Message Passing Interface (MPI)

Συναρτήσεις Συλλογικής Επικοινωνίας (συνέχεια)

Η συνάρτηση MPI_Reduce

- **Συγκεντρώνει δεδομένα από όλες τις διεργασίες μιας ομάδας σε μία διεργασία της ομάδας.**
- **Επιπλέον εφαρμόζεται και αποθηκεύεται το αποτέλεσμα μιας πράξης πάνω σε αυτά τα δεδομένα (reduction).**
- **Οι πράξεις οι οποίες υποστηρίζονται: το άθροισμα, το γινόμενο, το λογικό ή, το λογικό και, την εύρεση του μέγιστου και του ελάχιστου κ.α.**

Η συνάρτηση MPI_Reduce

int MPI_Reduce (void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

- **sendbuf: Η διεύθυνση μνήμης που περιέχει τα προς αποστολή δεδομένα κάθε διεργασίας που συμμετέχει στο μάζεμα.**
- **count: Ο αριθμός των στοιχείων που στέλνει κάθε διεργασία.**
- **datatype: Ο τύπος δεδομένων των στοιχείων που στέλνονται.**
- **op: Η πράξη που θα πραγματοποιηθεί στα δεδομένα.**
- **root: Η τάξη της ρίζας που θα συγκεντρώσει τα δεδομένα.**
- **comm: Ο communicator στον οποίο ανήκουν οι διεργασίες.**
- **recvbuf: Η διεύθυνση μνήμης όπου θα μαζευτούν τα δεδομένα-αποτελέσματα ρίζα μετά την εφαρμογή της πράξης.**

- **MPI_MAX εύρεση του μεγίστου των συμμετεχόντων στοιχείων (maximum)**
- **MPI_MIN εύρεση του ελαχίστου (minimum)**
- **MPI_SUM εύρεση του αθροίσματος (sum)**
- **MPI_PROD εύρεση του γινόμενου (product)**
- **MPI_LAND εύρεση του λογικού ‘and’**
- **MPI_BAND εύρεση του bit-προς-bit ‘and’**
- **MPI_LOR εύρεση του λογικού ‘or’**
- **MPI_BOR εύρεση του bit-προς-bit ‘or’**
- **MPI_LXOR εύρεση του λογικού ‘xor’**
- **MPI_BXOR εύρεση του bit-προς-bit ‘xor’**
- **MPI_MAXLOC εύρεση του μεγίστου στοιχείου και της θέσης αυτού**
- **MPI_MINLOC εύρεση του ελαχίστου στοιχείου και της θέσης αυτού**

Υπολογισμός $S = 1^2 + 2^2 + ... + (n-1)^2 + n^2$ με broadcast, reduce (bcast.c)

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

main(int argc, char** argv)
{
int my_rank;
int p, k;
int root;
int a1_local;
int a2_local;
int local_num;
int endnum;
int local_res;
int final_res;
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, &p);
MPI_Get_processor_name(proc_name, &namelen);
```

```
if (my_rank == 0)
 {
   printf("Dose plithos arithmvn:");
   scanf("%d", &endnum);
  }

root = 0;

/* Αποστολή στις διεργασίες του πλήθους των στοιχείων */

MPI_Bcast(&endnum, 1, MPI_INT, root, MPI_COMM_WORLD);

/* Υπολογισμός από κάθε μία του τοπικού της αθροίσματος */

local_res = 0;
local_num = endnum/p;
```

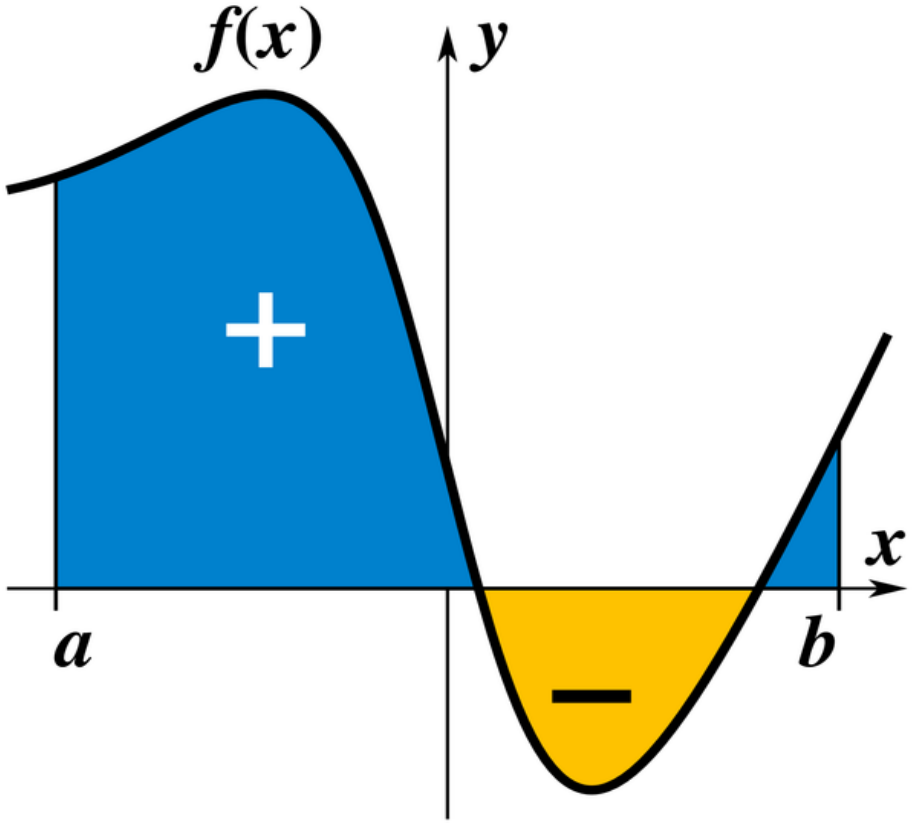
```
a1_local = (my_rank * local_num) + 1;
a2_local = a1_local + local_num - 1;

for (k=a1_local; k<=a2_local; k++)
   local_res = local_res + (k*k);
 printf("\n Process %d on %s : local result = %d \n", my_rank,
    proc_name, local_res);
 root = 0;
MPI_Reduce(&local_res, &final_res, 1, MPI_INT, MPI_SUM, root,
    MPI_COMM_WORLD);
 if (my_rank == 0)
 {
  printf("\n Total result for N = %d is equal to : %d \n", endnum,
    final_res);
 }
MPI_Finalize();   }
```

Υπολογισμός του αριθμού π (3,14…)

- Ο αριθμός ‘π’ μπορεί να υπολογιστεί ως η τιμή του ολοκληρώματος:

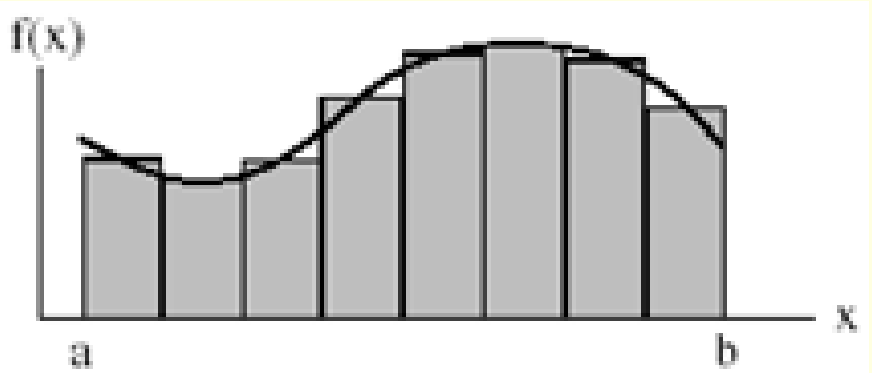
$$\int_0^1 \frac{4}{1+x^2}\,dx$$

Το ολοκλήρωμα της συνάρτησης *f*(*x*) από το *a* στο *b* είναι η επιφάνεια πάνω από τον άξονα *x* και κάτω από την καμπύλη *y* = *f*(*x*), μείον την επιφάνεια κάτω από τον άξονα *x* και πάνω από την καμπύλη, για *x* στο διάστημα [*a*,*b*].



Αριθμητική Ολοκλήρωση με Ορθογώνια

Ένας τρόπος να βρούμε προσεγγιστικές τιμές ολοκληρωμάτων είναι να αθροίσουμε εμβαδά ορθογωνίων:



Χρησιμοποιώντας n ορθογωνια, μηκους h = (b – a)/n:

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} h\,f(a + (i - 1/2)h)$$

Υπολογισμός του αριθμού π (3,14…)

- Ακολουθώντας τον κανόνα του ορθογωνίου χωρίζουμε το διάστημα [0,1] σε ‘n’ υποδιαστήματα ιδίου μεγέθους.
- Στη συνέχεια βρίσκουμε τις τιμές της συνάρτησης στα μεσαία σημεία των υποδιαστημάτων.
- Χρησιμοποιώντας αυτές τις τιμές ως ύψη στα υποδιαστήματα κατασκευάσουμε ‘n’ αντίστοιχα ορθογώνια.
- Αν αθροίσουμε το εμβαδόν όλων αυτών των ορθογωνίων μπορούμε να πάρουμε μια καλή προσέγγιση της τιμής του ολοκληρώματος δηλ. της τιμής του π.

Υπολογισμός του αριθμού π (3,14…) (pi_computation.c)

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

int main( int argc, char *argv[] )
{
 int n, myid, numprocs, i;
 double PIINIT = 3.14159265358979323846264 3;
 double mypi, pi, h, sum, x;
 int  namelen;

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

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

```
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 approximately %.15f, Error is %.15f\n", pi, fabs(pi-PIINIT));

  MPI_Finalize();
}
```

Υπολογισμός του average τυχαίων αριθμών με χρήση των MPI_Scatter , MPI_Gather (avg.c)

- **Δημιουργούμε στη ρίζα (διεργασία 0) ένα διάνυσμα τυχαίων αριθμών στο διάστημα [0,1]. Ο αριθμός των στοιχείων του διανύσματος προσδιορίζεται από το χρήστη.**
- **Διαμοιράζουμε (scatter) το διάνυσμα σε όλες τις διεργασίες (ίσος αριθμός στοιχείων/διεργασία).**
- **Κάθε διεργασία υπολογίζει το μέσο όρο (average) των αριθμών που της αντιστοιχούν.**
- **Συγκεντρώνουμε (gather) όλους τους μέσους όρους στη ρίζα (διεργασία 0). Η ρίζα υπολογίζει το μέσο όρο όλων των αριθμών.**

Υπολογισμός του average

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <assert.h>
int main(int argc, char** argv)
  {
   MPI_Init(&argc,&argv);
  int my_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  int p;
  MPI_Comm_size(MPI_COMM_WORLD, &p);
  int num_elements;
  int num_elements_per_proc;
```

```
 if (my_rank == 0)
 {
   printf("Dose plithos arithmvn:\n");
   scanf("%d", &num_elements);
  }
MPI_Bcast(&num_elements, 1, MPI_INT, 0, MPI_COMM_WORLD);
num_elements_per_proc = num_elements/p;

float *rand_nums = (float *)malloc(sizeof(float) * num_elements);

if (my_rank == 0) {
  int i;
  for (i = 0; i < num_elements; i++) {
    rand_nums[i] = (rand() / (float)RAND_MAX);
printf("%f\n",rand_nums[i]) ;}}
```

```
float *sub_rand_nums = (float *)malloc(sizeof(float) *
    num_elements_per_proc);
 MPI_Scatter(rand_nums, num_elements_per_proc, MPI_FLOAT,
    sub_rand_nums, num_elements_per_proc, MPI_FLOAT, 0,
    MPI_COMM_WORLD);


float sum = 0.0;
 float sub_avg;
  int l;
 for (l = 0; l < num_elements_per_proc; l++) {
  sum += sub_rand_nums[l];}
  sub_avg = sum/ (float)num_elements_per_proc ;
 float *sub_avgs ;
 if (my_rank == 0) {
  sub_avgs = (float *)malloc(sizeof(float) * p);
 }
```

```
MPI_Gather(&sub_avg, 1, MPI_FLOAT, sub_avgs, 1, MPI_FLOAT, 0,
    MPI_COMM_WORLD);

 if (my_rank == 0) {
   float avg;
   float tot_sum = 0.0;
   int j;
   for (j = 0; j < p; j++) {
   tot_sum += sub_avgs[j];}
   avg = tot_sum/ (float) p;
   printf("Avg of all elements is %f\n", avg);
  }

MPI_Finalize();
}
```