Υλοποιήσεις του MPI

Το MPI είναι ένα standard και όχι μια βιβλιοθήκη. Δημοφιλείς υλοποιήσεις του MPI standard είναι οι ακόλουθες:

- MPICH
- OPENMPI
- Intel MPI ($)
- MS-MPI

Εγκατάσταση του MPICH σε Ubuntu

1. Download της έκδοσης mpich-3.2 (stable release) από το https://www.mpich.org/downloads/
2. Αποσυμπίεση του αρχείου mpich-3.2.tar.gz σε έναν φάκελο π.χ. ~\mpich-3.2
3. Μετάβαση στο φάκελο ~\mpich-3.2
4. Εκτέλεση της εντολής ./configure –disable-fortran
5. Εκτέλεση της εντολής make
6. Εκτέλεση της εντολής sudo make install
7. Έλεγχος ότι έχει γίνει σωστή εγκατάσταση με την εντολή mpiexec --version

Παράδειγμα 1

Δημιουργία ενός αριθμού διεργασιών και αποστολή ενός μηνύματος από κάθε διεργασία πλην της διεργασίας 0 στην διεργασία 0.

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

#define MAX 100

int main(void) {
  char message[MAX];
  int comm_sz, my_rank;

  MPI_Init(NULL, NULL);
  MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

  if (my_rank != 0) {
    sprintf(message, "Process %d of %d says hello.", my_rank, comm_sz);
    MPI_Send(message, strlen(message) + 1, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
  } else {
    printf("This is  process %d waiting for messages\n", my_rank);
    for (int q = 1; q < comm_sz; q++) {
      MPI_Recv(message, MAX, MPI_CHAR, q, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      printf("%s\n", message);
    }
  }

  MPI_Finalize();
```

```
  return 0;
}
```

lab05_01.c

Μεταγλώττιση – εκτέλεση – έξοδος

```
mpicc lab05_01.c -o lab05_01
```

```
mpirun -n 5 ./lab05_01
```

```
This is  process 0 waiting for messages
Process 1 of 5 says hello.
Process 2 of 5 says hello.
Process 3 of 5 says hello.
Process 4 of 5 says hello.
```

Παράδειγμα 2

Η διεργασία 0 ζητά από το χρήστη να εισάγει μια πραγματική και μια ακέραια τιμή και τις στέλνει στις υπόλοιπες διεργασίες που εμφανίζουν τα δεδομένα που λαμβάνουν.

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

int main(int argc, char *argv[]) {
  int my_rank, comm_sz, dest;
  double a;
  int b;

  MPI_Init(NULL, NULL);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
  if (my_rank == 0) {
    printf("Enter 2 values (double and int): ");
    scanf("%lf %d", &a, &b);
    for (dest = 1; dest < comm_sz; dest++) {
      MPI_Send(&a, 1, MPI_DOUBLE, dest, 0, MPI_COMM_WORLD);
      MPI_Send(&b, 1, MPI_INT, dest, 0, MPI_COMM_WORLD);
    }
  } else {
    MPI_Recv(&a, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    MPI_Recv(&b, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Process %d got value %lf and value %d\n", my_rank, a, b);
  }
  MPI_Finalize();
  return 0;
}
```

lab05_02.c

Μεταγλώττιση – εκτέλεση – έξοδος

```
mpicc lab05_02.c -o lab05_02
```

```
mpirun -n 5 ./lab05_02
```

```
Enter 2 values (double and int): 2.1 3
Process 3 got value 2.100000 and value 3
Process 2 got value 2.100000 and value 3
Process 4 got value 2.100000 and value 3
```

```
Process 1 got value 2.100000 and value 3
```

Παράδειγμα 3 (broadcast)

Η διεργασία 0 ζητά από το χρήστη να εισάγει μια πραγματική και μια ακέραια τιμή και τις στέλνει με broadcast στις υπόλοιπες διεργασίες που εμφανίζουν τα δεδομένα που λαμβάνουν.

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

int main(int argc, char* argv[]){
    int my_rank, comm_sz;
    double a;
    int b;

    MPI_Init(NULL,NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
    if (my_rank==0){
        printf("Enter 2 values (double and int): ");
        scanf("%lf %d",&a,&b);
    }
    MPI_Bcast(&a, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(&b, 1, MPI_INT, 0, MPI_COMM_WORLD);
    if (my_rank!=0)
        printf("Process %d got value %lf and value %d\n", my_rank, a, b);

    MPI_Finalize();
    return 0;
}
```

lab05_03.c

Μεταγλώττιση – εκτέλεση – έξοδος

```
mpicc lab05_03.c -o lab05_03
```

```
mpirun -n 5 ./lab05_03
```

```
Enter 2 values (double and int): 2.1 3
Process 2 got value 2.100000 and value 3
Process 4 got value 2.100000 and value 3
Process 1 got value 2.100000 and value 3
Process 3 got value 2.100000 and value 3
```

Παράδειγμα 4

Κάθε διεργασία παράγει έναν τυχαίο αριθμό από το 1 μέχρι και το 100, τον στέλνει στην διεργασία 0 η οποία αθροίζει όλες τις τιμές που λαμβάνει.

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

int main(int argc, char *argv[]) {
```

```
  int my_rank, comm_sz, dest;
  int a, sum;

  MPI_Init(NULL, NULL);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
  if (my_rank == 0) {
    sum = 0;
    for (dest = 1; dest < comm_sz; dest++) {
      MPI_Recv(&a, 1, MPI_INT, dest, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      printf("Process 0 got value %d from process %d\n", a, dest);
      sum += a;
    }
    printf("The sum is %d\n", sum);
    fflush(stdout);
  } else {
    srand(time(NULL) * my_rank);
    int x = rand() % 100 + 1;
    MPI_Send(&x, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
  }
  MPI_Finalize();
  return 0;
}
```

lab05_04.c

Μεταγλώττιση – εκτέλεση – έξοδος

```
mpicc lab05_04.c -o lab05_04
```

```
mpirun -n 5 ./lab05_04
```

```
Process 0 got value 36 from process 1
Process 0 got value 6 from process 2
Process 0 got value 22 from process 3
Process 0 got value 92 from process 4
The sum is 156
```

Παράδειγμα 5 (reduce)

Κάθε διεργασία παράγει έναν τυχαίο αριθμό από το 1 μέχρι και το 100, τον στέλνει στην διεργασία 0 η οποία αθροίζει με τη χρήση του MPI_Reduce όλες τις τιμές που λαμβάνει.

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

int main(int argc, char* argv[]){
    int my_rank, comm_sz;
    int a=0, sum;

    MPI_Init(NULL,NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
    if (my_rank!=0){
```

```
        srand(time(NULL)*my_rank);
        a = rand() % 100 + 1;
        printf("Process %d produced value %d\n", my_rank, a);
    }
    MPI_Reduce(&a, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    if (my_rank==0)
        printf("The sum is %d\n", sum);

    MPI_Finalize();
    return 0;
}
```

lab05_05.c

Μεταγλώττιση – εκτέλεση – έξοδος

```
mpicc lab05_05.c -o lab05_05
```

```
mpirun -n 5 ./lab05_05
```

```
Process 3 produced value 45
Process 1 produced value 46
Process 2 produced value 15
The sum is 187
Process 4 produced value 81
```

Παράδειγμα 6 (all reduce)

Για έναν αριθμό από διεργασίες κάθε διεργασία παράγει έναν τυχαίο αριθμό από το 1 μέχρι και το 100. Όλες οι τυχαίες τιμές που παράγονται αθροίζονται και το αποτέλεσμα θα πρέπει να είναι διαθέσιμο σε όλες τις διεργασίες.

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

int main(int argc, char *argv[]) {
  int my_rank, comm_sz;
  int sum, a = 0;

  MPI_Init(NULL, NULL);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

  srand(time(NULL) * my_rank);
  a = rand() % 100 + 1;
  printf("Process %d produced value %d\n", my_rank, a);

  MPI_Allreduce(&a, &sum, 1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
  printf("Process %d knows that the sum is %d\n", my_rank, sum);

  MPI_Finalize();
  return 0;
}
```

lab05_06.c

Μεταγλώττιση – εκτέλεση – έξοδος

```
mpicc lab05_06.c -o lab05_06
```

```
mpirun -n 5 ./lab05_06
```

```
Process 4 produced value 61
Process 2 produced value 44
Process 0 produced value 84
Process 3 produced value 39
Process 1 produced value 60
Process 0 knows that the sum is 288
Process 4 knows that the sum is 288
Process 2 knows that the sum is 288
Process 3 knows that the sum is 288
Process 1 knows that the sum is 288
```

Παράδειγμα 7 (scatter-gather)

Η διεργασία 0 δημιουργεί έναν πίνακα με 10 τυχαίες τιμές. Αν εκτελεστεί με 5 διεργασίες στέλνει με την MPI_Scatter από 2 στοιχεία του πίνακα με block allocation σε κάθε διεργασία συμπεριλαμβανομένης και της ίδιας. Δηλαδή η διεργασία 0 λαμβάνει τα 2 πρώτα στοιχεία του πίνακα, η διεργασία 1 τα δύο επόμενα κ.ο.κ. Στη συνέχεια κάθε στοιχείο που έχει λάβει η κάθε διεργασία διπλασιάζεται και τα αποτελέσματα συγκεντρώνονται με την MPI_Gather.

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

#define N 10

int main(int argc, char *argv[]) {
  int *a = NULL;
  int my_rank, comm_sz;
  int i;
  int *local_a;
  int local_n;

  MPI_Init(NULL, NULL);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
  local_n = N / comm_sz;
  local_a = malloc(sizeof(int) * local_n);

  if (my_rank == 0) {
    srand(time(NULL));
    a = malloc(sizeof(int) * N);
    for (i = 0; i < N; i++) {
      a[i] = rand() % 10 + 1;
      printf("a[%d]=%d ", i, a[i]);
    }
    printf("\n");
    MPI_Scatter(a, local_n, MPI_INT, local_a, local_n, MPI_INT, 0,
                MPI_COMM_WORLD);
    free(a);
  } else {
```

```
    MPI_Scatter(a, local_n, MPI_INT, local_a, local_n, MPI_INT, 0,
                MPI_COMM_WORLD);
  }
  printf("Process %d ", my_rank);
  for (i = 0; i < local_n; i++)
    printf("%d ", local_a[i]);
  printf("\n");

  for (i = 0; i < local_n; i++)
    local_a[i] = local_a[i] * 2;

  if (my_rank == 0) {
    a = malloc(N * sizeof(int));
    MPI_Gather(local_a, local_n, MPI_INT, a, local_n, MPI_INT, 0,
               MPI_COMM_WORLD);
    printf("Result after gather ");
    for (int i = 0; i < N; i++)
      printf("a[%d]=%d ", i, a[i]);
    printf("\n");
  } else {
    MPI_Gather(local_a, local_n, MPI_INT, a, local_n, MPI_INT, 0,
               MPI_COMM_WORLD);
  }
  MPI_Finalize();
  return 0;
}
```

lab05_07.c

Μεταγλώττιση – εκτέλεση – έξοδος

```
mpicc lab05_07.c -o lab05_07
```

```
mpirun -n 5 ./lab05_07
```

```
a[0]=5 a[1]=9 a[2]=10 a[3]=7 a[4]=10 a[5]=1 a[6]=4 a[7]=8 a[8]=4 a[9]=6
Process 0 5 9
Process 4 4 6
Process 2 10 1
Process 1 10 7
Process 3 4 8
Result after gather a[0]=10 a[1]=18 a[2]=20 a[3]=14 a[4]=20 a[5]=2 a[6]=8 a[7]=16 a[8]=8
a[9]=12
```

Παράδειγμα 8

Υπολογισμός του αριθμού π με βάση τον τύπο:

$$\pi = \int_0^1 \frac{4}{1+x^2} dx$$

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

/*
σελίδα 191: Παράλληλα συστήματα και Προγραμματισμός (ΚΑΛΛΙΠΟΣ – Β. Δημακόπουλος)
*/
```

```c
int main(int argc, char *argv[]) {
  int N, i, myid, nproc;
  MPI_Status status;
  double w, result = 0.0, temp;

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

  // initialization
  if (myid == 0) {
    printf("Enter number of divisions: ");
    scanf("%d", &N);
    for (i = 1; i < nproc; i++)
      MPI_Send(&N, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
  } else
    MPI_Recv(&N, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);

  // computations
  w = 1.0 / N;
  for (i = myid; i < N; i += nproc)
    result += 4 * w / (1 + (i + 0.5) * (i + 0.5) * w * w);

  // gather results
  if (myid == 0) {
    for (i = 1; i < nproc; i++) {
      MPI_Recv(&temp, 1, MPI_DOUBLE, i, 0, MPI_COMM_WORLD, &status);
      // receives messages from any source
      // MPI_Recv(&temp, 1, MPI_DOUBLE, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD,
      // &status);
      result += temp;
    }
    printf("pi=%.14f\n", result);
  } else
    MPI_Send(&result, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);

  MPI_Finalize();
  return 0;
}
```

lab05_08.c

Μεταγλώττιση – εκτέλεση – έξοδος

```
mpicc lab05_08.c -o lab05_08
```

```
mpirun -n 5 ./lab05_08
```

```
Enter number of divisions: 3000
pi=3.14159266284905
```