MPI - Message Passing Interface

MPI este un **standard** pentru comunicarea prin mesaje
Elaborat de MPI Forum.
Sisteme anterioare:
- de la IBM, Intel (NX/2) Express, nCUBE (Vertex), PARMACS, Zipcode, Chimp, PVM, Chameleon, PICL.

MPI **are la bază** modelul proceselor comunicante prin mesaje.
MPI **este o bibliotecă** nu un limbaj.
MPI specifică reguli de apel pentru:
- C, C++, FORTRAN.77, FORTRAN90.

Istoric
- MPI 1 (1993) orientat pe comunicarea punct la punct.
- Alte versiuni – mai importanta MPI 1.2 (1997)
- MPI 2 (1997) include:
  - Procese dinamice
  - Comunicarea "one-sided"
  - Operatii de I/E paralele

Comunicarea punct la punct

Operaţii de bază

Forma clasică:
- send (adresa, lungime, destinatie, tag)
- recv (adresa, maxlung, sursă, tag, actlung)

In MPI:
MPI_SEND (buf, count, datatype, dest, tag, comm)
MPI_RECV (buf, count, datatype, source, tag, comm, status)

In C:

```
int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag,
                                   MPI_Comm comm);

int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source,
                int tag,  MPI_Comm comm, MPI_Status *status);
```

Elemente noi introduse de MPI

Tamponul de comunicatie este definit de tripleta
(adresa, contor, tip_de_date)

Contextul
- Fiecare comunicare de mesaj se derulează într-un anumit context.
- Mesajele caracterizate de **context** si **tag**
- Mesajele sunt intotdeauna primite în contextul în care au fost transmise.
- Mesajele transmise în contexte diferite nu interferă.

Grup de procese
- Contextul este partajat de un **grup de procese.**
- Fiecare proces are un rang (nr intreg de la 0 la n-1)
- Fiecare proces caracterizat de **grup** si **rang** in grup

Comunicator = Informaţiile de **contex**t şi de **grup**

Comunicator predefinit: MPI_COMM_WORLD

Un exemplu simplu

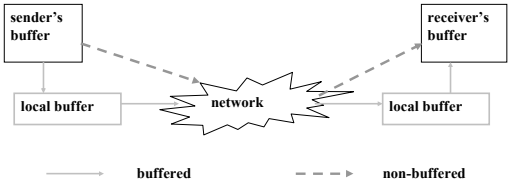
```
# include "mpi.h"
main (int argc, char **argv)
{ char message[40];
  int myrank;
  MPI_Status status;
  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &myrank);
  if (myrank==0)
     { strcpy (message, "Hello, there");
       MPI_Send (message, strlen(message),
            MPI_CHAR, 1, 99,MPI_COMM_WORLD);
     }
  else { MPI_Recv (message, 20, MPI_CHAR, 0,
            99, MPI_COMM_WORLD,&status);
         printf("\tReceived:%s \n", message);
       }
  MPI_Finalize();
}
```

Operaţii MPI

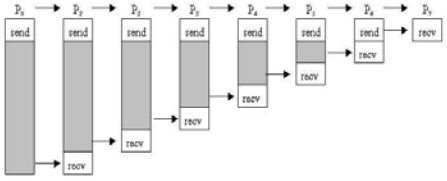
```
int MPI_Init (int *argc, char ***argv);
int MPI_Initialized(int *flag);
int MPI_Finalize (void);
int MPI_Comm_rank(MPI_Comm comm, int *rank);
```

Modurile de comunicare: cu şi fără tampon

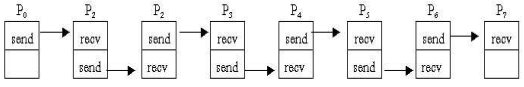
send si **receive** **sunt blocante**



Probleme de transmitere în modul standard



Ordonarea operaţiilor send-recv



Operaţii send-recv combinate

MPI_Sendrecv

```
int MPI_Sendrecv( void *sendbuf, int sendcount, MPI_Datatype
      sendtype, int dest, int sendtag, void *recvbuf, int recvcount,
      MPI_Datatype recvtype, int source, int recvtag, MPI_Comm
      comm, MPI_Status *status )
```

MPI_Sendrecv_replace

```
int MPI_Sendrecv_replace( void *buf, int count, MPI_Datatype
      datatype, int dest, int sendtag, int source, int recvtag,
      MPI_Comm comm, MPI_Status *status )
```

Transmitere prin tampon alocat explicit

```
#define BuffSize 100000
int size;
char *buff;
MPI_Buffer_attach(malloc(BuffSize), BuffSize);
/* Tamponul de BuffSize octeti poate fi folosit acum */
MPI_Bsend( ... );
MPI_Buffer_detach(&buff, &size);
/* Tamponul redus la zero */
MPI_Buffer_attach(buff, size);
/* Tamponul disponibil din nou */
```

```
int MPI_Buffer_attach( void *buffer, int size );
int MPI_Buffer_detach(void * bufferptr, int * size );
```

Transmitere sincronă, în mod pregătit, non-blocantă

Transmitere sincronă **MPI_Ssend**

Transmitere în modul pregătit **MPI_Rsend**

Comunicaţia non-blocantă

MPI_Isend **MPI_Test** **MPI_Wait**
MPI_Irecv

```
int MPI_Irecv( void *buf, int count, MPI_Datatype datatype, int
      source, int tag, MPI_Comm comm, MPI_Request *request )
```

MPI_Testall MPI_Testany MPI_Testsome
MPI_Waitall MPI_Waitany MPI_Waitsome

Evitarea întârzierii nelimitate

```
typedef struct {
        char data [MSIZE];
        int dsize;
} Buffer;
Buffer buf[];
MPI_Request req[];
MPI_Status status[];
int index[];
...
MPI_Comm_rank(comm, &rank);
MPI_Comm_size(comm, &size);
if(rank != size-1) {                          /* client code */
  buf = (Buffer *) malloc(sizeof(Buffer));
  while(1) {
    produce_request( buf->data, &buf->dsize);
    MPI_Send(buf->data, buf->dsize, MPI_CHAR, size-1, tag, comm);
        }
}
```

```
else
  { /* rank == size-1; server code */

  buf = (Buffer *) malloc(sizeof(Buffer)*(size-1));
  req = (MPI_Request *) malloc(sizeof(MPI_Request)*(size-1));
  status = (MPI_Status *) malloc(sizeof(MPI_Status)*(size-1));
  index = (int *) malloc(sizeof(int)*(size-1));
  for(i=0; i < size-1; i++)
    MPI_Irecv(buf[i].data, MSIZE, MPI_CHAR, i, tag, comm, &req[i]);
  while (1){
        MPI_Waitsome(size-1, req, &count, index, status);
        for (i=0; i < count; i++){
          j = index[i];
          MPI_Get_count(&status[i], MPI_CHAR, &buf[j].dsize);
          process_request(buf[j].data, buf[j].dsize);
          MPI_Irecv(buf[j].data, MSIZE, MPI_CHAR, j, tag, comm, &req[j]);
        }
  }
}
```

Cereri de comunicare persistente

Programatorul poate:

crea o cerere persistentă	**MPI_Send_init**, **MPI_Recv_init**
declansa operatia	**MPI_Start**, **MPI_Startall**
astepta terminarea	**MPI_Wait**, sau
dealoca cererea persistentă cu	**MPI_Request_free**.

Exemplu

```
MPI_request *request;
MPI_Recv_init (buf, count, datatype, source, tag, comm, request);
MPI_Start (request);          // are semantica MPI_Irecv
MPI_Wait (request, status);
MPI_Request_free (request);
```

```
int MPI_Start( MPI_Request *request)
int MPI_Wait ( MPI_Request *request, MPI_Status *status)
int MPI_Request_free( MPI_Request *request )
```

Tipuri de date în MPI

Tipuri predefinite

toate tipurile tipurile de bază din C (si din FORTRAN) plus MPI_BYTE si MPI_PACKED.

tip de date MPI	tip corespondent in C
MPI_CHAR MPI_SHORT MPI_INT	signed char signed short int signed int
MPI_LONG MPI_UNSIGNED_CHAR MPI_UNSIGNED_SHORT	signed long int unsigned char unsigned short int
MPI_UNSIGNED MPI_UNSIGNED_LONG MPI_FLOAT	unsigned int unsigned long int float
MPI_DOUBLE MPI_LONG_DOUBLE	double long double

Tipurile derivate

Tipurile derivate - trebuie construite; ex:

```
MPI_Type_contiguous (count, datatype, &newtype);
MPI_Type_commit (&newtype);
MPI_Send (buffer, 1, newtype, dest, tag, comm);
MPI_Type_free (&newtype);
```

Mecanisme pentru descrierea unui tip general de date.

harta tipului = secventa de perechi (tip, deplasare)

Typemap = $\{(type_0, disp_0), \ldots, (type_{n-1}, disp_{n-1})\}$

semnatura tipului = secventa tipurilor

Typesig = $\{type_0, \ldots, type_{n-1}\}$

titlu (handle)
extindere (extent)
adrese absolute = deplasari relative la "adresa zero" MPI_BOTTOM.

Cateva tipuri derivate

int MPI_Type_contiguous(int count, MPI_Datatype old_type, MPI_Datatype *newtype)

int MPI_Type_vector(int count, int blocklength, int stride, MPI_Datatype old_type, MPI_Datatype *newtype)

int MPI_Type_hvector(int count, int blocklength, MPI_Aint stride, MPI_Datatype old_type, MPI_Datatype *newtype)

int MPI_Type_indexed(int count, int array_of_blocklengths[], int array_of_indices[], MPI_Datatype old_type, MPI_Datatype *newtype)

int MPI_Type_hindexed(int count, int array_of_blocklengths[], MPI_Aint array_of_displacements[], MPI_Datatype old_type, MPI_Datatype *newtype)

int MPI_Type_create_struct(int count, int array_of_blocklengths[], MPI_Aint array_of_displacements[], MPI_Datatype array_of_types[], MPI_Datatype *newtype)

Exemplu

```
# define MAX_PART 1000
Struct Part_struct
 { int class;       /* clasa particulei */
   double d[6];     /* coordonatele particulei */
   char b[7];       /* alte informatii */
 };
struct Part_struct particle[MAX_PART];
int i, dest, rank;
MPI_Comm comm;
/* constructia tipului care descrie structura MPI a unei particule, ParticleType */
MPI_Datatype ParticleType;
MPI_Datatype type[3] = {MPI_INT, MPI_DOUBLE, MPI_CHAR}; /* semnat */
int blocklen [3] = { 1, 6, 7};   /* lungimi blocuri */
MPI_Aint disp[3];              /* deplasari */
int base;
/* calculul deplasarilor folosind adrese absolute*/
MPI_Address (particle, disp);
MPI_Address (particle[0].d, disp+1);
MPI_Address (particle[0].b, disp+2);
base=disp[0];
for (i=0; i<3; i++) disp [i] -= base;
MPI_Type_struct ( 3, blocklen, disp, type, &ParticleType);
/* transmiterea intregului tablou */
MPI_Type_commit (&ParticleType);
MPI_Send (particle, MAX_PART, ParticleType, dest, tag, comm);
```

Eficienţa transmisiei

Transmitere a două coloane matrice "a" de m linii şi n coloane

- Operaţii separate pe fiecare coloană

```
MPI_Type_vector (m, 1, n, MPI_FLOAT, &newtype);
for (i=2; i<4; i++)
   MPI_Send (&a[0][i], 1, newtype, ...);
```

- O singură operaţie

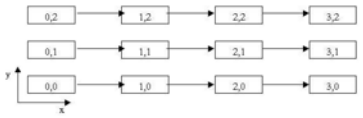
```
MPI_Type_vector (m, 2, n, MPI_FLOAT, &newtype2);
MPI_Send (&a[0][2], 1, newtype2,...);
```

Topologii de procese

Topologii virtuale

Grila sau graf
Usurinta de programare
Nu conteaza la maparea pe topologii reale
Procesele nu trebuie sa deschida explicit canale

Structura carteziana



int MPI_Cart_create (MPI_Comm comm_old, int ndims, int *dims, int *periods, int reorder, MPI_Comm *comm_cart)

Primitive pentru topologii carteziene

• Calcul dims
MPI_Dims_create (nrnodes, ndims, dims);

• Furnizare dims, periods si coords proces apelant
int MPI_Cart_get (MPI_Comm comm, int maxdims,
int *dims, int *periods, int *coords)
• Intoarce coords procesului rank din grupul comm
int MPI_Cart_coords (MPI_Comm comm, int rank, int maxdims,
int *coords)
• Intoarce rank-urile procs sursa si dest pentru send-recv
int MPI_Cart_shift (MPI_Comm comm, int direction, int displ,
int *source, int *dest)

Exemplu

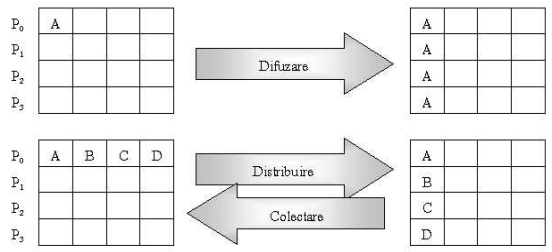
```
int coords[2];
MPI_Comm_rank(comm, &rank);
MPI_Cart_coords (comm, rank, 2, coords);
MPI_Cart_shift (comm, 0, coords[1], &source, &dest);
MPI_Sendrecv_replace(&A,1,MPI_FLOAT,dest,0,source,0,comm,&status);
```

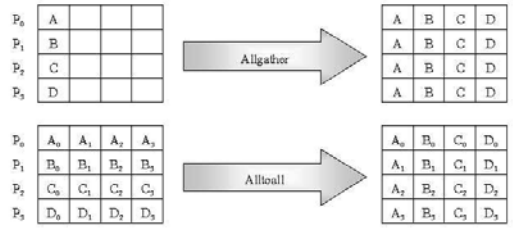
Comunicare colectivă

Sincronizare de tip **barieră** a tuturor proceselor grupului

Comunicări colective, în care se include:
- Difuzarea
- Colectarea datelor de la membrii grupului la rădăcină
- Distribuirea datelor de la rădăcină spre membrii grupului
- Combinații de colectare / distribuire (allgather şi alltoall)

Calcule colective:
- Operații de reducere
- Combinații reducere / distribuire



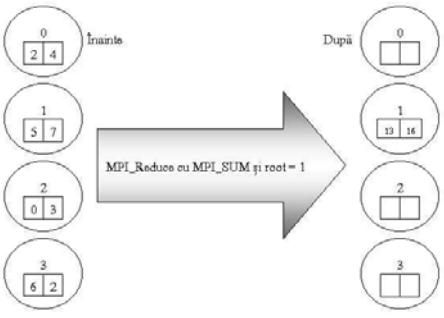


Exemplu: calculul lui π prin integrare numerică

$$\int_0^1 \frac{1}{1+x^2}\,dx = \arctan(x)\,|_0^1 = \frac{\pi}{4}$$

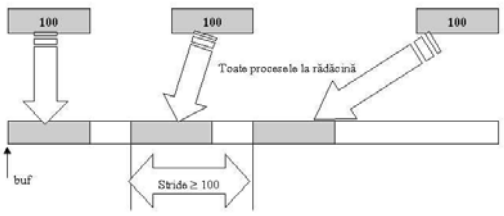
Foloseşte 2 operaţii colective

MPI_Bcast(buffer, count, datatype, root, comm)
MPI_Reduce(sendbuf, recvbuf, count, datatype, op, root, comm)




```
# include "mpi.h"
# include <math.h>
int main (int argc, char **argv)
{ int n, myid, numprocs, i, rc;
  double mypi, pi, h, sum, x, a;
  MPI_Status status;
  MPI_Init (&argc, &argv);
  MPI_Comm_size (MPI_COMM_WORLD, &numprocs);
  MPI_Comm_rank (MPI_COMM_WORLD, &myid);
  while (1)
   { if (myid==0)
       { printf("Numar subintervale (0=gata): ");
         scanf("%d",&n);
       }
     MPI_Bcast (&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
     if(!n) break;
     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) printf("pi este %f\n", pi);
   }
  MPI_Finalize();
}
```

Comunicări colective



MPI_Gatherv (sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs, recvtype, root, comm);