A quick introduction to MPI (Message Passing Interface)

M1IF - APPD

Oguz Kaya Pierre Pradic

École Normale Supérieure de Lyon, France

ENS DE LYON

Introduction

What is MPI ?

- Standardized and portable message-passing system.
- Started in the 90's, still used today in research and industry.
- Good theoretical model.
- Good performances on HPC networks (InfiniBand ...).

De facto standard for communications in HPC applications.

APIs:

- C and Fortran APIs.
- C++ API deprecated by MPI-3 (2008).

Environment:

- Many implementations of the standard (mainly OpenMPI and MPICH)
- Compiler (wrappers around gcc)
- Runtime (mpirun)

Compiling:

gcc	→	mpicc
g++	→	mpic++ / mpicxx
gfortran	→	mpifort

Executing:

mpirun -n <nb procs> <executable> <args>
ex : mpirun -n 10 ./a.out

note: mpiexec and orterun are synonyms of mpirun
see `man mpirun` for more details

Context limits

All MPI call must be nested in the MPI context delimited by `MPI_Init` and `MPI_Finalize`.

```
#include <mpi.h>

int main(int argc, char *argv[])
{
MPI_Init(&argc, &argv);

// ...

MPI_Finalize();

return 0;
}
```

Hello World

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

int main(int argc, char *argv[])
{
int rank, size;

MPI_Init(&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

printf("Hello␣from␣proc␣%d␣/␣%d\n", rank, size);

MPI_Finalize();

return 0;
}
```

Code:

```
printf("[%d]␣step␣1\n", rank);
MPI_Barrier(MPI_COMM_WORLD);
printf("[%d]␣step␣2\n", rank);
```

Output:

```
[0] step 1
[1] step 1
[2] step 1
[3] step 1
[3] step 2
[0] step 2
[2] step 2
[1] step 2
```

Point-to-point communication

Send and Receive

Sending data:

```
int MPI_Send(const void*   data,
             int           count,
             MPI_Datatype  datatype,
             int           destination,
             int           tag,
             MPI_Comm      communicator);
```

Receiving data:

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

Example

```
int rank, size;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

int number;
switch(rank)
{
  case 0:
    number = -1;
    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
    break;
  case 1:
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
    printf("received␣number:␣%d\n", number);
    break;
}
```

Asynchronious communications

Sending data:

```
int MPI_Isend(const void*   data,
              int           count,
              MPI_Datatype  datatype,
              int           destination,
              int           tag,
              MPI_Comm      communicator,
              MPI_Request*  request);
```

Receiving data:

```
int MPI_Irecv(void*         data,
              int           count,
              MPI_Datatype  datatype,
              int           source,
              int           tag,
              MPI_Comm      communicator,
              MPI_Request*  request);
```

Other functions

- MPI_Probe, MPI_Iprobe
- MPI_Test, MPI_Testany, MPI_Testall
- MPI_Cancel
- MPI_Wtime, MPI_Wtick

Simple datatypes

MPI_SHORT	short int
MPI_INT	int
MPI_LONG	long int
MPI_LONG_LONG	long long int
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_BYTE	char

Composed datatypes

Composed structure:

- Structures;
- Array.

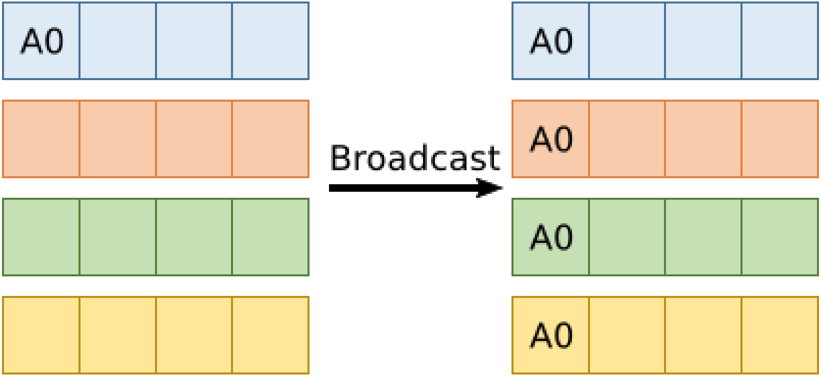
Possibilities are almost limitless ...

... but sometimes difficult to setup.

Collective communications

ENS DE LYON

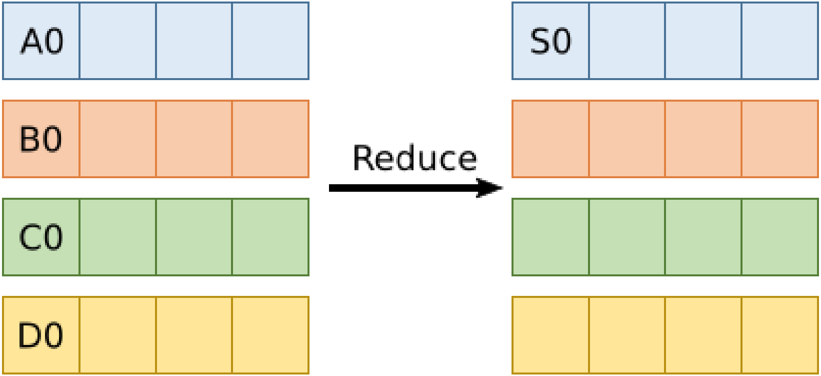
Broadcast



Broadcast

```
int MPI_Bcast(void*          data,
              int            count,
              MPI_Datatype   datatype,
              int            root,
              MPI_Comm       communicator);
```

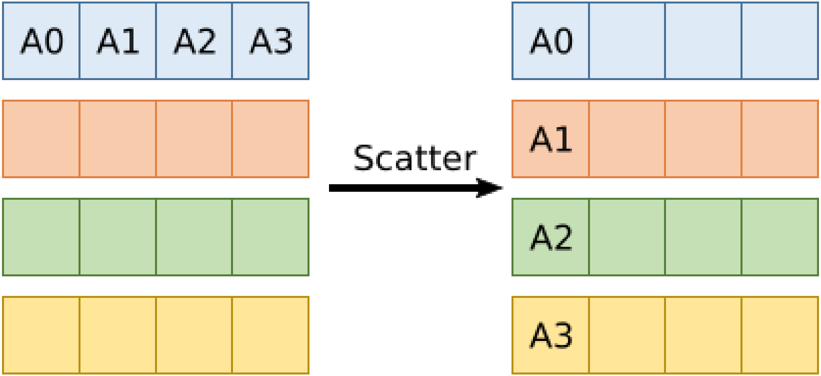
Reduce



Reduce

```
int MPI_Reduce(const void*  sendbuf,
               void*        recvbuf,
               int          count,
               MPI_Datatype datatype,
               MPI_Op       operator,
               int          root,
               MPI_Comm     communicator);
```

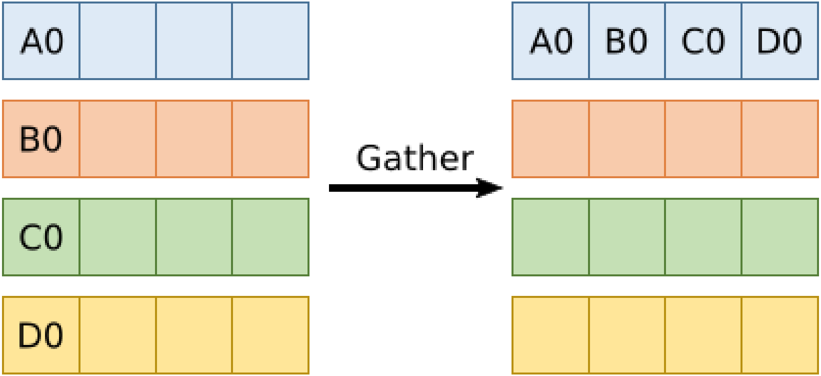
Scatter



Scatter

```
int MPI_Scatter(const void*  sendbuf,
                int          sendcount,
                MPI_Datatype sendtype,
                void*        recvbuf,
                int          recvcount,
                MPI_Datatype recvtype,
                int          root,
                MPI_Comm     communicator);
```

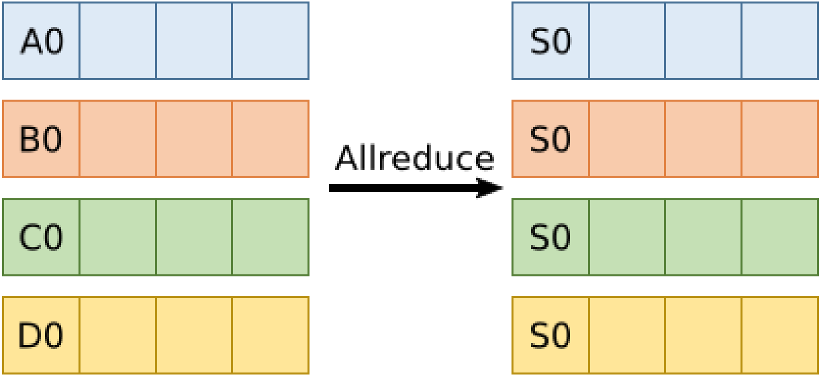
Gather



Gather

```
int MPI_Gather(const void*   sendbuf,
               int           sendcount,
               MPI_Datatype  sendtype,
               void*         recvbuf,
               int           recvcount,
               MPI_Datatype  recvtype,
               int           root,
               MPI_Comm      communicator);
```

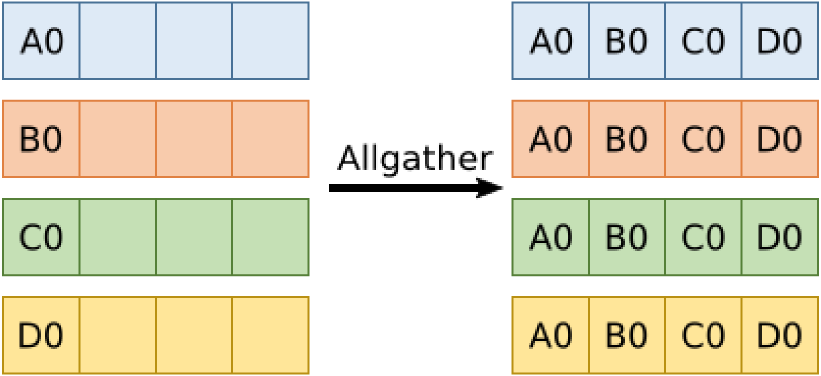
Allreduce



AllReduce

```
int MPI_Allreduce(const void*  sendbuf,
                  void*        recvbuf,
                  int          count,
                  MPI_Datatype datatype,
                  MPI_Op       operator,
                  MPI_Comm     communicator);
```

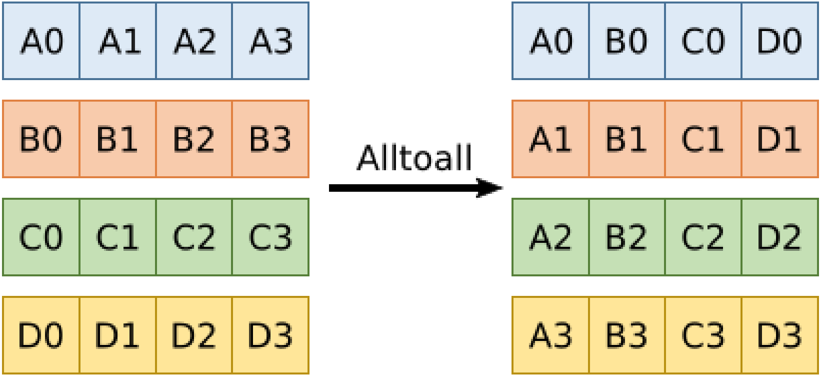
Allgather



AllGather

```
int MPI_Allgather(const void*   sendbuf,
                  int           sendcount,
                  MPI_Datatype  sendtype,
                  void*         recvbuf,
                  int           recvcount,
                  MPI_Datatype  recvtype,
                  MPI_Comm      communicator);
```

Alltoall



A0	A1	A2	A3
B0	B1	B2	B3
C0	C1	C2	C3
D0	D1	D2	D3

Alltoall

A0	B0	C0	D0
A1	B1	C1	D1
A2	B2	C2	D2
A3	B3	C3	D3

Alltoall

```
int MPI_Alltoall(const void*   sendbuf,
                 int           sendcount,
                 MPI_Datatype  sendtype,
                 void*         recvbuf,
                 int           recvcount,
                 MPI_Datatype  recvtype,
                 MPI_Comm      communicator);
```

Custom communicators

MPI_COMM_WORLD can be split into smaller, more appropriate communicators.

```
int MPI_Comm_split(MPI_Comm  communicator,
                   int       color,
                   int       key,
                   MPI_Comm* newcommunicator);
```

Example

```
...
int rank, size;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

int      hrank, vrank;
int      hsize, vsize;
MPI_Comm hcomm, vcomm;
MPI_Comm_split(MPI_COMM_WORLD, rank%p, rank, &vcomm);
MPI_Comm_split(MPI_COMM_WORLD, rank/p, rank, &hcomm);
MPI_Comm_rank(hcomm, &hrank);
MPI_Comm_size(hcomm, &hsize);
MPI_Comm_rank(vcomm, &vrank);
MPI_Comm_size(vcomm, &vsize);
...
```

ENS DE LYON