Message Passing Interface (MPI)

Εισαγωγή – Βασικές Συναρτήσεις

- **Βιβλιοθήκη προτυποποιημένων συναρτήσεων για τη συγγραφή παράλληλων προγραμμάτων με πέρασμα μηνυμάτων (message passing).**
- **Υποστηρίζεται για τις γλώσσες προγραμματισμού Fortran και C/C++.**
- **Έκδοση 1.0/1994**
- **Έκδοση 2.2/2009**
- **Έκδοση 3.0/2012**
- **Έκδοση 3.1/2015 (τελευταία παγιωμένη έκδοση)**
- **Έκδοση 4.0/2018 (?)**

Ένα πρόγραμμα MPI αποτελείται από πολλές διεργασίες (processes) οι οποίες

- **έχουν το δικό τους χώρο διευθύνσεων καθ' όλη τη διάρκεια της εκτέλεσής τους**
- **εκτελούνται παράλληλα σε ένα ή περισσότερα υπολογιστικά συστήματα / επεξεργαστές.**

Ο προγραμματιστής όταν εκτελεί το πρόγραμμά του, μπορεί να καθορίσει ο ίδιος

- **τον αριθμό των διεργασιών που θα δημιουργηθούν**
- **το σύνολο των υπολογιστικών συστημάτων τα οποία θα αποτελούν το παράλληλο περιβάλλον του.**

Η κατανομή των διεργασιών στα διαθέσιμα υπολογιστικά συστήματα γίνεται από το MPI.

Πρότυπο Προγραμματισμού του MPI

<u>Σε χαμηλό επίπεδο:</u> Πρότυπο παράλληλου προγραμματισμού της *ανταλλαγής μηνυμάτων.*

- **Η ανταλλαγή δεδομένων μεταξύ των διεργασιών γίνεται με πέρασμα μηνυμάτων από τη μία διεργασία στην άλλη.**
- **Την αποστολή των μηνυμάτων την αναλαμβάνει το MPI.**
- **Ο προγραμματιστής καλεί συναρτήσεις αποστολής και λήψης μηνυμάτων χωρίς να ασχολείται με τις λεπτομέρειες της επικοινωνίας μεταξύ των διεργασιών.**

Πρότυπο Προγραμματισμού του MPI

Σε υψηλότερο επίπεδο: **Πρότυπο *SPMD (single program multiple data)*.**

- **Όλες οι διεργασίες (ανεξάρτητα από το υπολογιστικό σύστημα / επεξεργαστή στο οποίο τρέχουν) εκτελούν το ίδιο πρόγραμμα αλλά σε διαφορετικά εν δυνάμει δεδομένα.**

Πρότυπο Προγραμματισμού του MPI
Πρότυπο *SPMD*

Η διαφοροποίηση των λειτουργιών που εκτελεί μία διεργασία είναι εφικτή εφόσον:

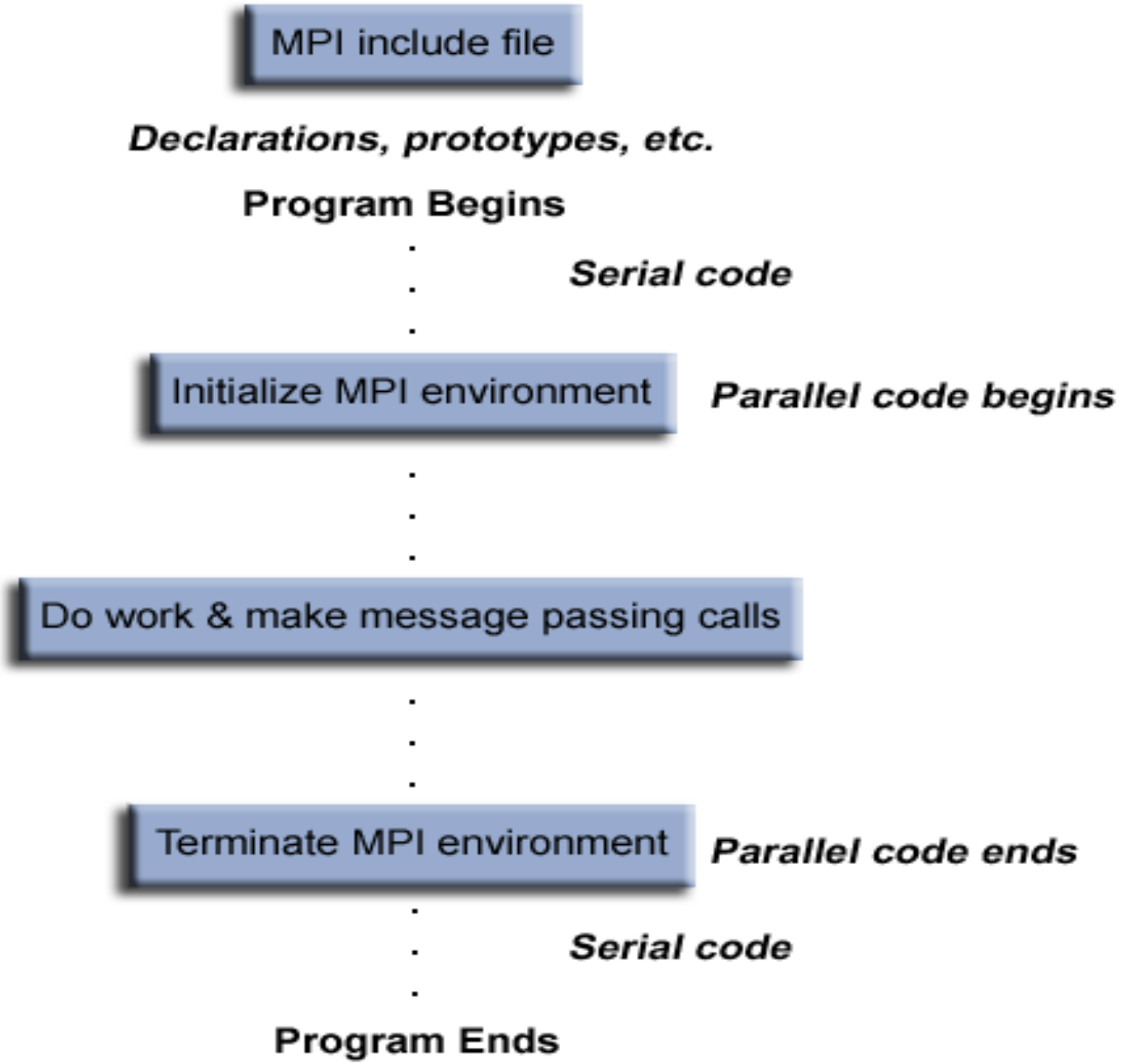
- **Όλες οι διεργασίες αριθμούνται με μια ξεχωριστή ταυτότητα η κάθε μία.**

  Στο MPI όταν εκτελείται ένα παράλληλο πρόγραμμα εκκινώντας *p* διεργασίες, αυτές αριθμούνται αυτόματα από *0* έως *p-1*. Οι ταυτότητες συνοδεύουν τις διεργασίες μέχρι την ολοκλήρωσή τους.

- **Κάθε διεργασία μπορεί να πληροφορείται σε χρόνο εκτέλεσης ποια είναι η ταυτότητά της.**

  Στο MPI αυτό γίνεται μέσω της συνάρτησης βιβλιοθήκης *MPI_Comm_Rank()*.

Η Δομή ενός Προγράμματος MPI



Η Δομή ενός Προγράμματος MPI

```
#include "mpi.h"
 /* Άλλες δηλώσεις #include - Ορισμός καθολικών μεταβλητών, συναρτήσεων  */

main(int argc, char **argv)
{
  /* Ορισμός μεταβλητών, δομών δεδομένων κ.λπ. Συγγραφή κώδικα σε C */

  /* Αρχικοποίηση του MPI */
  /* Δεν πρέπει να καλέσουμε άλλες MPI συναρτήσεις πριν από αυτήν */
   MPI_Init(&argc, &argv);
            .
 /* Υλοποίηση του αλγορίθμου της εφαρμογής με κλήση συναρτήσεων C και MPI */
            .
  /* Τερματισμός του MPI */
  /* Δεν πρέπει να καλέσουμε άλλες MPI συναρτήσεις μετά από αυτήν */
   MPI_Finalize();

  /* Δυνατότητα συγγραφής κώδικα μόνο σε C */
}
```

Η έννοια του Communicator

Communicator: αντιπροσωπεύει μία ομάδα διεργασιών που μπορούν να επικοινωνούν μεταξύ τους.

- ✓ **Για να επικοινωνήσουν δύο ή περισσότερες διεργασίες, πρέπει απαραίτητα, να ανήκουν στον ίδιο communicator.**

Βασικός communicator : MPI_COMM_WORLD.

- **δημιουργείται αυτόματα από το MPI και περιέχει όλες τις διεργασίες που** δημιουργούνται όταν αρχίζει η εκτέλεση του προγράμματος.
- **εξασφαλίζει ότι οποιαδήποτε διεργασία μπορεί να επικοινωνήσει με οποιαδήποτε άλλη.**

- ✓ **Μπορούμε να ορίσουμε επιπλέον communicators**

Η συνάρτηση MPI_Comm_size

int MPI_Comm_size(MPI_Comm Comm, int *size)

Comm : ο communicator το μέγεθος του οποίου επιστρέφεται μέσω της παραμέτρου size

MPI_Comm_size (MPI_COMM_WORLD, &size);

Μετά την κλήση, η μεταβλητή size περιέχει τον αριθμό των διεργασιών που περιέχονται στον MPI_COMM_WORLD, δηλαδή, τον αριθμό όλων των διεργασιών του εκτελούμενου προγράμματος.

Η συνάρτηση MPI_Comm_rank

int MPI_Comm_rank(MPI_Comm Comm, int *rank)

Comm είναι ο communicator στον οποίο ανήκει η διεργασία.

Στην παράμετρο rank επιστρέφεται η τάξη της διεργασίας η οποία κάνει την κλήση.

Επειδή η τάξη επιστρέφεται μέσω παραμέτρου, πρέπει στη συνάρτηση να περάσουμε τη διεύθυνση της παραμέτρου.

MPI_Comm_rank(MPI_COMM_WORLD, &rank);

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

int main(argc, argv)
int argc;
char *argv[ ];
{
  int    numtasks, rank, rc;

  rc = MPI_Init(&argc,&argv);
  if (rc != 0) {
    printf ("MPI initialization error\n");
    MPI_Abort(MPI_COMM_WORLD, rc);
  }

  MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  printf ("Number of processes = %d My rank = %d \n",
         numtasks,rank);

/*************************************************/
/*          Το κυρίως πρόγραμμα                  */
/*************************************************/

  MPI_Finalize();
}
```

program.c

mpicc –o program program.c

mpiexec –n 10 ./program

mpiexec –n 10 - -hostfile myhosts ./program

Η έξοδος του προγράμματος αν εκκινήσουμε 10 διεργασίες θα είναι η εξής:

Αριθμός διεργασιών = 10 My rank = 0
Αριθμός διεργασιών = 10 My rank = 2
Αριθμός διεργασιών = 10 My rank = 1
Αριθμός διεργασιών = 10 My rank = 4
Αριθμός διεργασιών = 10 My rank = 3
Αριθμός διεργασιών = 10 My rank = 6
Αριθμός διεργασιών = 10 My rank = 5
Αριθμός διεργασιών = 10 My rank = 9
Αριθμός διεργασιών = 10 My rank = 8
Αριθμός διεργασιών = 10 My rank = 7

- **Συνήθως με τυχαία σειρά το output**

Η συνάρτηση MPI_Get_processor_name

int MPI_Get_processor_name(char *name, int *resultlen)

name: Η ενδιάμεση μνήμη στην οποία επιστρέφεται μία ακολουθία χαρακτήρων (string) με το όνομα του επεξεργαστή.

resultlen: Το μήκος του string σε χαρακτήρες.

Η ενδιάμεση μνήμη name πρέπει να έχει μέγεθος τουλάχιστον, όσο είναι η τιμή της σταθεράς MPI_MAX_PROCESSOR_NAME.

- **Μπορούμε να έχουμε διαφορετικές αρχιτεκτονικές επεξεργαστών σε ένα παράλληλο σύστημα.**
- **Δεν μπορούμε να έχουμε δυο διαφορετικές υλοποιήσεις του MPI στο ίδιο σύστημα.**

Παράδειγμα

Να γραφεί πρόγραμμα MPI σε C στο οποίο κάθε μία από τις δημιουργούμενες διεργασίες να τυπώνει ένα μήνυμα που να λέει:

- **ποια είναι η ταυτότητά της (rank)**
- **πόσες διεργασίες έχουν εκκινηθεί συνολικά**
- **σε ποιον επεξεργαστή τρέχει.**

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

main(int argc, char** argv)
{
 int my_rank;
int size;
int namelen;
char proc_name[MPI_MAX_PROCESSOR_NAME];

/* η συνάρτηση αρχικοποίησης του MPI */
MPI_Init(&argc, &argv);

/* επιστροφή της ταυτότητας κάθε διεργασίας */
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

/* επιστροφή του συνολικού πλήθους των διεργασιών */
MPI_Comm_size(MPI_COMM_WORLD, &size);

/* επιστροφή του ονόματος της μηχανής στην οποία τρέχει η διεργασία, στη μεταβλητή
‘proc_name’ */
MPI_Get_processor_name(proc_name, &namelen);

printf("Hello from process %d of %d on %s\n", my_rank, size, proc_name);

MPI_Finalize();
}
```

Άσκηση

‘hello’ πρόγραμμα στο οποίο

- **ο επεξεργαστής ‘2’ τυπώνει προσωποποιημένα την ταυτότητα του ενώ**
- **οι υπόλοιποι τυπώνουν την ταυτότητά τους, το πλήθος των επεξεργαστών και το hostname του υπολογιστή στον οποίον τρέχουν, παραμετρικά.**

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

main(int argc, char** argv)
{
 int my_rank;
int size;
int namelen;
char proc_name[MPI_MAX_PROCESSOR_NAME];

MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Get_processor_name(proc_name, &namelen);

if (my_rank == 2)
printf(“Hello – I am process 2\n”);
else
printf(“Hello from process %d of  %d on %s\n”, my_rank, size, proc_name);
 MPI_Finalize();
}
```

mpiexec –n 4 ./hello

Hello from process 1 of 4 on mpi9

Hello – I am process 2

Hello from process 0 of 4 on mpi9

Hello from process 3 of 4 on mpi9