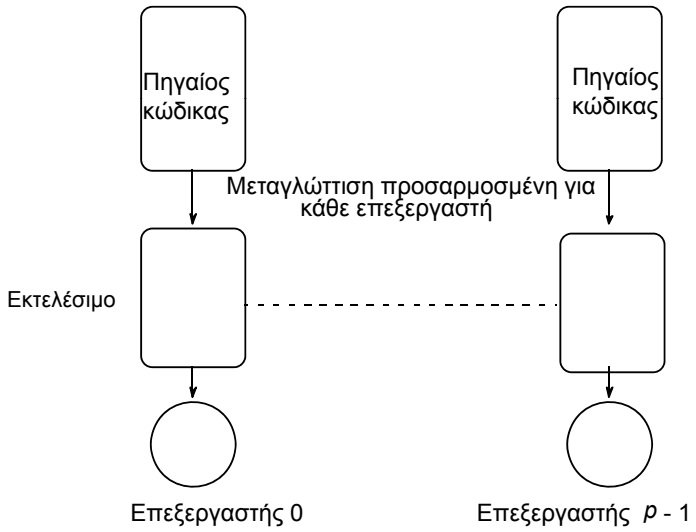
Υπολογισμός με βάση το πέρασμα μηνυμάτων

Προγραμματισμός με πέρασμα μηνυμάτων

Δύο μηχανισμοί απαιτούνται:

1. Μία μέθοδος για τη δημιουργία διεργασιών που θα εκτελούνται σε διαφορετικούς υπολογιστές.
2. Μία μέθοδος για την αποστολή και παραλαβή μηνυμάτων

Multiple program, multiple data (MPMD) model

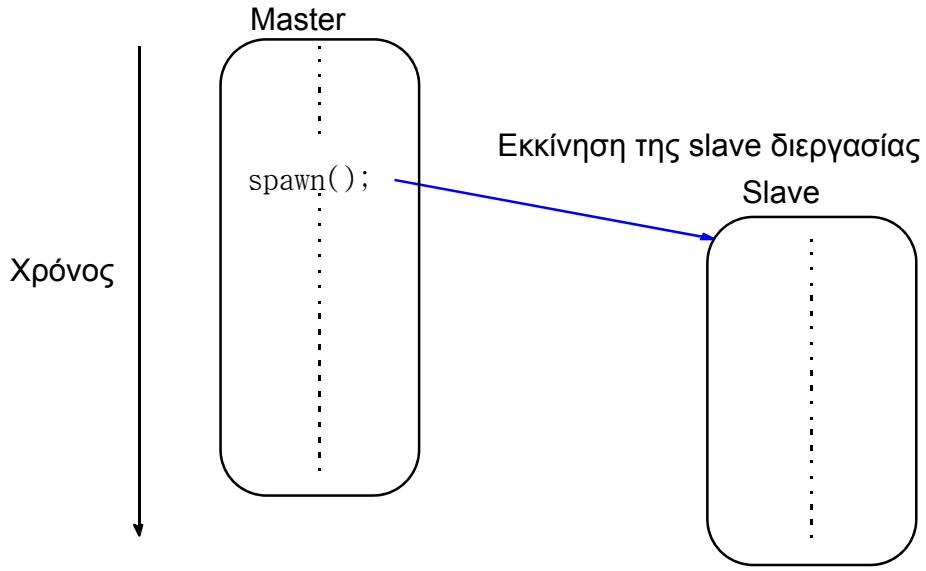


• Ξεχωριστός κώδικας γράφεται και μεταφράζεται για κάθε επεξεργαστή.

• Οι διεργασίες που εκτελούν τους διαφορετικούς κώδικες στους επεξεργαστές, μπορούν να δημιουργούνται δυναμικά από άλλες διεργασίες που ήδη τρέχουν σε άλλους επεξεργαστές

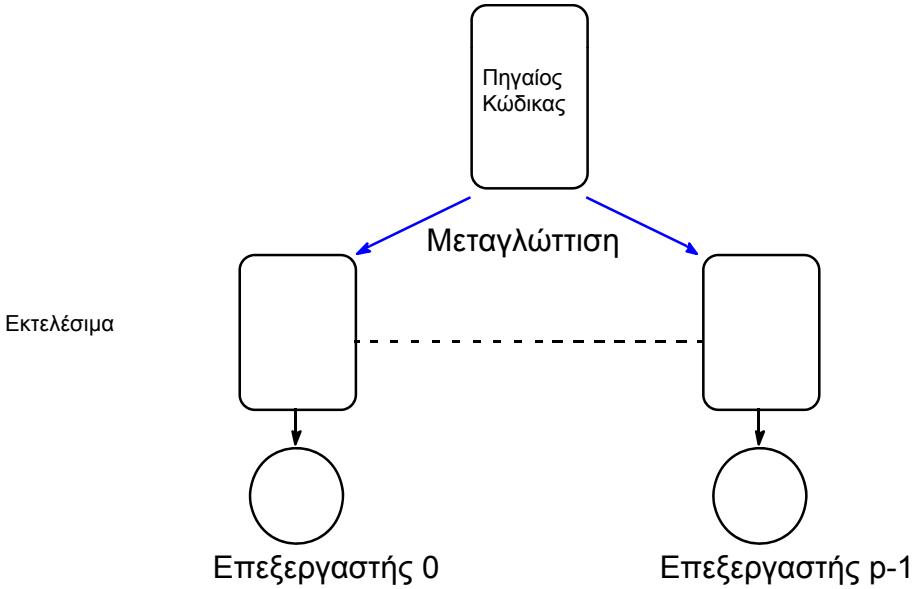
• Συνήθως οι διεργασίες έχουν σχέση master-slave. Μία διεργασία (master) αναλαμβάνει το συντονισμό όλου του υπολογισμού. Όλες οι υπόλοιπες διεργασίες (slaves) δημιουργούνται δυναμικά από τη master διαδικασία. Ουσιαστικά, οι slave διαδικασίες αναλαμβάνουν το κύριο όγκο επεξεργασίας και στέλνουν τα μερικά τους αποτελέσματα στη master διεργασία η οποία τα συνδυάζει και ενδεχομένως τα ξαναστέλνει στις slave διεργασίες.

Multiple Program Multiple Data (MPMD) Model



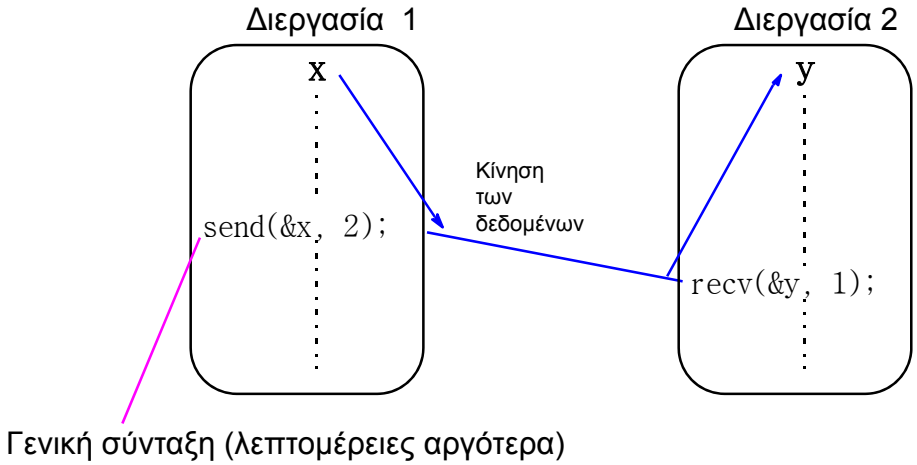
Single Program Multiple Data (SPMD) model

Όλες οι διεργασίες εκτελούν το ίδιο πρόγραμμα. Με τη χρήση όμως εντολών ελέγχου και ανάλογα το αναγνωριστικό της διαδικασίας, διαφορετικά τμήματα κώδικα εκτελούνται από τις διεργασίες. Όλα τα εκτελέσιμα ξεκινούν από την αρχή. Έχουμε δηλ. στατική δημιουργία διεργασιών όπου ο χρήστης θα πρέπει να έχει καθορίσει εξ' αρχής το πλήθος των διεργασιών (στατική δημιουργία διεργασιών)



Βασικές ρουτίνες αποστολής λήψης μεταξύ δύο επεξεργαστών

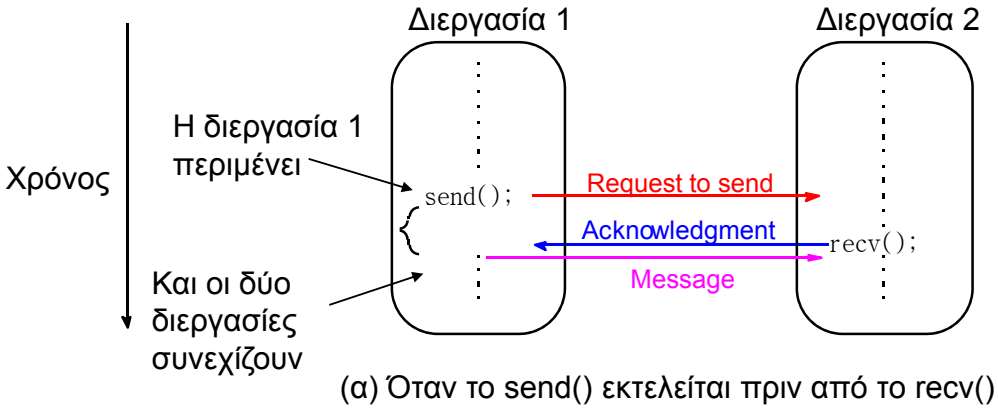
• Η αποστολή ενός μηνύματος μεταξύ δύο διεργασιών γινεται με τη κλήση των συναρτήσεων send() και recv() της βιβλιοθήκης του MPI.
• Στο συγκεκριμένο παράδειγμα, η τιμή της τοπικής μεταβλητής x της διεργασίας 1 στέλνεται στη διεργασία 2 και από θηκεύεται στη τοπική μεταβλητή της διεργασίας 2.



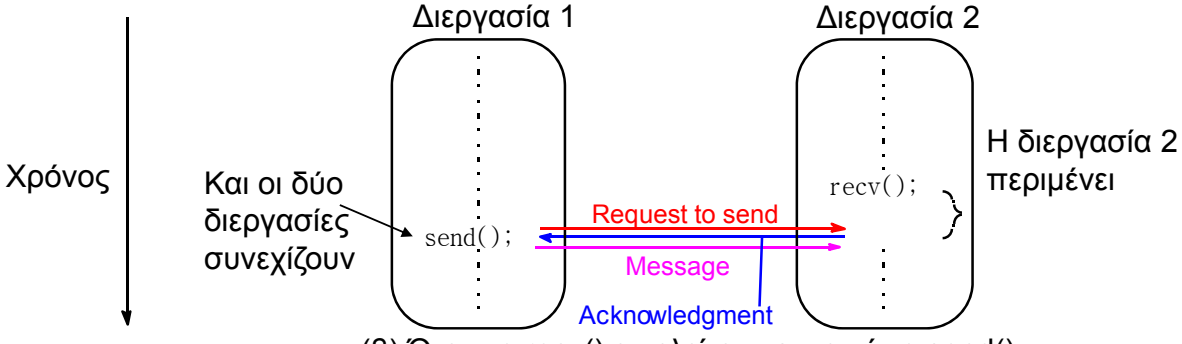
Σύγχρονο (Synchronous) πέρασμα μηνυμάτων

- Οι συναρτήσεις που υλοποιούν σύγχρονο πέρασμα μηνυμάτων, επιστρέφουν μόνο αφότου έχει ολοκληρωθεί η μεταφορά του μηνύματος
- *Σύγχρονη ρουτίνα αποστολής μηνύματος:* Αυτή η λειτουργία περιμένει μέχρι ολόκληρο το μήνυμα ληφθεί από τη διαδιακάσία παραλήπτη πριν να αρχίσει την αποστολή νέου μηνύματος
- *Synchronous receive routine*: Αυτή η λειτουργία περιμένει μέχρι το μήνυμα που αναμένει φθάσει
- Ουσιαστικά, οι σύγχρονες συναρτήσεις εκτελούν δύο ενέργειες: Μεταφέρουν δεδομένα και συγχρονίζουν τις διαδικασίες

Σύγχρονο send() and recv() χρησιμοποιώντας ένα πρωτόκολλο τριών σταδίων (3-way protocol)



(α) Όταν το send() εκτελείται πριν από το recv()



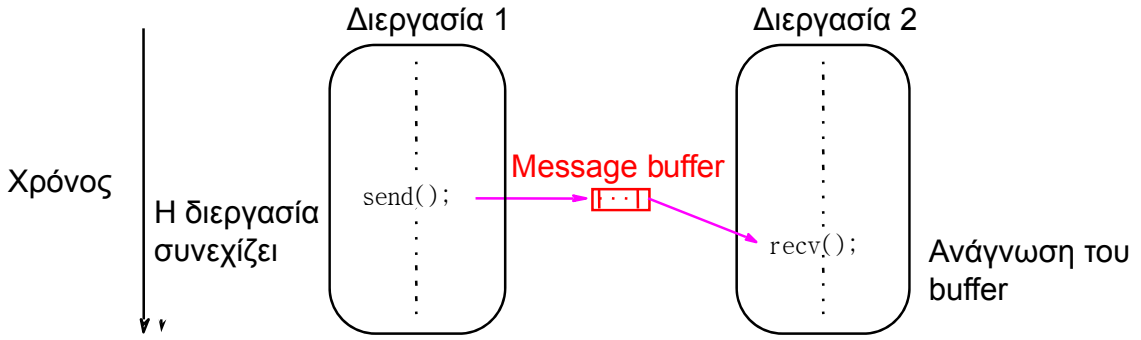
(β) Όταν το recv() εκτελείται πριν από το send()

Ασύγχρονο πέρασμα μηνυμάτων

- Οι συναρτήσεις αυτές επιστρέφουν πριν ολοκληρωθεί η λειτουργία επικοινωνίας που εκτελούν. Απαιτείται συνήθως τοπική αποθήκευση των μηνυμάτων που είναι σε εκκρεμότητα για αποστολή ή λήψη
- Γενικά, δεν συγχρονίζουν τις διεργασίες αλλά επιτρέπουν σε αυτές να προχωρούν στην εκτέλεση του προγράμματός τους νωρίτερα.
- Θα πρέπει όμως να χρησιμοποιούνται με προσοχή.
- Υπάρχουν δύο κατηγορίες ασύγχρονων λειτουργιών:
- Blocking – επιστρέφουν αφότου οι τοπικές τους ενέργειες ολοκληρωθούν αν και η μεταφορά των μηνυμάτων μπορεί να μην έχει ολοκληρωθεί εκείνη τη χρονική στιγμή.
- Non-blocking – επιστρέφουν κατευθείαν μετά την κλήση τους.Βασική υπόθεση είναι ότι πριν να ολοκληρωθεί η μεταφορά του μηνύματος, η τοπική μνήμη που διατίθεται για τη μεταφορά δεδομένων δεν τροποποιείται από τις εντολές που ακολουθούν. Είναι ευθύνη του προγραμματιστή να εξασφαλίσει ότι δεν θα συμβεί το αντίθετο

Πως οι ρουτίνες περάσματος μηνυμάτων επιστρέφουν πριν η μεταφορά μηνυμάτων ολοκληρωθεί

Προσωρινός αποθηκευτικός χώρος απαιτείται μεταξύ αποστολέα παραλήπτη για την αποθήκευση του μηνύματος:

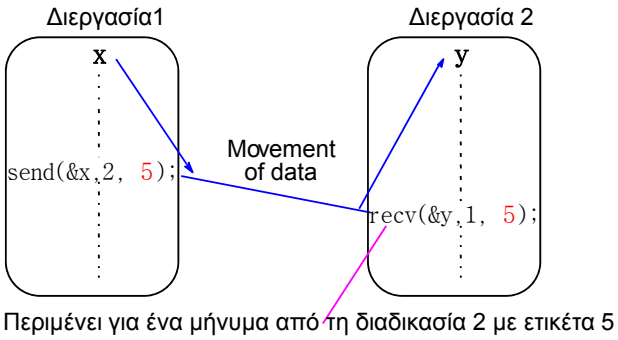


Οι ασύγχρονες Blocking συναρτήσεις μπορούν να «εκφυλιστούν» σε σύγχρονες ρουτίνες

- Μόλις οι τοπικές ενέργειες ολοκληρωθούν και η αποστολή του μηνύματος έχει αρχίσει, ο αποστολέας μπορεί να συνεχίσει στην εκτέλεση των επόμενων εντολών.
- Όμως, οι Buffers έχουν μόνο πεπερασμένο μέγεθος και έτσι μπορεί να προκύψει η κατάσταση όπου η διαδικασία αποστολής σταματά διότι όλος ο διαθέσιμος χώρος στους buffers έχει εξαντληθεί.
- Σε αυτή την περίπτωση, η διαδικασία αποστολής θα περιμένει μέχρι να ελευθερωθεί χώρος. Με άλλα λόγια η ρουτίνα συμπεριφέρεται ως μία σύγχρονη συνάρτηση.

Ετικέτα Μηνύματος

- Η ετικέτα μηνύματος χρησιμοποιείται για να διαφοροποιήσει μεταξύ των διαφορετικών τύπων δεδομένων που πρόκειται να σταλούν.
- Η ετικέτα μεταφέρεται με το μήνυμα.
- Αν δεν υπάρχει απαίτηση για συγκεκριμένο τύπο δεδομένων, μία ετικέτα μπαλαντέρ (wild card message tag) χρησιμοποιείται, έτσι ώστε το recv() να λάβει τα δεδομένα οποιουδήποτε send().
- Π.χ. για να στείλουμε τα δεδομένα που είναι αποθηκευμένα στη μεταβλητή x, με ετικέτα μηνύματος 5 από τη διαδικασία 1 στη διαδικασία 2 και καταχώρηση των ληφθέντων δεδομένων στη μεταβλητή y, θα έχουμε:

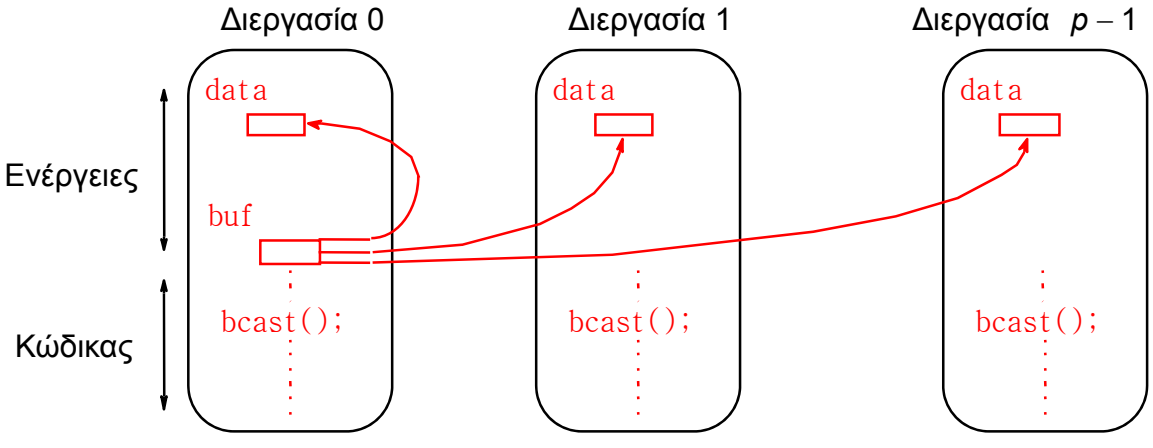


Ομαδικές ρουτίνες περάσματος μηνυμάτων

- Οι ρουτίνες αυτές στέλνουν μηνύματα σε μία ομάδα διεργασιών ή λαμβάνουν μηνύματα από μία ομάδα διεργασιών
- Θα μπορούσαν να υλοποιηθούν με μία σειρά από «σημείο-σε-σημείο» διαδικασιών.
- Συνήθως όμως, υψηλότερες επιδόσεις επιτυγχάνονται όταν υλοποιούνται κατευθείαν χωρίς την εκτέλεση πολλαπλών «σημείο-σε-σημείο» διαδικασιών

Εκπομπή - Broadcast

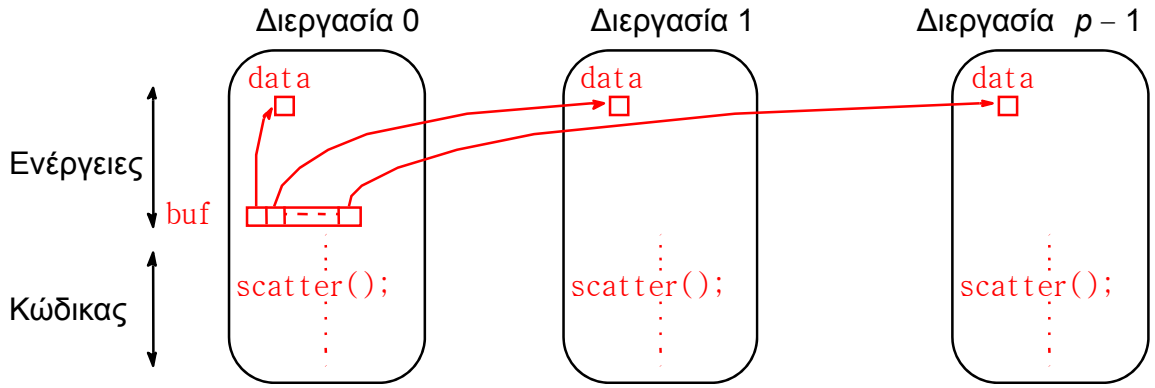
- Γίνεται αποστολή του ίδιου μηνύματος σε όλες τις διεργασίες.
- Μία παραλλαγή της εκπομπής είναι το Multicast όπου το ίδιο μήνυμα παραλαμβάνεται από ένα συγκεκριμένο υποσύνολο διεργασιών που έχει από πριν προσδιορισθεί



MPI υλοποίηση

Διασπορά - Scatter

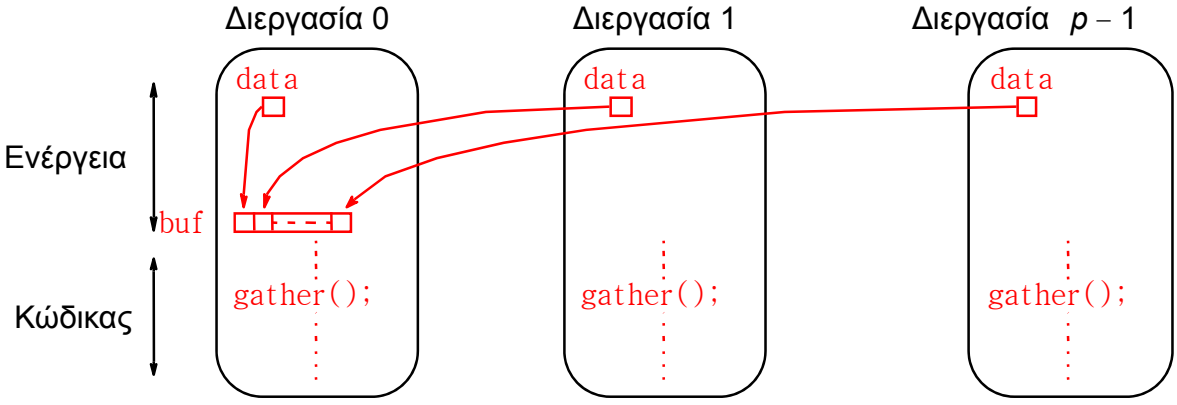
Η διεργασία ρίζα μοιράζει τα στοιχεία ενός πίνακα σε ξεχωριστές διεργασίες. Συγκεκριμένα τα περιεχόμενα της ι-οστης θέσης του πίνακα στέλνονται στην ι-οστη διεργασία



MPI υλοποίηση

Συγκέντρωση -Gather

Σε αυτή τη λειτουργία μία διεργασία (η διεργασία ρίζα) συλλέγει ξεχωριστά δεδομένα από ένα σύνολο διεργασιών. Έχει ουσιαστικά το αντίστροφο αποτέλεσμα από τη scatter

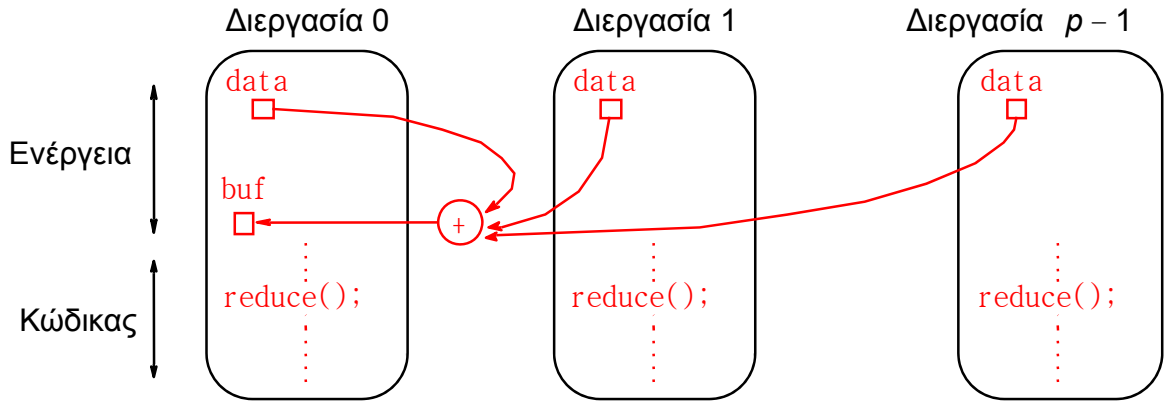


MPI υλοποίηση

Μείωση - Reduce

Η λειτουργία αυτή είναι μία λειτουργία συγκέντρωσης όπου στα δεδομένα που συγκεντρώνονται εκτελείται μία συγκεκριμένη αριθμητική/λογική λειτουργία

Παράδειγμα: Οι τιμές συγκεντρώνονται σε μία διεργασία (διεργασία ρίζα) και στη συνέχεια αθροίζονται από αυτή τη διεργασία:



MPI υλοποίηση

MPI (Message Passing Interface)

- Η βιβλιοθήκη MPI είναι ένα πρότυπο παράλληλου προγραμματισμού συστημάτων κατανεμημένης μνήμης και είναι αποτέλεσμα της σύμπραξης ερευνητικών κέντρων και επιχειρήσεων
- Ορίζει μόνο τις συναρτήσεις (παράμετροι εισόδου, τιμές επιστροφής κτλ.) και όχι πως αυτές υλοποιούνται.
- Υπάρχουν πολλές «ελεύθερες» υλοποιήσεις της βιβλιοθήκης

MPI - Δημιουργία Διεργασίας και Εκτέλεση

- Σκοπίμως δεν ορίζεται ο τρόπος με τον οποίο δημιουργούνται οι διεργασίες και εκτελούνται. Οι λεπτομέρειες καθορίζονται από τη συγκεκριμένη υλοποίηση.
- Στην έκδοση 1 του MPI, μόνο στατικές διεργασίες υποστηριζόταν. Όλες οι διεργασίες θα πρέπει να οριστούν πριν την εκτέλεση του παράλληλου προγράμματος και αρχίσουν όλες μαζί.
- Αρχικά επίσης, υποστηριζόταν μόνο το μοντέλο υπολογισμού SPMD.
- Και το μοντέλο MPMD μπορεί να συνδυαστεί με το στατικό τρόπο δημιουργίας διεργασιών – κάθε πρόγραμμα που θα εκτελεστεί ορίζεται εξ' αρχής.

Communicators

- Οι communicators ορίζουν την εμβέλεια μίας λειτουργίας επικοινωνίας.
- Σε κάθε διεργασία που ανήκει στον ίδιο Communicator αντιστοιχεί ένα μοναδικό αναγνωριστικό (rank).
- Αρχικά, όλες οι διεργασίες ανήκουν σε ένα «παγκόσμιο» communicator, τον MPI_COMM_WORLD. Επίσης κάθε διεργασία έχει ένα μοναδικό αναγνωριστικό, ένα αριθμό που κυμαίνεται από 0 μέχρι *p* – 1 όπου *p* είναι το πλήθος των διεργασιών του παράλληλου προγράμματος
- Στη συνέχεια, άλλοι communicators μπορούν να οριστούν σε υποομάδες των p διεργασιών.

Χρησιμοποιώντας το υπολογιστικό μοντέλο SPMD

```
main (int argc, char *argv[])
{
MPI_Init(&argc, &argv);
        .
        .
MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /*find process rank */

if (myrank == 0)
        master();
else
        slave();
        .
        .
MPI_Finalize();
}
```

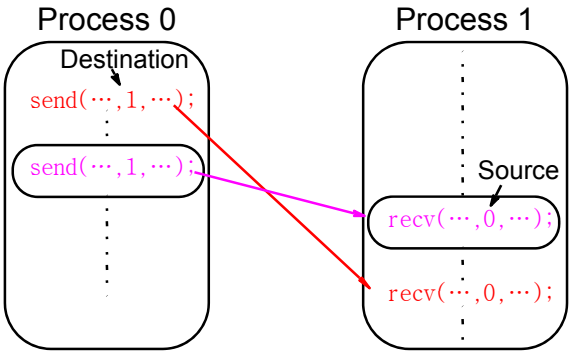
Υπάρχει μία διεργασία master (myrank=0) και μία ή περισσότερες διεργασίες slave (myrank >0)

Αν και το ίδιο πρόγραμμα τρέχει σε όλες τις διεργασίες, εντούτοις οι διεργασίες εκτελούν διαφορετικούς υπολογισμούς ανάλογα με τη τιμή του αναγνωριστικού κάθε διεργασίας.

Η συνάρτηση MPI_Comm_rank επιστρέφει το αναγνωριστικό της διεργασίας που την εκτελεί.

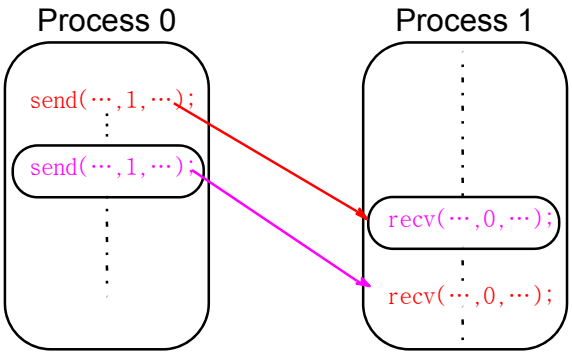
Πιθανά σενάρια εκτέλεσης των συναρτήσεων send και receive

α) Το δεδομένα από το πρώτο (δεύτερο) send της διεργασίας 0 θα πρέπει να παραληφθούν από το δεύτερο (πρώτο) recv της δεύτερης διεργασίας .



β) Πιθανή συμπεριφορά

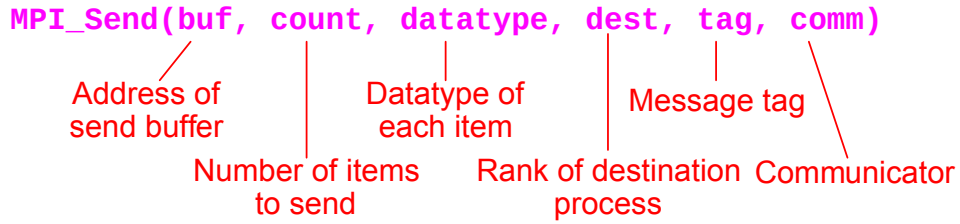
Για να λυθεί το πρόβλημα, κάθε μήνυμα θα πρέπει να έχει μία μοναδική ετικέτα (MESSAGE_TAG) και κάθε recv πρέπει να δηλώνει την ετικέτα του μηνύματος που θα λάβει



MPI – Επικοινωνία Σημείο σε Σημείο (Point-to-Point)

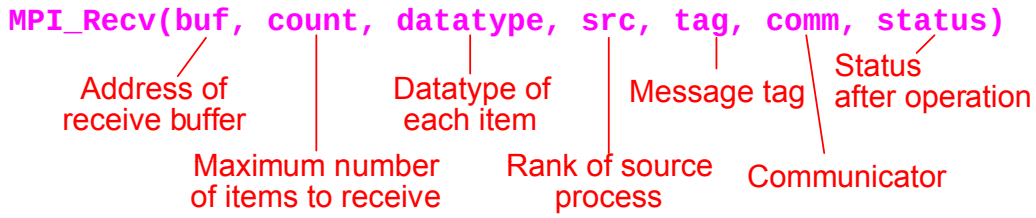
- Οι βασικές ρουτίνες send και receive υλοποιούν τις επικοινωνίες Σημείο-σε-Σημείο.
- Υπάρχουν και blocking και non-blocking διαδικασίες send και recv.
- Οι blocking ρουτίνες επιστρέφουν όταν η τοπική επεξεργασία έχει ολοκληρωθεί δηλ. όταν η θέση που κρατά το μήνυμα μπορεί να χρησιμοποιηθεί ξανά ή να αλλαχθεί χωρίς να επηρεαστεί η αποστολή ή λήψη του μηνύματος.
- Όταν ένα blocking send επιστρέφει δεν σημαίνει ότι το μήνυμα έχει ληφθεί από τον παραλήπτη, απλά η διεργασία είναι ελεύθερη να συνεχίσει με την εκτέλεση των επόμενων εντολών στο πρόγραμμα χωρίς να επηρεαστεί η εν εξελίξει αποστολή μηνύματος

Παράμετροι του blocking send



• buf: ο χώρος που περιέχει τα δεδομένα που θα σταλούν
• count: το πλήθος των δεδομένων που θα σταλούν
• datatype: ο τύπος των δεδομένων που θα σταλούν: MPI_INT για ακεραίους, MPI_CHAR για χαρακτήρες, MPI_FLOAT για αριθμούς κινητής υποδιαστολής
• dest: το αναγνωριστικό της διεργασίας παραλήπτη του μηνύματος
• tag: η ετικέτα με την οποία θα σταλεί το μήνυμα
• comm: ο communicator στην εμβέλεια του οποίου θα εκτελεσθεί η αποστολή του μηνύματος

Παράμετροι του blocking receive



• buf: ο χώρος που θα δεχθεί τα δεδομένα που θα ληφθούν
• count: το μέγιστο πλήθος των δεδομένων που θα ληφθούν
• datatype: ο τύπος των δεδομένων που θα ληφθούν: MPI_INT για ακεραίους, MPI_CHAR για χαρακτήρες, MPI_FLOAT για αριθμούς κινητής υποδιαστολής
• src: το αναγνωριστικό της διεργασίας αποστολέα του μηνύματος
• tag: η ετικέτα του μηνύματος που θα λάβει το παραλήπτης
• comm: ο communicator στην εμβέλεια του οποίου θα εκτελεσθεί η λήψη του μηνύματος

Παράδειγμα

- Ακολουθεί παράδειγμα αποστολής ενός ακεραίου αποθηκευμένο στη τοπική μεταβλητή x της διεργασίας 0 στη διεργασία 1. Τα δεδομένα λαμβάνονται και αποθηκεύονται στη τοπική μεταβλητή x της διεργασίας 1.

```
MPI_Comm_rank(MPI_COMM_WORLD,&myrank); /* find rank */

if (myrank == 0) {
    int x;
    MPI_Send(&x, 1, MPI_INT, 1, msgtag, MPI_COMM_WORLD);
} else if (myrank == 1) {
    int x;
    MPI_Recv(&x, 1, MPI_INT, 0,msgtag,MPI_COMM_WORLD,status);
}
```

Nonblocking MPI συναρτήσεις

- Nonblocking send - MPI_Isend(). Η λειτουργία αυτή επιστρέφει κατευθείαν πριν ακόμα ο αποθηκευτικός χώρος του μηνύματος που θα σταλεί να ελευθερωθεί και επομένως να είναι έτοιμος για τροποποίηση.
- Nonblocking receive - MPI_Irecv(). Η λειτουργία αυτή θα επιστρέψει αμέσως ακόμα και όταν δεν υπάρχει μήνυμα για παραλαβή

Nonblocking MPI Συναρτήσεις

- Υπάρχουν οι ακόλουθες non-blocking συναρτήσεις:

```
MPI_Isend(buf,count,datatype,dest,tag,comm,request)
MPI_Irecv(buf,count,datatype,source,tag,comm, request)
```

- Η ολοκλήρωση των συναρτήσεων μπορούν να διαπιστωθούν με τις συναρτήσεις **`MPI_Wait()`** και **`MPI_Test()`**.
- Η **`MPI_Wait()`** περιμένει μέχρι η λειτουργία να ολοκληρωθεί και στη συνέχεια επιστρέφει.
- Η **`MPI_Test()`** επιστρέφει αμέσως και θέτει μία λογική μεταβλητή που προσδιορίζει αν η εκκρεμής λειτουργία είχε ολοκληρωθεί τη στιγμή που έγινε ο έλεγχος.
- Για να ελέγξουμε μία συγκεκριμένη διαδικασία θα πρέπει να περάσουμε τη παράμετρο request που έχει πάρει τιμή από την MPI_Send()/MPI_Recv() στις συναρτήσεις **`MPI_Wait()`**, **`MPI_Test()`**

Παράδειγμα

- Ακολουθεί παράδειγμα αποστολής ενός ακεραίου x από την διεργασία 0 στη διεργασία 1. Η διεργασία 0 επιτρέπεται να συνεχίσει χωρίς να περιμένει την ολοκλήρωση της αποστολής μηνύματος. Θα περιμένει όμως στην MPI_Wait

```
MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /* find rank */
if (myrank == 0) {
   int x;
   MPI_Isend(&x,1,MPI_INT, 1, msgtag, MPI_COMM_WORLD, req1);
   compute();
   MPI_Wait(req1, status);
} else if (myrank == 1) {
   int x;
   MPI_Recv(&x,1,MPI_INT,0,msgtag, MPI_COMM_WORLD, status);
}
```

Συλλεκτική (Collective) Επικοινωνία

- Οι λειτουργίες συλλογικής επικοινωνίας δρουν στο σύνολο των διεργασιών που περιλαμβάνονται σε ένα communicator. Οι λειτουργίες αυτές δεν χρησιμοποιούν ετικέτες μηνυμάτων. Οι κύριες συλλεκτικές λειτουργίες επικοινωνίας είναι:
- **MPI_Bcast()**- Εκπομπή από μία συγκεκριμένη διεργασία (διεργασία ρίζα) σε όλες τις άλλες διεργασίες
- **MPI_Gather()**- Συλλογή δεδομένων από μία ομάδα διεργασιών στη διεργασία ρίζα
- **MPI_Scatter()**- Διασκόρπιση των δεδομένων της διεργασίας ρίζας σε μία ομάδα διεργασιών
- **MPI_Alltoall()**- Κάθε διεργασία στέλνει διαφορετικά δεδομένα σε κάθε άλλη διεργασία
- **MPI_Reduce()**- Εκτελεί το συνδυασμό (π.χ. άθροισμα, εύρεση ελάχιστου/μέγιστου) όλων των τιμών που δίνουν οι διεργασίες και το αποτέλεσμα αποθηκεύεται στη διεργασία ρίζα
- **MPI_Reduce_scatter()** Combine values and scatter results Συνδυάζει τις τιμές των διεργασιών όπως η Reduce αλλά στη συνέχεια διασκορπίζει το αποτέλεσμα σε όλες τις διεργασίες
- **MPI_Scan()** - Εκτελεί το υπολογισμό prefix στα δεδομένα των διεργασιών

Παράδειγμα

- Ακολουθεί παράδειγμα συλλογής δεδομένων από μία ομάδα διεργασιών στη διεργασία 0, χρησιμοποιώντας δυναμικά καταχωρημένη μνήμη στη διεργασία ρίζα:

```
int data[10];                 /*data to be gathered from processes*/
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);      /* find rank */
if (myrank == 0) {
   MPI_Comm_size(MPI_COMM_WORLD, &grp_size); /*find group size*/
   buf = (int *)malloc(grp_size*10*sizeof (int)); /*allocate memory*/
}
MPI_Gather(data,10,MPI_INT,buf,grp_size*10,MPI_INT,0,MPI_COMM_WORLD)
   ;
```

Η **`MPI_Gather()`** συλλέγει δεδομένα από τις διεργασίες συμπεριλαμβανομένη και της ρίζας.

Barrier (Φράγμα)

- Όπως σε όλα τα συστήματα περάσματος μηνυμάτων, έτσι και το MPI παρέχει ένα τρόπο για συγχρονισμό των διεργασιών. Αυτό σημαίνει ότι όταν η διεργασία κατά την εκτέλεση του προγράμματος φτάσει στη εντολή Barrier περιμένει μέχρι όλες οι άλλες διεργασίες να εκτελέσουν επίσης την εντολή Barrier. Στη συνέχεια όλες οι διεργασίες προχωρούν στην εκτέλεση των επόμενων εντολών.

Παράδειγμα προγράμματος MPI

```
#include “mpi.h”
#include <stdio.h>
#include <math.h>
#define MAXSIZE 1000
void main(int argc, char *argv)
{
    int myid, numprocs;
    int data[MAXSIZE], i, x, low, high, myresult, result;
    char fn[255];
    char *fp;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    if (myid == 0) {  /* Open input file and initialize data */
          strcpy(fn,getenv(“HOME”));
          strcat(fn,”/MPI/rand_data.txt”);
          if ((fp = fopen(fn,”r”)) == NULL) {
                    printf(“Can’t open the input file: %s\n\n”, fn);
                    exit(1);
          }
          for(i = 0; i < MAXSIZE; i++) fscanf(fp,”%d”, &data[i]);
    }
    MPI_Bcast(data, MAXSIZE, MPI_INT, 0, MPI_COMM_WORLD); /* broadcast data */
    x = n/nproc; /* Add my portion Of data */
    low = myid * x;
    high = low + x;
    for(i = low; i < high; i++)
          myresult += data[i];
    printf(“I got %d from %d\n”, myresult, myid); /* Compute global sum */
    MPI_Reduce(&myresult, &result, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    if (myid == 0) printf(“The sum is %d.\n”, result);
    MPI_Finalize();
}
```

Αποτίμηση απόδοσης παράλληλων προγραμμάτων

Χρόνος εκτέλεσης ακολουθιακού υπολογισμού t_s: Υπολογίζεται με τη μέτρηση των βημάτων του καλύτερου ακολουθιακού αλγόριθμου.

Χρόνος εκτέλεσης παράλληλου υπολογισμού, t_p: Επιπροσθέτως του πλήθους των υπολογιστικών βημάτων, t_{comp}, χρειάζεται να υπολογίσουμε την χρονική επιβάρυνση λόγω των επικοινωνιών μεταξύ των διεργασιών του παράλληλου προγράμματος, t_{comm}:

$$t_p = t_{comp} + t_{comm}$$

Χρόνος Υπολογισμού

Όπως αναφέρθηκε, προκύπτει από τη μέτρηση των βημάτων υπολογισμού. Όταν περισσότερες από μία διαδικασία εκτελείται ταυτόχρονα, μετρούμε τα βήματα της πιο σύνθετης διαδικασίας. Γενικά, ο χρόνος υπολογισμού είναι μία συνάρτηση του μεγέθους των δεδομένων n και του πλήθους των επεξεργαστών p, δηλ.

$$t_{comp} = f(n, p)$$

Συχνά ο υπολογισμός σπάει σε επιμέρους στάδια τα οποία χωρίζονται από γύρους επικοινωνίας όπου οι επεξεργαστές ανταλλάσουν αποτελέσματα από τα προηγούμενα στάδια επικοινωνίας. Έτσι ο συνολικός χρόνος υπολογισμού μπορεί να γραφεί ως:

$$t_{comp} = t_{comp1} + t_{comp2} + t_{comp3} + \ldots$$

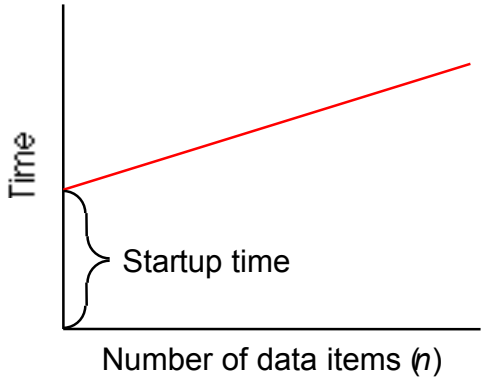
Η ανάλυση συνήθως γίνεται με την υπόθεση ότι όλοι οι επεξεργαστές είναι ίδιοι και λειτουργούν με την ίδια ταχύτητα.

Χρόνος Επικοινωνίας

Πολλοί παράγοντες επηρεάζουν το χρόνο επικοινωνίας όπως η δομή του διασυνδετικού δικτύου καθώς και της συμφόρεσης που μπορεί να παρουσιάζει. Σαν μία πρώτη προσέγγιση, χρησιμοποιούμε τον τύπο

$$t_{comm} = t_{startup} + nt_{data}$$

όπου $t_{startup}$ είναι ο χρόνος εκκίνησης και ουσιαστικά είναι ο χρόνος να σταλεί ένα μήνυμα όταν δεν περιέχει δεδομένα. Συνήθως ο χρόνος αυτός θεωρείται σταθερός. Η παράμετρος tdata είναι ο χρόνος μεταφοράς για να σταλεί μία λέξη. Επίσης στο παραπάνω τύπο, n είναι το πλήθος των λέξεων που στέλνονται κατά την επικοινωνία μεταξύ δύο διεργασιών..



Ο τελικός χρόνος επικοινωνίας είναι το άθροισμα των χρόνων επικοινωνίας όλων των μηνυμάτων που εστάλησαν από μία διεργασία, δηλ.

$$t_{comm} = t_{comm1} + t_{comm2} + t_{comm3} + \ldots$$

Τυπικά, τα μοτίβα επικοινωνίας όλων των διεργασιών είναι ίδια και επιπλέον θεωρείται ότι συμβαίνουν ταυτόχρονα. Έτσι για το υπολογισμό του συνολικού χρόνου επικοινωνίας συνήθως αρκεί να εξετάσουμε μία μόνο από τις παράλληλες διεργασίες

Τόσο ο χρόνος εκκίνησης $t_{startup}$ όσο και ο χρόνος μετάδοσης δεδομένων t_{data} εκφράζονται σε μονάδες χρόνου του ενός βήματος υπολογισμού και έτσι μπορούμε να αθροίσουμε t_{comp} και t_{comm} μαζί για να λάβουμε το χρόνο εκτέλεσης t_p.

Δείκτες Απόδοσης

Με γνωστά τα t_s, t_{comp}, and t_{comm}, μπορούμε να υπολογίσουμε τους παρακάτω δείκτες απόδοσης:

$$\text{Speedup factor} = \frac{t_s}{t_p} = \frac{t_s}{t_{comp} + t_{comm}}$$

$$\text{Computation/communication ratio} = \frac{t_{comp}}{t_{comm}}$$

Και οι δύο δείκτες είναι συναρτήσεις του πλήθους των επεξεργαστών, p, και των πλήθους των δεδομένων, n.

Και οι δύο δείκτες δείχνουν πόσο κλιμακούμενη είναι η παράλληλη λύση με την αύξηση του πλήθους των επεξεργαστών και του μεγέθους του προβλήματος.

Επίσης ο λόγος Υπολογισμός/Επικοινωνία δείχνει την επιβάρυνση λόγω επικοινωνίας στο συνολικό χρόνο λύσης με την αύξηση του πλήθους των επεξεργαστών και του μεγέθους του προβλήματος