Lectia 5. FUNCȚIILE MPI PENTRU OPERAȚII DE REDUCERE

În programarea paralelă operațiile matematice pe blocuri de date care sunt distribuite între procesoare se numesc *operații globale de reducere*. O operație de acumulare se mai numește și operație globală de reducere. Fiecare proces pune la dispoziție un bloc de date, care sunt combinate cu o operație binară de reducere; rezultatul acumulat este colectat la procesul **root**. În MPI, o operațiune globală de reducere este reprezentată în următoarele moduri:

- menținerea rezultatelor în spațiul de adrese al unui singur proces (funcția **MPI_Reduce**);
- menținerea rezultatelor în spațiul de adrese al tuturor proceselor (funcția **MPI_Allreduce**);
- operația de reducere prefix, care în calitate de rezultat returnează un vector al cărui componentă *i* este rezultatul operației de reducere ale primelor *i* componente din vectorul distribuit (funcția **MPI_Scan**).

Funcția **MPI_Reduce** se execută astfel. Operația globală de reducere indicată prin specificarea parametrului **op**, se realizează pe primele elemente ale tamponului de intrare, iar rezultatul este trimis în primul element al tamponului de recepționare al procesului **root**. Același lucru se repetă pentru următoarele elemente din memoria tampon etc. Prototipul acestei funcții în limbajul C este:

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

unde

IN **sendbuf** – adresa inițială a tamponului pentru datele de intrare;

OUT **recvbuf** – adresa inițială a tamponului pentru rezultate (se utilizează numai de procesul **root**);

IN **count** – numărul de elemente în tamponul de intrare;

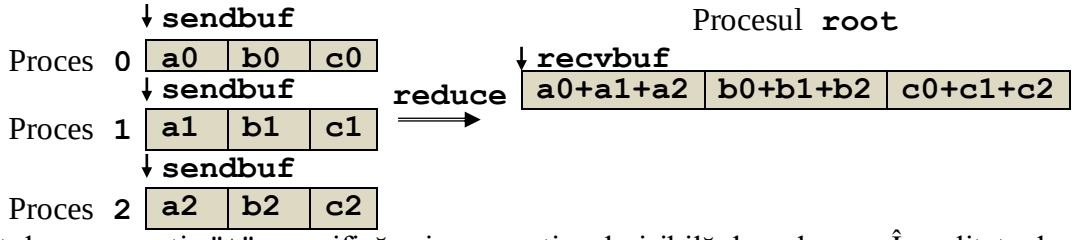
IN **datatype** – tipul fiecărui element din tamponul de intrare;

IN **op** – operația de reducere;

IN **root** – numărul procesului care recepționează rezultatele operației de reducere;

IN **comm** – comunicatorul implicat.

Grafic această funcție poate fi interpretată astfel:



În acest desen operația "+" semnifică orice operație admisibilă de reducere. În calitate de operație de reducere se poate utiliza orice operație predefinită sau operații determinate (construite) de utilizator folosind funcția **MPI_Op_create.**

În tabelul de mai jos sunt prezentate operațiile predefinite care pot fi utilizate în funcția **MPI_Reduce**.

Nume	**Operația**	**Tipuri de date admisibile**
MPI_MAX **MPI_MIN**	Maximum Minimum	integer, floating point
MPI_SUM **MPI_PROD**	Suma Produsul	integer,floating point, complex
MPI_LAND	AND	integer, logical

MPI_LOR **MPI_LXOR**	OR excludere OR	
MPI_BAND **MPI_BOR** **MPI_BXOR**	AND OR excludere OR	integer, byte
MPI_MAXLOC **MPI_MINLOC**	Valoarea maximală și indicele Valoarea minimală și indicele	Tip special de date

În tabelul de mai sus pentru tipuri de date se utilizează următoarele notații:

integer	**MPI_INT, MPI_LONG, MPI_SHORT, MPI_UNSIGNED_SHORT, MPI_UNSIGNED, MPI_UNSIGNED_LONG**
floating point	**MPI_FLOAT, MPI_DOUBLE, MPI_REAL, MPI_DOUBLE_PRECISION, MPI_LONG_DOUBLE**
logical	**MPI_LOGICAL**
complex	**MPI_COMPLEX**
byte	**MPI_BYTE**

Operațiile **MAXLOC** și **MINLOC** se execută pe un tip special de date fiecare element conținând două valori: valoarea maximului sau minimului și indicele elementului. În MPI există 9 astfel de tipuri predefinite.

MPI_FLOAT_INT	float and int
MPI_DOUBLE_INT	double and int
MPI_LONG_INT	long and int
MPI_2INT	int and int
MPI_SHORT_INT	short and int
MPI_LONG_DOUBLE_INT	long double and int

Vom ilustra utilizarea operațiilor globale de reducere **MPI_SUM** în baza următorului exemplu.

Exemplul 5.1 *Să se calculeze valoarea aproximativă a lui π prin integrare numerică cu formula* $\pi = \int_0^1 \frac{4}{1+x^2} dx$*, folosind formula dreptunghiurilor. Intervalul închis [0,1] se împarte într-un număr de n subintervale și se însumează ariile dreptunghiurilor având ca bază fiecare subinterval. Pentru execuția algoritmului în paralel, se atribuie, fiecăruia dintre procesele din grup, un anumit număr de subintervale. Cele două operații colective care apar în rezolvare sunt:*

- ✓ *difuzarea valorilor lui n, tuturor proceselor;*
- ✓ *însumarea valorilor calculate de procese.*

Mai jos este prezentat codul programului în limbajul C++ care determină valoarea aproximativă a lui π.

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
double f(double a)
{
   return (4.0 / (1.0 + a*a));
}
int main(int argc, char *argv[])
{
int done = 0, n, myid, numprocs, i;
```

```
double PI25DT = 3.141592653589793238462643;
double mypi, pi, h, sum, x;
double startwtime, endwtime;
int  namelen;
char processor_name[MPI_MAX_PROCESSOR_
       NAME];
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,
       &numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,
       &myid);
MPI_Get_processor_name(processor_name,&namelen);
n = 0;
   while (!done)
   {
       if (myid == 0)
       {
        printf("===== Rezultatele programului '%s' =====\n",argv[0]);
        printf("Enter the number of intervals: (0 quits) ");fflush(stdout);
          scanf("%d",&n);
          MPI_Barrier(MPI_COMM_WORLD);
          startwtime = MPI_Wtime();
       }
       else MPI_Barrier(MPI_COMM_WORLD);
       MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
       if (n == 0)
          done = 1;
       else
       {
          h   = 1.0 / (double) n;
          sum = 0.0;
          for (i = myid + 1; i <= n; i += numprocs)
          {
             x = h * ((double)i - 0.5);
             sum += f(x);
          }
          mypi = h * sum;
       fprintf(stderr,"Process %d on %s mypi= %.16f\n", myid, processor_name, mypi);
       fflush(stderr);
       MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
             if (myid == 0)
            {
             printf("Pi is approximately %.16f,
       Error is %.16f\n",
              pi, fabs(pi - PI25DT));
           endwtime = MPI_Wtime();
           printf("wall clock time = %f\n", endwtime-startwtime);
           }
       }
```

```
  }
  MPI_Finalize();
}
```

Rezultatele posibile ale executării programului:

```
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC -o Exemplu 3.4.4.exe Exemplu_3_4_4.cpp
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpirun -n 16 -machinefile ~/nodes4 Exemplu 3.4.4.exe
===== Rezultatele programului ' Exemplu_3_4_4.exe' =====
Enter the number of intervals: (0 quits) 100000
Process 1 on compute-0-2.local mypi= 0.1963576657186469
Process 2 on compute-0-2.local mypi= 0.1963564157936524
Process 3 on compute-0-2.local mypi= 0.1963551658561532
Process 4 on compute-0-4.local mypi= 0.1963539159061520
Process 10 on compute-0-6.local mypi= 0.1963464159436181
Process 12 on compute-0-8.local mypi= 0.1963439158561127
Process 0 on compute-0-2.local mypi= 0.1963589156311392
Process 6 on compute-0-4.local mypi= 0.1963514159686426
Process 8 on compute-0-6.local mypi= 0.1963489159811299
Process 15 on compute-0-8.local mypi= 0.1963401656311267
Process 7 on compute-0-4.local mypi= 0.1963501659811359
Process 9 on compute-0-6.local mypi= 0.1963476659686233
Process 14 on compute-0-8.local mypi= 0.1963414157186182
Process 5 on compute-0-4.local mypi= 0.1963526659436477
Process 11 on compute-0-6.local mypi= 0.1963451659061137
Process 13 on compute-0-8.local mypi= 0.1963426657936135
Pi is approximately 3.1415926535981260, Error is 0.0000000000083329
wall clock time = 0.000813
===== Rezultatele programului 'HB_Pi.exe' =====
Enter the number of intervals: (0 quits) 100000000
Process 8 on compute-0-6.local mypi= 0.1963495402243708
Process 15 on compute-0-8.local mypi= 0.1963495314743671
Process 4 on compute-0-4.local mypi= 0.1963495452243360
Process 5 on compute-0-4.local mypi= 0.1963495439743813
Process 3 on compute-0-2.local mypi= 0.1963495464743635
Process 11 on compute-0-6.local mypi= 0.1963495364743647
Process 1 on compute-0-2.local mypi= 0.1963495489743604
Process 13 on compute-0-8.local mypi= 0.1963495339743620
Process 14 on compute-0-8.local mypi= 0.1963495327243415
Process 0 on compute-0-2.local mypi= 0.1963495502243742
Process 7 on compute-0-4.local mypi= 0.1963495414743800
Process 2 on compute-0-2.local mypi= 0.1963495477243492
Process 12 on compute-0-8.local mypi= 0.1963495352243697
Process 9 on compute-0-6.local mypi= 0.1963495389743577
Process 6 on compute-0-4.local mypi= 0.1963495427243758
Process 10 on compute-0-6.local mypi= 0.1963495377243211
Pi is approximately 3.1415926535897749, Error is 0.0000000000000182
wall clock time = 0.172357
===== Rezultatele programului 'HB_Pi.exe' =====
Enter the number of intervals: (0 quits) 0
```

```
[Hancu_B_S@hpc]$
```

Vom exemplifica utilizarea operațiilor globale de reducere **MPI_MAX** și **MPI_MIN** în cele ce urmează.

Exemplul 5.2 *Să se determine elementele maximale de pe coloanele unei matrice pătrate (dimensiunea este egală cu numărul de procese). Elementele matricei sunt inițializate de procesul cu rankul 0. Să fie utilizate funcțiile* **MPI_Reduce** *și operația* **MPI_MAX**..

Mai jos este prezentat codul programului în limbajul C++ în care se realizează cele menționate în exemplul.5.2.

```
#include<mpi.h>
#include<stdio.h>
int main(int argc, char *argv[])
{
int numtask,sendcount,reccount,source;
double *A,*Max_Col;
int i, myrank, root=0;
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD, &numtask);
double Rows[numtask];
sendcount=numtask;
reccount=numtask;
if(myrank==root)
{
printf("\n=====REZULTATUL PROGRAMULUI
       '%s' \n",argv[0]);
A=(double*)malloc(numtask*numtask*sizeof
      (double));
Max_Col=(double*)malloc(numtask*sizeof
      (double));
      for(int i=0;i<numtask*numtask;i++)
      A[i]=rand()/1000000000.0;
      printf("Tipar datele initiale\n");
      for(int i=0;i<numtask;i++)
      {
      printf("\n");
      for(int j=0;j<numtask;j++)
      printf("A[%d,%d]=%5.2f ",i,j,A[i*numtask+j]);
      }
      printf("\n");
      MPI_Barrier(MPI_COMM_WORLD);
}
else MPI_Barrier(MPI_COMM_WORLD);
MPI_Scatter(A,sendcount,MPI_DOUBLE, Rows,reccount,MPI_DOUBLE,root,MPI_COMM_WORLD);
MPI_Reduce(Rows,Max_Col,numtask, MPI_DOUBLE, MPI_MAX, root, MPI_COMM_WORLD);
if (myrank==root) {
for(int i=0;i<numtask;i++)
     {
       printf("\n");
       printf("Elementul maximal de pe coloana %d=%5.2f ",i,Max_Col[i]);
```

```
      }
      printf("\n");
      MPI_Barrier(MPI_COMM_WORLD); }
      else MPI_Barrier(MPI_COMM_WORLD);
 MPI_Finalize();
return 0;
}
```

Rezultatele posibile ale executării programului:

```
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC -o Exemplu _3_4_5.exe Exemplu _3_4_5.cpp[1]
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpirun -n 6 -machinefile ~/nodes4 o Exemplu _3_4_5.exe

=====REZULTATUL PROGRAMULUI ' Exemplu _3_4_5.exe'
Tipar datele initiale

A[0,0]= 1.80 A[0,1]= 0.85 A[0,2]= 1.68 A[0,3]= 1.71 A[0,4]= 1.96 A[0,5]= 0.42
A[1,0]= 0.72 A[1,1]= 1.65 A[1,2]= 0.60 A[1,3]= 1.19 A[1,4]= 1.03 A[1,5]= 1.35
A[2,0]= 0.78 A[2,1]= 1.10 A[2,2]= 2.04 A[2,3]= 1.97 A[2,4]= 1.37 A[2,5]= 1.54
A[3,0]= 0.30 A[3,1]= 1.30 A[3,2]= 0.04 A[3,3]= 0.52 A[3,4]= 0.29 A[3,5]= 1.73
A[4,0]= 0.34 A[4,1]= 0.86 A[4,2]= 0.28 A[4,3]= 0.23 A[4,4]= 2.15 A[4,5]= 0.47
A[5,0]= 1.10 A[5,1]= 1.80 A[5,2]= 1.32 A[5,3]= 0.64 A[5,4]= 1.37 A[5,5]= 1.13

Elementul maximal de pe coloana 0= 1.80
Elementul maximal de pe coloana 1= 1.80
Elementul maximal de pe coloana 2= 2.04
Elementul maximal de pe coloana 3= 1.97
Elementul maximal de pe coloana 4= 2.15
Elementul maximal de pe coloana 5= 1.73
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpirun -n 8 -machinefile ~/nodes4 Exemplu _3_4_5.exe

=====REZULTATUL PROGRAMULUI ' Exemplu _3_4_5.exe'
Tipar datele initiale

A[0,0]= 1.80 A[0,1]= 0.85 A[0,2]= 1.68 A[0,3]= 1.71 A[0,4]= 1.96 A[0,5]= 0.42 A[0,6]= 0.72 A[0,7]= 1.65
A[1,0]= 0.60 A[1,1]= 1.19 A[1,2]= 1.03 A[1,3]= 1.35 A[1,4]= 0.78 A[1,5]= 1.10 A[1,6]= 2.04 A[1,7]= 1.97
A[2,0]= 1.37 A[2,1]= 1.54 A[2,2]= 0.30 A[2,3]= 1.30 A[2,4]= 0.04 A[2,5]= 0.52 A[2,6]= 0.29 A[2,7]= 1.73
A[3,0]= 0.34 A[3,1]= 0.86 A[3,2]= 0.28 A[3,3]= 0.23 A[3,4]= 2.15 A[3,5]= 0.47 A[3,6]= 1.10 A[3,7]= 1.80
A[4,0]= 1.32 A[4,1]= 0.64 A[4,2]= 1.37 A[4,3]= 1.13 A[4,4]= 1.06 A[4,5]= 2.09 A[4,6]= 0.63 A[4,7]= 1.66
A[5,0]= 1.13 A[5,1]= 1.65 A[5,2]= 0.86 A[5,3]= 1.91 A[5,4]= 0.61 A[5,5]= 0.76 A[5,6]= 1.73 A[5,7]= 1.97
A[6,0]= 0.15 A[6,1]= 2.04 A[6,2]= 1.13 A[6,3]= 0.18 A[6,4]= 0.41 A[6,5]= 1.42 A[6,6]= 1.91 A[6,7]= 0.75
A[7,0]= 0.14 A[7,1]= 0.04 A[7,2]= 0.98 A[7,3]= 0.14 A[7,4]= 0.51 A[7,5]= 2.08 A[7,6]= 1.94 A[7,7]= 1.83

Elementul maximal de pe coloana 0= 1.80
Elementul maximal de pe coloana 1= 2.04
Elementul maximal de pe coloana 2= 1.68
Elementul maximal de pe coloana 3= 1.91
```

[1] Numele programului corespunde numelui exemplului din notele de curs Boris HÎNCU, Elena CALMÎȘ "MODELE DE PROGRAMARE PARALELĂ PE CLUSTERE. PARTEA I. PROGRAMARE MPI". Chisinau 2016.

```
Elementul maximal de pe coloana 4= 2.15
Elementul maximal de pe coloana 5= 2.09
Elementul maximal de pe coloana 6= 2.04
Elementul maximal de pe coloana 7= 1.97
[Hancu_B_S@hpc Finale]$
```

Exemplul 5.2a *Să se determine elementele maximale de pe coloanele unei matrici patrate (dimensiunea este egala cu numarul de procese). Liniile matricei sunt initializate de fiecare proces în parte. Procesul cu rankul 0 în baza acestor linii "construieşte" matricea. Să fie utilizate functia* **`MPI_Reduce`** *si operatia* **`MPI_MAX`**.

Mai jos este prezentat codul programului în limbajul C++ în care se realizează cele menționate în exemplul 5.2a.

```
#include<mpi.h>
#include<stdio.h>
#include<stdlib.h>
int main(int argc, char *argv[])
{
int numtask,sendcount,reccount,source;
double *A,*Max_Col;
int i, myrank, root=0;
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD, &numtask);
double Rows[numtask];
sendcount=numtask;
reccount=numtask;
if(myrank==root)
{
printf("\n=====REZULTATUL PROGRAMULUI '%s' \n",argv[0]);
A=(double*)malloc(numtask*numtask*sizeof(double));
Max_Col=(double*)malloc(numtask*sizeof(double));
}
sleep(myrank);
srand(time(NULL));
for(int i=0;i<numtask;i++)
Rows[i]=rand()/1000000000.0;
printf("Tipar datele initiale ale procesului cu rankul %d \n",myrank);
for(int i=0;i<numtask;i++)
{
printf("Rows[%d]=%5.2f ",i,Rows[i]);
}
printf("\n");
MPI_Barrier(MPI_COMM_WORLD);
MPI_Gather(Rows, sendcount, MPI_DOUBLE, A, reccount, MPI_DOUBLE, root, MPI_COMM_WORLD);
if(myrank==root){
printf("\n");
printf("Resultatele f-tiei MPI_Gather ");
for(int i=0;i<numtask;i++)
```

```
      {
      printf("\n");
      for(int j=0;j<numtask;j++)
      printf("A[%d,%d]=%5.2f ", i,j, A[i*numtask+j]);
      }
      printf("\n");
      MPI_Barrier(MPI_COMM_WORLD);
      }
      else MPI_Barrier(MPI_COMM_WORLD);
MPI_Reduce(Rows,Max_Col,numtask, MPI_DOUBLE,MPI_MAX,root, MPI_COMM_WORLD);
if (myrank==root) {
for(int i=0;i<numtask;i++)
     {
       printf("\n");
       printf("Elementul maximal de pe coloana %d = %5.2f ",i,Max_Col[i]);
      }
      printf("\n");
      MPI_Barrier(MPI_COMM_WORLD); }
      else MPI_Barrier(MPI_COMM_WORLD);
 MPI_Finalize();
return 0;
}
```

Rezultatele posibile ale executării programului:

```
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC -o Exemplu _3_4_5a.exe Exemplu _3_4_5a.cpp
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpirun -n 5 -machinefile ~/nodes6 Exemplu _3_4_5a.exe
=====REZULTATUL PROGRAMULUI 'Exemplu_3_4_5a.exe'
Tipar datele initiale ale procesului cu rankul 0
Rows[0]= 1.98 Rows[1]= 1.97 Rows[2]= 0.42 Rows[3]= 0.49 Rows[4]= 2.04
Tipar datele initiale ale procesului cu rankul 1
Rows[0]= 0.60 Rows[1]= 0.71 Rows[2]= 2.15 Rows[3]= 1.18 Rows[4]= 0.82
Tipar datele initiale ale procesului cu rankul 2
Rows[0]= 1.38 Rows[1]= 1.59 Rows[2]= 1.73 Rows[3]= 0.80 Rows[4]= 1.76
Tipar datele initiale ale procesului cu rankul 3
Rows[0]= 0.00 Rows[1]= 0.34 Rows[2]= 0.24 Rows[3]= 1.49 Rows[4]= 1.63
Tipar datele initiale ale procesului cu rankul 4
Rows[0]= 1.84 Rows[1]= 1.22 Rows[2]= 1.96 Rows[3]= 1.10 Rows[4]= 0.40

Resultatele f-tiei MPI_Gather
A[0,0]= 1.98 A[0,1]= 1.97 A[0,2]= 0.42 A[0,3]= 0.49 A[0,4]= 2.04
A[1,0]= 0.60 A[1,1]= 0.71 A[1,2]= 2.15 A[1,3]= 1.18 A[1,4]= 0.82
A[2,0]= 1.38 A[2,1]= 1.59 A[2,2]= 1.73 A[2,3]= 0.80 A[2,4]= 1.76
A[3,0]= 0.00 A[3,1]= 0.34 A[3,2]= 0.24 A[3,3]= 1.49 A[3,4]= 1.63
A[4,0]= 1.84 A[4,1]= 1.22 A[4,2]= 1.96 A[4,3]= 1.10 A[4,4]= 0.40

Elementul maximal de pe coloana 0 =  1.98
Elementul maximal de pe coloana 1 =  1.97
Elementul maximal de pe coloana 2 =  2.15
Elementul maximal de pe coloana 3 =  1.49
Elementul maximal de pe coloana 4 =  2.04
```

[Hancu_B_S@hpc Finale]$

Exemplul 5.3 *Utilizând funcția* **`MPI_Reduce`** *și operațiile* **`MPI_MAX`**, **`MPI_MIN`**, *să se determine elementele maximale de pe liniile și coloanele unei matrice de dimensiuni arbitrare. Elementele matricei sunt inițializate de procesul cu rankul 0.*

Indicații: Fiecare proces MPI va executa o singură dată funcția **`MPI_Reduce.`** Pentru aceasta în programul de mai jos a fost elaborată funcția **`reduceLines`** în care se realizează următoarele. Fie că dimensiunea matricei A este $n \times m$ și procesul cu rankul k, în baza funcției **`MPI_Scatterv`** a recepționat l_k linii, adică

$a_{r1} \ \dots \ a_{rm}$	$a_{r+11} \ \dots \ a_{r+1m}$	$\dots$ $a_{r+l_k1} \ \dots \ a_{r+l_km}$

Atunci procesul k construiește următorul vector de lungimea m:

$\max\{a_{r1}, a_{r+11}, \dots, a_{r+l_k1}\}$	$\dots$	$\max\{a_{rm}, a_{r+1m}, \dots, a_{r+l_km}\}$,

care și este utilizat în funcția **`MPI_Reduce.`**

Mai jos este prezentat codul programului în limbajul C++ în care se realizează cele menționate în exemplul 5.3.

```
#include<mpi.h>
#include<stdio.h>
#include <iostream>
using namespace std;
void calculateSendcountsAndDispls(int rows, int cols, int size, int sendcounts[], int displs[])
{
 int rowsPerProc = rows / size;
int remainRows = rows % size;
int currDispl = 0;
for (int i = 0; i < size; ++i)
{
      displs[i] = currDispl;
      if (i < remainRows)
      sendcounts[i] = (rowsPerProc + 1)*cols; else
      sendcounts[i] = rowsPerProc * cols;
      currDispl += sendcounts[i];
      }
}
void invertMatrix(double *m, int mRows, int mCols, double *rez)
{
      for (int i = 0; i < mRows; ++i)
       for (int j = 0; j < mCols; ++j)
       rez[j * mRows + i] = m[i * mCols + j];
}
void reduceLines(double* Rows, int reccount, int cols, double myReducedRow[], bool min)
{
      int myNrOfLines = reccount / cols;
      for (int i = 0; i < cols; ++i)
         {
      double myMaxPerCol_i = Rows[i];
      for (int j = 1; j < myNrOfLines; ++j)
      {
      if (min)
```

```
      {
      if (Rows[j * cols + i]<myMaxPerCol_i)
      myMaxPerCol_i = Rows[j * cols + i];
         }
      else
       {
        if (Rows[j * cols + i] > myMaxPerCol_i)
      myMaxPerCol_i = Rows[j * cols + i];
        }
      }
      myReducedRow[i] = myMaxPerCol_i;
      }
}
int main(int argc, char *argv[])
{
int size,reccount, source;
double *A;
int myrank, root=0;
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
int sendcounts[size], displs[size];
int rows, cols;
double *Rows;
      if(myrank==root)
      {
printf("\n=====REZULTATUL PROGRAMULUI '%s' \n",argv[0]);
      cout << "Introduceti nr. de rinduri a matricei: ";
      cin >> rows;
      cout << "Introduceti nr. de coloane a matricei: ";
      cin >> cols;
      A=(double*)malloc(rows*cols*sizeof
      (double));
       for(int i=0;i<rows*cols;i++)
      A[i]=rand()/1000000000.0;
         printf("Tipar datele initiale\n");
         for(int i=0;i<rows;i++)
         {
                  printf("\n");
                  for(int j=0;j<cols;j++)
      printf("A[%d,%d]=%5.2f ",i,j,A[i * cols + j]);
         }
         printf("\n");
         MPI_Barrier(MPI_COMM_WORLD);
      }
      else
      MPI_Barrier(MPI_COMM_WORLD);
MPI_Bcast(&rows, 1, MPI_INT, root, MPI_COMM_WORLD);
MPI_Bcast(&cols, 1, MPI_INT, root, MPI_COMM_WORLD);
```

```
if (rows >= size)
	{
	calculateSendcountsAndDispls(rows, cols, size, sendcounts, displs);
	}
	else
	{
	cout << "Introduceti un numar de linii >= nr de procese." << endl;
	MPI_Finalize();
	return 0;
	}
reccount = sendcounts[myrank];
Rows = new double[reccount];
MPI_Scatterv(A, sendcounts, displs, MPI_DOUBLE, Rows, reccount, MPI_DOUBLE, root,
	MPI_COMM_WORLD);
MPI_Barrier(MPI_COMM_WORLD);
if(myrank==root) cout << "Liniile matricei au fost repartizate astfel:" << endl;
cout << "\nProces " << myrank << " - " << reccount / cols << " liniile" << endl;
double myReducedRow[cols];
reduceLines(Rows, reccount, cols, myReducedRow, false);
double* maxPerCols;
if (myrank == root)
maxPerCols = new double[cols];
MPI_Reduce(myReducedRow,maxPerCols,
	cols,MPI_DOUBLE,MPI_MAX,root, MPI_COMM_WORLD);
if (myrank == root)
{
printf("\nValorile de maxim pe coloanele matricii sunt:\n");
for (int i = 0; i < cols; ++i)
printf("Coloana %d - %.2f\n", i, maxPerCols[i]);
delete[] maxPerCols;
}
double *invMatr;
if (myrank == root)
{
invMatr = new double[cols * rows];
invertMatrix(A, rows, cols, invMatr);
}
if (cols >= size)
{
calculateSendcountsAndDispls(cols, rows, size, sendcounts, displs);
}
else
{
cout << "Introduceti un numar de coloane >= nr de procese." << endl;
MPI_Finalize();
return 0;
	}
reccount = sendcounts[myrank];
delete[] Rows;
```

```
Rows = new double[reccount];
MPI_Scatterv(invMatr, sendcounts, displs, MPI_DOUBLE, Rows, reccount, MPI_DOUBLE, root,
      MPI_COMM_WORLD);
double myReducedCol[rows];
reduceLines(Rows, reccount, rows, myReducedCol, true);
double* minPerRows;
if (myrank == root)
minPerRows = new double[rows];
MPI_Reduce(myReducedCol,minPerRows,
      rows,MPI_DOUBLE,MPI_MIN,root,
      MPI_COMM_WORLD);
if (myrank == root)
{
printf("\nValorile de minim pe liniile matricii sunt:\n");
for (int i = 0; i < rows; ++i)
printf("Rindul %d - %.2f\n", i, minPerRows[i]);
delete[] minPerRows;
free(A);
}
MPI_Finalize();
delete[] Rows;
return 0;
}
```

Rezultatele posibile ale executării programului:

```
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC -o Exemplu _3_4_6.exe Exemplu _3_4_6.cpp
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpirun -n 3 -machinefile ~/nodes4  Exemplu _3_4_6.exe

=====REZULTATUL PROGRAMULUI 'Exemplu _3_4_6.exe'
Introduceti nr. de rinduri a matricei: 7
Introduceti nr. de coloane a matricei: 6
Tipar datele initiale
A[0,0]= 1.80 A[0,1]= 0.85 A[0,2]= 1.68 A[0,3]= 1.71 A[0,4]= 1.96 A[0,5]= 0.42
A[1,0]= 0.72 A[1,1]= 1.65 A[1,2]= 0.60 A[1,3]= 1.19 A[1,4]= 1.03 A[1,5]= 1.35
A[2,0]= 0.78 A[2,1]= 1.10 A[2,2]= 2.04 A[2,3]= 1.97 A[2,4]= 1.37 A[2,5]= 1.54
A[3,0]= 0.30 A[3,1]= 1.30 A[3,2]= 0.04 A[3,3]= 0.52 A[3,4]= 0.29 A[3,5]= 1.73
A[4,0]= 0.34 A[4,1]= 0.86 A[4,2]= 0.28 A[4,3]= 0.23 A[4,4]= 2.15 A[4,5]= 0.47
A[5,0]= 1.10 A[5,1]= 1.80 A[5,2]= 1.32 A[5,3]= 0.64 A[5,4]= 1.37 A[5,5]= 1.13
A[6,0]= 1.06 A[6,1]= 2.09 A[6,2]= 0.63 A[6,3]= 1.66 A[6,4]= 1.13 A[6,5]= 1.65
Liniile matricei au fost repartizate astfel:
Proces 2 - 2 liniile
Proces 0 - 3 liniile
Proces 1 - 2 liniile
Valorile de maxim pe coloanele matricii sunt:
Coloana 0 - 1.80
Coloana 1 - 2.09
Coloana 2 - 2.04
Coloana 3 - 1.97
Coloana 4 - 2.15
Coloana 5 - 1.73
```

```
Valorile de minim pe liniile matricii sunt:
Rindul 0 - 0.42
Rindul 1 - 0.60
Rindul 2 - 0.78
Rindul 3 - 0.04
Rindul 4 - 0.23
Rindul 5 - 0.64
Rindul 6 - 0.63
[Hancu_B_S@hpc Finale]$
```

Vom ilustra utilizarea operațiilor globale de reducere **`MPI_MAXLOC`** prin exemplul ce urmează.

Exemplu 5.4 *Utilizând funcția* **`MPI_Reduce`** *și operațiile* **`MPI_MAXLOC`** *să se determine elementele maximale de pe coloane și indicele liniei, unei matrice pătrate (dimensiunea este egală cu numărul de procese). Elementele matricei sunt inițializate de procesul cu rankul 0.*

Mai jos este prezentat codul programului în limbajul C++ în care se realizează cele menționate în exemplul 5.4.

```
#include<stdio.h>
int main(int argc, char *argv[])
{
 int numtask,sendcount,reccount,source;
double *A;
int i, myrank, root=1;
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD, &numtask);
double ain[numtask], aout[numtask];
int ind[numtask];
struct {
double val;
int rank;
        } în[numtask], out[numtask];
          sendcount=numtask;
          reccount=numtask;
if(myrank==root)
 {
printf("===== Rezultatele programului '%s' =====\n",argv[0]);
A=(double*)malloc(numtask*numtask*sizeof(double));
 for(int i=0;i<numtask*numtask;i++)
 A[i]=rand()/1000000000.0;
 printf("Tipar datele initiale\n");
 for(int i=0;i<numtask;i++)
      {
      printf("\n");
      for(int j=0;j<numtask;j++)
      printf("A[%d,%d]=%.2f ",i,j, A[i*numtask+j]);
      }
      printf("\n");
      MPI_Barrier(MPI_COMM_WORLD);
 }
```

```
 else MPI_Barrier(MPI_COMM_WORLD);
MPI_Scatter(A, sendcount, MPI_DOUBLE,ain, reccount, MPI_DOUBLE, root, MPI_COMM_WORLD);
for (i=0; i<numtask; ++i)
      {
      în[i].val = ain[i];
      în[i].rank = myrank;
      }
MPI_Reduce(în,out,numtask,MPI_DOUBLE_
      INT, MPI_MAXLOC, root, MPI_COMM_WORLD);
if (myrank == root)
      {printf("\n");
      printf("Valorile maximale de pe coloane și indicele liniei:\n");
      for (i=0; i<numtask; ++i) {
      aout[i] = out[i].val;
      ind[i] = out[i].rank;
      printf("Coloana %d, valoarea=  %.2f, linia= %d\n",i, aout[i],ind[i]); }
      }
MPI_Finalize();
return 0;
}
```

Rezultatele posibile ale executării programului:

```
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpiCC -o Exemplu_3_4_7.exe Exemplu_3_4_7.cpp
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpirun -n 6 -machinefile ~/nodes4  Exemplu_3_4_7.exe
===== Rezultatele programului 'Exemplu_3_4_7.exe' =====
Tipar datele initiale

A[0,0]=1.80 A[0,1]=0.85 A[0,2]=1.68 A[0,3]=1.71 A[0,4]=1.96 A[0,5]=0.42
A[1,0]=0.72 A[1,1]=1.65 A[1,2]=0.60 A[1,3]=1.19 A[1,4]=1.03 A[1,5]=1.35
A[2,0]=0.78 A[2,1]=1.10 A[2,2]=2.04 A[2,3]=1.97 A[2,4]=1.37 A[2,5]=1.54
A[3,0]=0.30 A[3,1]=1.30 A[3,2]=0.04 A[3,3]=0.52 A[3,4]=0.29 A[3,5]=1.73
A[4,0]=0.34 A[4,1]=0.86 A[4,2]=0.28 A[4,3]=0.23 A[4,4]=2.15 A[4,5]=0.47
A[5,0]=1.10 A[5,1]=1.80 A[5,2]=1.32 A[5,3]=0.64 A[5,4]=1.37 A[5,5]=1.13
Valorile maximale de pe coloane și indicele liniei:
Coloana 0, valoarea=  1.80, linia= 0
Coloana 1, valoarea=  1.80, linia= 5
Coloana 2, valoarea=  2.04, linia= 2
Coloana 3, valoarea=  1.97, linia= 2
Coloana 4, valoarea=  2.15, linia= 4
Coloana 5, valoarea=  1.73, linia= 3
[Hancu_B_S@hpc]$ /opt/openmpi/bin/mpirun -n 8 -machinefile ~/nodes4  Exemplu_3_4_7.exe
===== Rezultatele programului 'Exemplu_3_4_7.exe' =====
Tipar datele initiale
A[0,0]=1.80 A[0,1]=0.85 A[0,2]=1.68 A[0,3]=1.71 A[0,4]=1.96 A[0,5]=0.42 A[0,6]=0.72 A[0,7]=1.65
A[1,0]=0.60 A[1,1]=1.19 A[1,2]=1.03 A[1,3]=1.35 A[1,4]=0.78 A[1,5]=1.10 A[1,6]=2.04 A[1,7]=1.97
A[2,0]=1.37 A[2,1]=1.54 A[2,2]=0.30 A[2,3]=1.30 A[2,4]=0.04 A[2,5]=0.52 A[2,6]=0.29 A[2,7]=1.73
A[3,0]=0.34 A[3,1]=0.86 A[3,2]=0.28 A[3,3]=0.23 A[3,4]=2.15 A[3,5]=0.47 A[3,6]=1.10 A[3,7]=1.80
A[4,0]=1.32 A[4,1]=0.64 A[4,2]=1.37 A[4,3]=1.13 A[4,4]=1.06 A[4,5]=2.09 A[4,6]=0.63 A[4,7]=1.66
A[5,0]=1.13 A[5,1]=1.65 A[5,2]=0.86 A[5,3]=1.91 A[5,4]=0.61 A[5,5]=0.76 A[5,6]=1.73 A[5,7]=1.97
A[6,0]=0.15 A[6,1]=2.04 A[6,2]=1.13 A[6,3]=0.18 A[6,4]=0.41 A[6,5]=1.42 A[6,6]=1.91 A[6,7]=0.75
```

```
A[7,0]=0.14 A[7,1]=0.04 A[7,2]=0.98 A[7,3]=0.14 A[7,4]=0.51 A[7,5]=2.08 A[7,6]=1.94 A[7,7]=1.83
Valorile maximale de pe coloane și indicele liniei:
Coloana 0, valoarea=  1.80, linia= 0
Coloana 1, valoarea=  2.04, linia= 6
Coloana 2, valoarea=  1.68, linia= 0
Coloana 3, valoarea=  1.91, linia= 5
Coloana 4, valoarea=  2.15, linia= 3
Coloana 5, valoarea=  2.09, linia= 4
Coloana 6, valoarea=  2.04, linia= 1
Coloana 7, valoarea=  1.97, linia= 5
[Hancu_B_S@hpc Notate_Exemple]$
```