικό διάστημα.
- Ειδικά όταν η μετάδοση γίνεται πάνω σε μη αξιόπιστο δίκτυο.
- Στο παράδειγμα που ακολουθεί, η διεργασία με rank 0 αναμένει μπλοκαρισμένη την παράδοση 100 μηνυμάτων από κάθε άλλη διεργασία.

Παράδειγμα

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

int main (int argc, char ** argv) {
    int rank, size, i, buf[1];
    MPI_Status status;

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

    if (rank == 0) {
       for (i = 1; i < size; i++) {
          for (j = 0;  j < 100; j++) {
             MPI_Recv(buf,1, MPI_INT, i, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
             printf("Msg from %d tag %d\n",status.MPI_SOURCE, status.MPI_TAG);
          }
       }
    } else {
       for (i = 0; i < 100; i++)
          MPI_Send (buf, 1, MPI_INT, 0, i, MPI_COMM_WORLD);
    }
    MPI_Finalize();
}
```

Ταχείες επικοινωνίες

- Αν καθυστερήσει η παράδοση ενός μηνύματος, καθυστερεί και η παραλαβή όλων των επόμενων, ακόμα και αν δεν υπάρχει κανένα πρόβλημα.
- Καθυστέρηση.
- Και πολλά επιπλέον προβλήματα:
  - Π.χ. να γεμίσει ο receive buffer και να αρχίσουν να απορρίπτονται μηνύματα.
- **Λύση**: άμεση (non-blocking) παραλαβή όσων μηνυμάτων έχουν παραδοθεί.
  - Αναμονή με την `MPI_Waitsome`.

Λύση (Process 0)

```
#define large 128
MPI_Request requests[large];
MPI_Status stats[large];
int indices[large];
int buf[large];

for (i = 1; i < size; i++) {
  MPI_Irecv(buf+i,1,MPI_INT, i, MPI_ANY_TAG, MPI_COMM_WORLD, &requests[i-1]);
}

while (not done) {
  MPI_Waitsome(size - 1, requests, &ndone, indices, stats);
  for (i = 0; i < ndone; i++) {
    j = indices[i];
    printf("Msg from %d, tag %d\n", stats[i].MPI_SOURCE, stats[i].MPI_TAG);
    MPI_Irecv(buf+j,1,MPI_INT,j, MPI_ANY_TAG, MPI_COMM_WORLD, &requests[j]);
  }
}
```

Τρόποι επικοινωνιών

- Το MPI προσφέρει πολλούς τρόπους αποστολής μηνυμάτων:
- Σύγχρονος τρόπος (`MPI_Ssend`): Το send δεν ολοκληρώνεται μέχρι να εκκινήσει ένα ταιριαστό receive.
- Buffered τρόπος (`MPI_Bsend`): ο χρήστης ορίζει τον buffer για να χρησιμοποιηθεί από το σύστημα.
- Ready τρόπος (`MPI_Rsend`): ο χρήστης εγγυάται ότι ένα ταιριαστό receive έχει ήδη εκκινήσει.
  - Επιτρέπει γρήγορα πρωτόκολλα επικοινωνίας
  - Απροσδιόριστη συμπεριφορά αν το receive δεν έχει εκκινήσει
- Non-blocking εκδοχές:
  - `MPI_Issend`, `MPI_Irsend`, `MPI_Ibsend`
  - Το `MPI_Recv` λαμβάνει μηνύματα σταλμένα με οποιοδήποτε τρόπο send.

MPI_Ssend, MPI_Issend

- Εκκινεί μία σύγχρονη και blocking αποστολή μηνύματος. Δεν ολοκληρώνεται μέχρι να εκκινήσει ένα ταιριαστό receive.
- **int MPI_Ssend(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)**
- Non-blocking σύγχρονη αποστολή μηνύματος:
- **int MPI_Issend(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request * request)**

MPI_Rsend, MPI_Irsend

- Εκκινεί μία ready και blocking αποστολή μηνύματος. Ο χρήστης εγγυάται ότι ένα ταιριαστό receive έχει ήδη εκκινήσει.
- **int MPI_Rsend(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)**
- Non-blocking ready αποστολή μηνύματος:
- **int MPI_Irsend(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request * request)**

MPI_Bsend (1)

- Εκκινεί μία buffered και blocking αποστολή μηνύματος. Ο χρήστης ορίζει τον buffer για να χρησιμοποιηθεί από το σύστημα.
- **`int MPI_Bsend(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_Bsend (2)

- Τοπική εκτέλεση: η ολοκλήρωση της εντολής δεν εξαρτάται από το αν έχει εκδοθεί ένα αντίστοιχο ταιριαστό receive.
- Αν ξεκινήσει ένα `Bsend` και δεν έχει εκδοθεί αντίστοιχο ταιριαστό receive από τον παραλήπτη η διεργασία πρέπει να βάλει το μήνυμα στον buffer.
- Ο χρήστης ορίζει ο ίδιος χώρο buffer με την `MPI_Buffer_attach`. Ο χώρος στον buffer δεν είναι διαθέσιμος για επόμενο `MPI_Bsend` εκτός αν ληφθεί το μήνυμα.
- Ανά πάσα χρονική στιγμή μπορεί να υπάρχει μόνο ένας buffer. Πρέπει να προβλεφθεί να είναι αρκετά μεγάλος ώστε να χωράει το ή τα μηνύματα που δεν έχουν ταιριαστό receive. Σε αντίθετη περίπτωση προκύπτει σφάλμα.

MPI_Bsend (3)

- Ο παρακάτω κώδικας δε δίνει αρκετό χώρο buffer. Χώρος buffer για ένα μόνο send. Το επόμενο MPI_Bsend μπορεί να ξεκινήσει πριν τελειώσει το πρώτο MPI_Bsend τη χρήση του buffer.

```
MPI_Buffer_attach(b, n * sizeof(double) + MPI_BSEND_OVERHEAD);
for (i=0; i<m; i++) {
   MPI_Bsend(buf, n, MPI_DOUBLE, ... );
}
```

Μπορούμε να εξαναγκάσουμε τα μηνύματα να παραδοθούν

```
MPI_Buffer_detach(&b, &n);
MPI_Buffer_attach(b, n);
```

Η `MPI_Buffer_detach` δε θα ολοκληρωθεί μέχρι όλα τα buffered μηνύματα να παραδοθούν.