MPI – Message Passing Interface

Comunicatii one-to-one blocante

Comunicatii one-to-one non-blocante

Comunicatii colective

MPI – Message Passing Interface

Comunicatie Point to Point

Send, fara zona tampon: **MPI_Send(buffer,count,type,dest,tag,comm)**
Send, cu zona tampon : **MPI_Isend(buffer,count,type,dest,tag,comm,request)**
Receive, cu zona tampon : **MPI_Recv(buffer,count,type,source,tag,comm,status)**
Receive, fara zona tampon : **MPI_Irecv(buffer,count,type,source,tag,comm,request)**

buffer: spatiul de adrese al aplicatiei care refera data care trebuie sa fie trimisa sau primita;

count: numarul de elemente de date de un anumit tip care trebuie trimise sau primite;

type: tipul de date care trebuie trimise sau primite;

dest: argument pentru rutinele Send care indica rank-ul procesului caruia i se adreseaza mesajul;

source: argument pentru rutinele Recv care indica rank-ul procesului de la care se primeste mesajul; daca se specifica MPI_ANY_SOURCE, mesajul poate fi primit de la orice sursa;

tag: identificator unic pentru mesaj;

comm: specifica un comunicator sau un set de procese pentru care campurile sursa sau destinatie sunt valide;

status: in C, e un pointer catre o structura MPI_Status;

request: folosit de metode Send si Recv neblocante.

MPI – Message Passing Interface

Tipuri de date MPI

Tipuri MPI	Tipuri C
MPI_CHAR	signed char
MPI_WCHAR	wchar_t - wide character
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_LONG_INT	signed long long int
MPI_SIGNED_CHAR	signed char
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_C_COMPLEX	float _Complex
MPI_C_DOUBLE_COMPLEX	double _Complex
MPI_C_LONG_DOUBLE_COMPLEX	long double _Complex
MPI_C_BOOL	_Bool
MPI_C_LONG_DOUBLE_COMPLEX	long double _Complex
MPI_INT8_T	int8_t
MPI_INT16_T	int16_t
MPI_INT32_T	int32_t
MPI_INT64_T	int64_t
MPI_UINT8_T	uint8_t
MPI_UINT16_T	uint16_t
MPI_UINT32_T	uint32_t
MPI_UINT64_T	uint64_t
MPI_BYTE	8 binary digits
MPI_PACKED	data packed or unpacked with MPI_Pack()/ MPI_Unpack

MPI – Message Passing Interface

Exemplu de comunicatie Point to Point blocanta

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

int main(argc,argv)
int argc;
char *argv[];
{
    int numtasks, rank, dest, source, rc, count, tag=1;
    int prev, next;
    char inmsg, outmsg='x';
    MPI_Status Stat;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if(numtasks > 1) {
        prev = rank-1;
        next = rank+1;
        if (rank == 0)  prev = numtasks - 1;
        if (rank == (numtasks - 1))  next = 0;

        dest = next;
        source = prev;
        rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
        rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);

        rc = MPI_Get_count(&Stat, MPI_CHAR, &count);
        printf("Task %d: Received %d char(s) from task %d with tag %d \n",
               rank, count, Stat.MPI_SOURCE, Stat.MPI_TAG);
    }
    else
        printf("At least two processors are needed\n");

    MPI_Finalize();
}
```

Se realizeaza o topologie virtuala in inel, fiecare vecin din stanga trimitand procesorului curent un caracter si fiecare vecin din dreapta primind de la procesorul curent un caracter ('x')

C:\WINDOWS\system32\cmd.exe

```
C:\Program Files\MPICH2\exex>..\bin\mpiexec.exe -localonly 1 sendRecvEx.exe
At least two processors are needed

C:\Program Files\MPICH2\exex>..\bin\mpiexec.exe -localonly 2 sendRecvEx.exe
Task 1: Received 1 char(s) from task 0 with tag 1
Task 0: Received 1 char(s) from task 1 with tag 1

C:\Program Files\MPICH2\exex>..\bin\mpiexec.exe -localonly 4 sendRecvEx.exe
Task 0: Received 1 char(s) from task 3 with tag 1
Task 1: Received 1 char(s) from task 0 with tag 1
Task 2: Received 1 char(s) from task 1 with tag 1
Task 3: Received 1 char(s) from task 2 with tag 1

C:\Program Files\MPICH2\exex>..\bin\mpiexec.exe -localonly 8 sendRecvEx.exe
Task 1: Received 1 char(s) from task 0 with tag 1
Task 7: Received 1 char(s) from task 6 with tag 1
Task 0: Received 1 char(s) from task 7 with tag 1
Task 3: Received 1 char(s) from task 2 with tag 1
Task 6: Received 1 char(s) from task 5 with tag 1
Task 5: Received 1 char(s) from task 4 with tag 1
Task 2: Received 1 char(s) from task 1 with tag 1
Task 4: Received 1 char(s) from task 3 with tag 1
```

MPI – Message Passing Interface

Exemplu de comunicatie Point to Point non-blocanta

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

int main(argc,argv)
int argc;
char *argv[];
{
    int numtasks, rank, next, prev, buf[2], tag1=1, tag2=2;
    int payload;
    MPI_Request reqs[4];
    MPI_Status stats[2];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if(numtasks > 1) {
        prev = rank-1;
        next = rank+1;
        if (rank == 0)  prev = numtasks - 1;
        if (rank == (numtasks - 1))  next = 0;

        MPI_Irecv(&buf[0], 1, MPI_INT, prev, tag1, MPI_COMM_WORLD, &reqs[0]);
        MPI_Irecv(&buf[1], 1, MPI_INT, next, tag2, MPI_COMM_WORLD, &reqs[1]);

        payload = 100 * rank;
        MPI_Isend(&payload, 1, MPI_INT, prev, tag2, MPI_COMM_WORLD, &reqs[2]);
        MPI_Isend(&payload, 1, MPI_INT, next, tag1, MPI_COMM_WORLD, &reqs[3]);

        MPI_Waitall(4, reqs, stats);
        printf("Task %d: Received payload int value %03d from task %d\n", rank, buf[0],prev);
        printf("Task %d: Received payload int value %03d from task %d\n", rank, buf[1],next);
    }
    else
        printf("At least two processors are needed\n");

    MPI_Finalize();
}
```

C:\WINDOWS\system32\cmd.exe

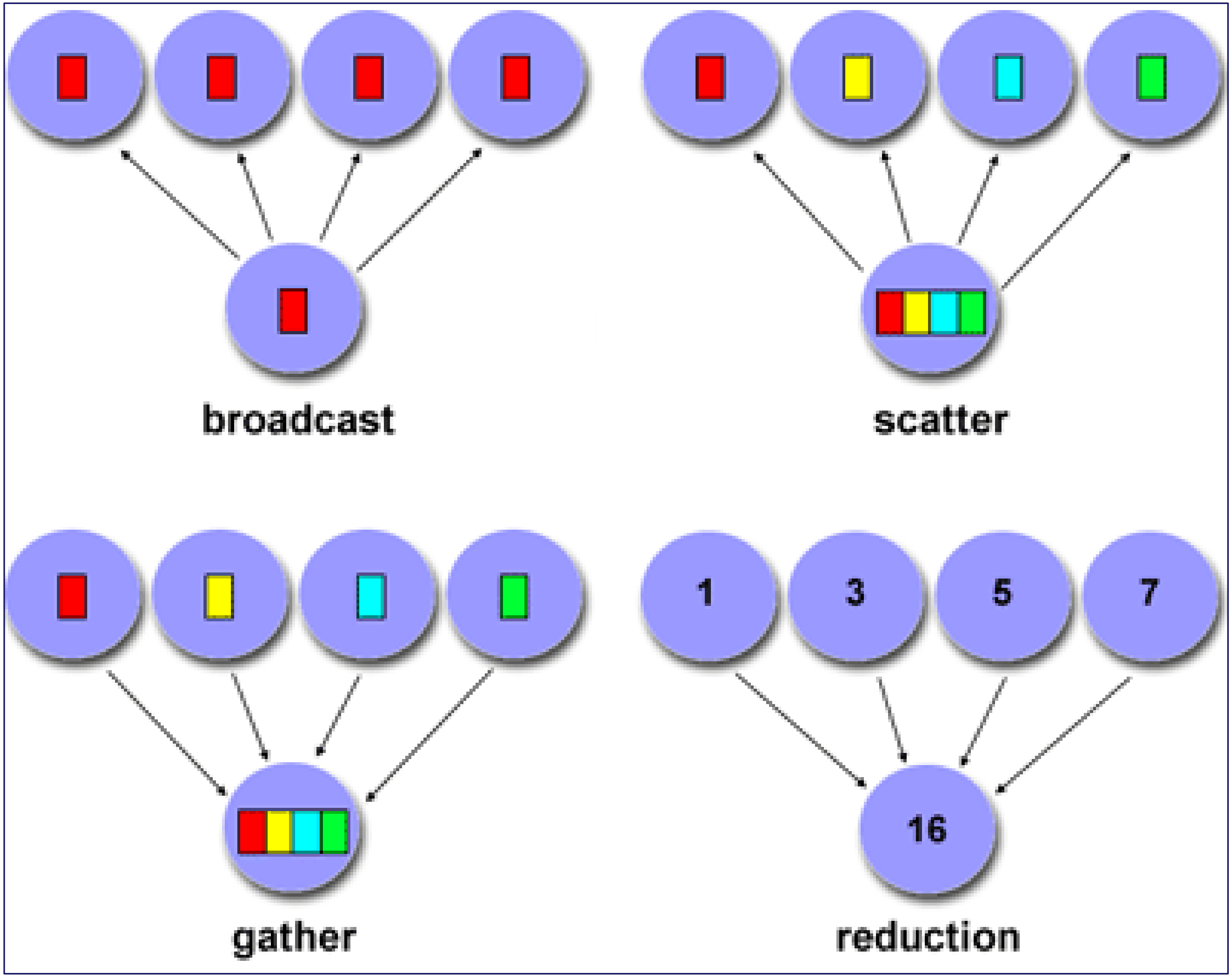
```
C:\Program Files\MPICH2\exex>..\bin\mpiexec.exe -localonly 8 sendRecvNonBlockEx.
exe
Task 0: Received payload int value 700 from task 7
Task 0: Received payload int value 100 from task 1
Task 1: Received payload int value 000 from task 0
Task 1: Received payload int value 200 from task 2
Task 2: Received payload int value 100 from task 1
Task 2: Received payload int value 300 from task 3
Task 6: Received payload int value 500 from task 5
Task 6: Received payload int value 700 from task 7
Task 5: Received payload int value 400 from task 4
Task 5: Received payload int value 600 from task 6
Task 3: Received payload int value 200 from task 2
Task 3: Received payload int value 400 from task 4
Task 7: Received payload int value 600 from task 6
Task 7: Received payload int value 000 from task 0
Task 4: Received payload int value 300 from task 3
Task 4: Received payload int value 500 from task 5

C:\Program Files\MPICH2\exex>
```

MPI – Message Passing Interface

Comunicatie colectiva

- Comunicatia colectiva implica toate procesele din spatiul unui comunicator. Toate procesele sunt membre in comunicatorul ***MPI_COMM_WORLD***;
- Programatorul are responsabilitatea sa se asigure ca toate procesele dintr-un comunicator participa in oricare din comunicatiile colective;
- Tipurile de operatii colective:
  - Sincronizare: procesele asteapta pana cand toti membrii grupului ajung in punctul de sincronizare;
  - Deplasarea datelor: broadcast, scatter/gather, all to all;
  - Calcul colectiv (reduction): un membru al grupului colecteaza date de la ceilalti membri ai grupului si executa o operatie (minim, maxim, adunare, inmultire);
- Consideratii privind programarea si restrictii:
  - Operatiile colective sunt blocante;
  - Rutinele de comunicatie colectiva nu au argumente de tip tag;
  - Operatii colective intre submultimi ale proceselor se realizeaza prin impartirea prealabila in submultimi ale grupurilor de procese si atasarea submultimilor la comunicatori diferiti;
  - Operatiile colective pot fi utilizate numai cu tipuri de date MPI predefinite (nu si tipuri de date derivate);

MPI – Message Passing Interface



MPI – Message Passing Interface

MPI_Barrier (comm)

Creaza o bariera de sincronizare intr-un grup (specificat de parametrul comm); fiecare proces, odata ajuns in punctul de executie corespunzator barierei, se blocheaza pana cand toate task-urile ajung in acelasi punct.

MPI_Bcast (&buffer,count,datatype,root,comm)

Trimite un mesaj de la procesul cu rangul ***root*** la toate celelalte procese din grup;

MPI_Scatter (&sendbuf,sendcnt,sendtype,&recvbuf, recvcnt,recvtype,root,comm)

Distribuie mesaje distincte dintr-un singur task sursa fiecarui task din grup

MPI_Gather (&sendbuf,sendcnt,sendtype,&recvbuf, recvcount,recvtype,root,comm)

Reuneste (gather) mesaje de la fiecare task din grup intr-un singur task destinatie. Aceasta rutina realizeaza operatia inversa a rutinei **MPI_Scatter.**

MPI_Allgather (&sendbuf,sendcount,sendtype,&recvbuf, recvcount,recvtype,comm)

Concatenarea datelor tuturor taskurilor din grup. Fiecare task din grup realizeaza un broadcast unul-la-toti in interiorul grupului.

MPI_Reduce (&sendbuf,&recvbuf,count,datatype,op,root,comm)

Aplica o operatie de reducere asupra tuturor taskurilor din grup si plaseaza rezultatul unui task.

Tipuri operatii de reducere

Tip operatie MPI	Semnificatie	Tip C la care se aplica
MPI_MAX	maximum	integer, float
MPI_MIN	minimum	integer, float
MPI_SUM	suma	integer, float
MPI_PROD	produs	integer, float
MPI_LAND	AND logic	integer
MPI_BAND	bit-wise AND	integer, MPI_BYTE
MPI_LOR	OR logic	integer
MPI_BOR	bit-wise OR	integer, MPI_BYTE
MPI_LXOR	XOR logic	integer
MPI_BXOR	bit-wise XOR	integer, MPI_BYTE
MPI_MAXLOC	valoarea maxima si locatie	float, double and long double
MPI_MINLOC	valoarea minima si locatie	float, double and long double

MPI – Message Passing Interface

Exemplu de comunicatie colectiva: scatter

```c
#include "mpi.h"
#include <stdio.h>
#define SIZE 4

int main(argc,argv)
int argc;
char *argv[];
{
    int numtasks, rank, sendcount, recvcount, source;
    float sendbuf[SIZE][SIZE] = {
      {1.0, 2.0, 3.0, 4.0},
      {5.0, 6.0, 7.0, 8.0},
      {9.0, 10.0, 11.0, 12.0},
      {13.0, 14.0, 15.0, 16.0}  };
    float recvbuf[SIZE];

    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

    if (numtasks == SIZE)
    {
        source = 1;
        sendcount = SIZE;
        recvcount = SIZE;
        MPI_Scatter(sendbuf,sendcount,MPI_FLOAT,recvbuf,recvcount,
                  MPI_FLOAT,source,MPI_COMM_WORLD);

        printf("rank= %d  Results: %2.1f %2.1f %2.1f %2.1f\n",rank,recvbuf[0],
             recvbuf[1],recvbuf[2],recvbuf[3]);
    }
    else
        printf("Must specify %d processors. Terminating.\n",SIZE);

    MPI_Finalize();
}
```

C:\WINDOWS\system32\cmd.exe

```
C:\Program Files\MPICH2\exex>..\bin\mpiexec.exe -localonly 2 scatterEx.exe
Must specify 4 processors. Terminating.
Must specify 4 processors. Terminating.

C:\Program Files\MPICH2\exex>..\bin\mpiexec.exe -localonly 4 scatterEx.exe
rank= 0  Results: 1.0 2.0 3.0 4.0
rank= 3  Results: 13.0 14.0 15.0 16.0
rank= 2  Results: 9.0 10.0 11.0 12.0
rank= 1  Results: 5.0 6.0 7.0 8.0

C:\Program Files\MPICH2\exex>_
```

MPI – Message Passing Interface

Exemplu de comunicatie colectiva: Bcast si Reduce

```
/* -*- Mode: C; c-basic-offset:4 ; -*- */
/*
 *   (C) 2001 by Argonne National Laboratory.
 *       See COPYRIGHT in top-level directory.
 */

/* This is an interactive version of cpi */
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f(double);

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 = 0.0, 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);


    fprintf(stdout,"Process %d of %d is on %s\n",
        myid, numprocs, processor_name);
    fflush(stdout);
```

Se calculeaza valoarea lui PI prin impartirea intervalului introdus de utlizator intre procesele din comunicator si la sfarsit se insumeaza (utilizand Reduce) valorile calculate de fiecare procesor.

MPI – Message Passing Interface

Exemplu de comunicatie colectiva: Bcast si Reduce (continuare)

```
    while (!done) {
        if (myid == 0) {
            fprintf(stdout, "Enter the number of intervals: (0 quits) ");
            fflush(stdout);
            if (scanf("%d",&n) != 1) {
                fprintf( stdout, "No number entered; quitting\n" );
                n = 0;
            }
            startwtime = MPI_Wtime();
        }
        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;
            printf("Task %d contribution to PI is: %.16f\n",myid,mypi);

            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);
                fflush( stdout );
            }
        }
    }
    MPI_Finalize();
    return 0;
}
```

MPI – Message Passing Interface

Exemplu de comunicatie colectiva: Bcast si Reduce (continuare)

Rezultatele executiei fiecarui task sunt adunate (functia MPI_SUM); in exemplul curent, s-au rulat 16 task-uri, folosindu-se 64 de intervale pentru aproximarea lui PI.

```
C:\WINDOWS\system32\cmd.exe - ..\bin\mpiexec.exe -localonly 16 cpi.exe

C:\Program Files\MPICH2\exex>..\bin\mpiexec.exe -localonly 16 cpi.exe
Process 3 of 16 is on ADVANTEC4
Process 10 of 16 is on ADVANTEC4
Process 9 of 16 is on ADVANTEC4
Process 7 of 16 is on ADVANTEC4
Process 0 of 16 is on ADVANTEC4
Enter the number of intervals: (0 quits) Process 5 of 16 is on ADVANTEC4
Process 8 of 16 is on ADVANTEC4
Process 12 of 16 is on ADVANTEC4
Process 14 of 16 is on ADVANTEC4
Process 6 of 16 is on ADVANTEC4
Process 4 of 16 is on ADVANTEC4
Process 11 of 16 is on ADVANTEC4
Process 1 of 16 is on ADVANTEC4
Process 13 of 16 is on ADVANTEC4
Process 2 of 16 is on ADVANTEC4
Process 15 of 16 is on ADVANTEC4
64
Task 3 contribution to PI is: 0.205234759602137O
Task 2 contribution to PI is: 0.207030655405976O
Task 0 contribution to PI is: 0.210488613861743 8
Task 1 contribution to PI is: 0.2087827189378362
Task 10 contribution to PI is: 0.1916383646673934
Task 11 contribution to PI is: 0.1895788666762351
Task 8 contribution to PI is: 0.1956817294289781
Task 9 contribution to PI is: 0.1936734653427186
Task 13 contribution to PI is: 0.1853963437924781
Task 12 contribution to PI is: 0.1874973935077794
Task 4 contribution to PI is: 0.2033974039569982
Task 5 contribution to PI is: 0.2015209905334091
Task 14 contribution to PI is: 0.1832780864873025
Task 15 contribution to PI is: 0.1811449549581153
Task 6 contribution to PI is: 0.1996079444252699
Task 7 contribution to PI is: 0.1976607070574779
pi is approximately 3.1416129986418486, Error is 0.0000203450520555
wall clock time = 0.056699
Enter the number of intervals: (0 quits) _
```