Introduction à MPI

Anne Cadiou

Laboratoire de Mécanique des Fluides et d'Acoustique

Ateliers et séminaires pour l'informatique et le calcul scientifique
PMCS2I - LMFA
Vendredi 5 avril 2019

Introduction

Définition

- **M**essage **P**assing **I**nterface
- Bibliothèque et standard de communications entre noeuds de calcul

Utilisable sur les machines à mémoire partagées et distribuées
Rapide et portable

MPI

- Gère l'échange des messages entre les processus (tranfert de données, synchronisation, opérations globales)
- Repose sur le principe SPMD (Single Program Multiple Data)
- Chaque processus dispose de ses propres données
- Les échanges entre 1 ou plusieurs processus se font dans un communicateur
- Chaque processus est identifié par son rang au sein du groupe

Historique

- Concept de standard discuté en 1991
- Premier standard présenté à Supercomputing 93'
- MPI-1 release, 1994
  - bibliothèque de fonctions utilisables en C, C++, Fortran
- MPI-2, 1998
  - MPI I/O,
  - gestion dynamique de processus,
  - communications unilatérales (one-sided),
  - C++ interface obsolète (external "C")
- MPI-3, 2012
  - opérations collectives non-bloquantes,
  - fonctions explicites pour les architectures à mémoire partagée,
  - interfaces pour Fortran (2003, 2008)
  - abandon du C++
  - interfaçage d'outils externes (débogage, profiling)
- MPI-4 ?
  - exploitation des accélérateurs ?
  - tolérance aux fautes ?
  - Discussion EuroMPI, sept. 2019
  - Enquête sur les usages (fév. 2019), `https://bosilca.github.io/MPIsurvey/`

Implémentations

Sous licence libre

- MPICH (MPI 1.x), MPICH 2 (MPI-2)
- OpenMPI
- Boost (C++)

Il existe aussi différentes implémentations propriétaires (constructeurs)
Implémentations classiques pour C, C++, Fortran. Existent aussi des implémentations en Python, Julia, Java, OCaml, Perl, etc.

Alternatives ?

Parallélismes différents

- OpenMP, TBB, Pthreads (mémoire partagée)
- OpenACC, CUDA, OpenCL (sur accélérateurs)
- StarPU (parallélisme de tâches)
- kokkos, Alpaka (cross-platforms)

Structure générale d'un programme MPI

```
// debut du programme

// declaration de l'appel a la bibliotheque

// initialisation de l'environnement MPI

// appels a la bibliotheque pour echanger des messages

// fermeture l'environnement MPI

// fin du programme
```

Déclarer l'appel à MPI

Fortran

```
include 'mpif.h'
```

```
use mpi
```

```
f95 prog.f -lmpi
mpif90 prog.f

mpirun -np 4 ./a.out
```

Fortran 2008 (MPI-3)

```
use mpi_f08
```

C

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

```
gcc prog.c -lmpi
mpicc prog.c

mpirun -np 4 ./a.out
```

C++

```
#include <mpi.h>

extern "C" {
  #include <mpi.h>
}
```

```
gcc prog.cpp -lmpi
mpicxx prog.cpp

mpirun -np 4 ./a.out
```

Python

```
import mpi4py.MPI as MPI
```

```
mpirun -np 4 python \
script.py
```

Environnement MPI

Fortran

Initialisation

```
CALL MPI_INIT(IERR)
```

Clôture

```
CALL MPI_FINALIZE(IERR)
```

C

Initialisation

```
int MPI_Init(int *argc,char
      **argv)
```

Clôture

```
int MPI_Finalize(void)
```

Python

Initialisation automatique

```
print(MPI.Is_initialized())

mpi4py.rc.initialize = False
MPI.Init()
```

Clôture

```
print(MPI.Is_finalized())

mpi4py.rc.finalize = False
MPI.Finalize()
```

MPI crée un communicateur qui regroupe tous les processus.
Par défaut, il s'appelle `MPI_COMM_WORLD` (resp. `MPI.COMM_WORLD` pour Python)

Communicateur(s)

Type `MPI` : `MPI_Comm`

Exemple :

Fortran

```
use mpi_f08
Type(MPI_Comm) :: comm =
     MPI_COMM_WORLD
```

C

```
#include <mpi.h>
MPI_Comm comm =
     MPI_COMM_WORLD;
```

Python

```
comm = MPI.COMM_WORLD
```

- Défini un ensemble de processus actifs
  - Peut être créé ou détruit en cours d'application ($\geq$ MPI-2)
- Un processus a un ou plusieurs identifiants (communicateur, numéro)
  - Un processus peut appartenir à plusieurs communicateurs
  - Un processus peut avoir un numéro différent dans chaque communicateur
- Les communications MPI doivent spécifier le communicateur dans lequel elles se font

Un communicateur peut être décomposé, associé à une topologie particulière, etc.

Syntaxe des appels aux variables et fonctions de MPI

Utiliser le préfixe MPI_
En C++ (obsolète), appel de la classe MPI::
mpi4py avec Python s'en inspire. Classe MPI.

Fortran

```
CALL MPI_XXX(parameter,...,
     ierr)
call mpi_xxx(parameter,...,
     ierr)

CALL MPI_BSEND(buf,count,
     type,dest,tag,comm,
     ierr)
```

ierr retourne le code d'erreur.
MPI_SUCCESS si ok.

C
Attention : en C, respecter les majuscules et minuscules.

```
rc = MPI_Xxx(parameter,...)

rc = MPI_Bsend(&buf,count,
     type,dest,tag,comm)
```

rc retourne le code d'erreur.
MPI_SUCCESS si ok.

Python
basé sur l'interface C++ de MPI-2

```
comm = MPI.COMM_WORLD

  # communication of Python
        objects (lent)
comm.xxx(data,parameters
     ,...)
comm.bsend(buffer,dest=0,
     tag=1)

# communication of buffer-
     like objects
comm.Xxx([data,count,MPI.
     type],parameters,...)
comm.Bsend([buffer,buffsize
     ,MPI.INT],dest=0,tag
     =1)
```

Exemple d'identification des processus

Initialiser l'environnement MPI et retourner le numéro de chaque processus actif dans le communicateur crée.

Exécution :

```
mpirun -np 4 ./a.out
```

```
mpirun -np 4 python script.py
```

Résultat :

```
Hello! I am process 0 of 4 on plume.
Hello! I am process 1 of 4 on plume.
Hello! I am process 2 of 4 on plume.
Hello! I am process 3 of 4 on plume.
```

Exemple d'identification des processus en Fortran

```
PROGRAM hello

  USE mpi
  IMPLICIT NONE

  INTEGER :: numtasks, rank, reslen, ierr = 0
  CHARACTER(MPI_MAX_PROCESSOR_NAME) :: hostname

  CALL MPI_INIT(ierr)

  CALL MPI_COMM_SIZE(MPI_COMM_WORLD, numtasks, ierr)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)

  CALL MPI_GET_PROCESSOR_NAME(hostname, reslen, ierr)
  WRITE(*, '(2A,I2,A,I2,3A)') &
       'Hello! ', &
       'I am process ', rank, &
       ' of ', numtasks, &
       ' on ', hostname(1:reslen), '.'

  CALL MPI_FINALIZE(ierr)

END PROGRAM hello
```

Exemple d'identification des processus en C

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

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

  int numtasks, rank, reslen, rc;
  char hostname[MPI_MAX_PROCESSOR_NAME];

  MPI_Init(&argc,&argv);

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

  MPI_Get_processor_name(hostname, &reslen);
  printf("Hello! I am process %d of %d on %s.\n",rank,numtasks,hostname);

  MPI_Finalize();
}
```

Exemple d'identification des processus en Python

```
#!/usr/bin/env python
"""
for python3
"""

import mpi4py.MPI as MPI

rank = MPI.COMM_WORLD.Get_rank()
numtasks = MPI.COMM_WORLD.Get_size()
hostname = MPI.Get_processor_name()

mess = "Hello! I am process %d of %d on %s."
print(mess % (rank, numtasks, hostname))
```

Communications

Les communications échangent des messages entre au moins 2 processus

- 1 qui envoie
- 1 qui reçoit

Différents types de communications

- Communications point à point
- Communications globales (ou collectives)
- Communications one-sided (découpler le mouvement des données de la synchronisation des processus) ($\geq$ MPI-2)

Les communications peuvent être bloquantes ou non-bloquantes.
Les messages échangés sont des données typées.

Principaux types de données en Fortran

type de données MPI	type Fortran
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER
MPI_BYTE	
MPI_PACKED	

Principaux types de données en C

type de données MPI	type C
MPI_CHAR	char
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	
MPI_PACKED	

Principaux types de données en Python

type de données MPI	type NumPy
MPI.INTEGER	np.intc
MPI.LONG	np.int
MPI.FLOAT	np.float32
MPI.DOUBLE	np.float64

Communications point à point

C'est une communication entre deux processus : un expéditeur et un destinataire
Elle comprend 2 étapes : l'envoi et la réception.

Un message est constitué de son enveloppe, qui comprend

- Le communicateur
- l'identifiant du processus expéditeur
- l'identifiant du processus destinataire
- une étiquette (`tag`) qui permet au programme de distinguer différents messages

Le contenu du message comprend

- Les données échangées
- Le type de la donnée
- Sa taille

Envoi/réception d'un message

Fortran

```
CALL MPI_SEND(message,taille,type,destination,identificateur,MPI_COMM_WORLD,ierr)
```

```
CALL MPI_RECV(message,taille,type,source,identificateur,MPI_COMM_WORLD,status,ierr)
```

C

```
MPI_Send(void* data, int count, datatype, int dest, int tag, communicator)
```

```
MPI_Recv(void* data, int count, datatype, int source, int tag, communicator, status)
```

Python

```
MPI.COMM_WORLD.send(data, dest=#, tag=#)
```

```
MPI.COMM_WORLD.Send(data, dest=#, tag=#)
```

```
data = MPI.COMM_WORLD.recv(source=#, tag=#)
```

```
MPI.COMM_WORLD.Recv(data, source=#, tag=#)
```

(send/recv : objets génériques Python, Send/Recv : tableaux NumPy, plus rapide)

Exemple : lecture d'une donnée par un processus

Le processus 0 (existe toujours) lit un entier et l'envoie aux autres processus, qui lui retournent sa valeur, multipliée par leur rang dans le communicateur. (configuration de type maître/esclave)

```
$ mpirun -np 4 ./a.out

Enter integer number:
12
Slave 1 has received n=12 from 0
Slave 2 has received n=12 from 0
Slave 3 has received n=12 from 0
Master 0 received from slave 1: 12
Master 0 received from slave 2: 24
Master 0 received from slave 3: 36
```

Exemple en Fortran

```
PROGRAM point_a_point

  USE mpi
  IMPLICIT NONE

  INTEGER, DIMENSION(MPI_STATUS_SIZE) :: statut
  INTEGER, PARAMETER :: tagm=101, tagr=201, master=0
  INTEGER :: rang,nprocs,i,n,ierr

  CALL MPI_INIT(ierr)

  CALL MPI_COMM_RANK(MPI_COMM_WORLD,rang,ierr)
  CALL MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)

  IF (rang == master) THEN
    WRITE(*,*) 'Enter integer number:'
    READ(*,*) n
    DO i=1,nprocs-1
      CALL MPI_SEND(n,1,MPI_INTEGER,i,tagm,MPI_COMM_WORLD,ierr)
    END DO
    DO i=1,nprocs-1
      CALL MPI_RECV(n,1,MPI_INTEGER,i,tagr,MPI_COMM_WORLD,statut,ierr)
      WRITE(*,'(A,I2,A,I2,A,I3)') "Master",rang," received from slave ",i,": n=",n
      WRITE(*,*) n,statut(MPI_SOURCE),statut(MPI_TAG),statut(MPI_ERROR)
    END DO
  ELSE
```

```
    CALL MPI_RECV(n,1,MPI_INTEGER,master,tagm,MPI_COMM_WORLD,statut,ierr)
    WRITE(*,'(A,I2,A,I3,A,I2)') "Slave ",rang," has received n=",n," from ",master
    WRITE(*,*) n,statut(MPI_SOURCE),statut(MPI_TAG),statut(MPI_ERROR)
    n = n*rang
    CALL MPI_SEND(n,1,MPI_INTEGER,master,tagr,MPI_COMM_WORLD,ierr)
  END IF

  CALL MPI_FINALIZE(ierr)

END PROGRAM point_a_point
```

Exemple en C

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

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

  int rang,nprocs;
  int master,tagm,tagr;
  int n,i;
  MPI_Status status;

  MPI_Init(&argc,&argv);

  master = 0;
  tagm = 101;
  tagr = 201;

  MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&rang);

  if (rang == master) {
```

```
    printf("Enter integer number:\n");
    scanf("%d",&n);
    for (i=1; i<nprocs; i++) {
      MPI_Send(&n,1,MPI_INT,i,tagm,MPI_COMM_WORLD);
    }
    for (i=1; i<nprocs; i++) {
      MPI_Recv(&n,1,MPI_INT,i,tagr,MPI_COMM_WORLD,&status);
      printf("Master %d received from slave %d: %d\n",rang, i, n);
    }
  } else {
    MPI_Recv(&n,1,MPI_INT,master,tagm,MPI_COMM_WORLD,&status);
    printf("Slave %d has received n=%d from %d\n",rang,n,master);
    n = n*rang;
    MPI_Send(&n,1,MPI_INT,master,tagr,MPI_COMM_WORLD);
  }

  MPI_Finalize();
}
```

Exemple en Python

```
#!/usr/bin/env python
""" for python3 """
import mpi4py.MPI as MPI

rang = MPI.COMM_WORLD.Get_rank()
nprocs = MPI.COMM_WORLD.Get_size()

master = 0
tagm = 101
tagr = 201

if rang == master:
    print('Enter integer number:')
    n = int(input())
    for i in range(1,nprocs):
        MPI.COMM_WORLD.send(n,dest=i,tag=tagm)
    for i in range(1,nprocs):
        n = MPI.COMM_WORLD.recv(source=i,tag=tagr)
        print("Master ",rang," received from slave ",i,": n=",n)
else:
    n = MPI.COMM_WORLD.recv(source=0,tag=tagm)
    print("Slave ",rang," has received n=",n," from ",master)
    n = n*rang
    MPI.COMM_WORLD.send(n,dest=0,tag=tagr)
```

Exemple en Python avec les structures NumPy

```
#!/usr/bin/env python
"""
for python3
"""

import mpi4py.MPI as MPI
import numpy as np

rang = MPI.COMM_WORLD.Get_rank()
nprocs = MPI.COMM_WORLD.Get_size()

master = 0
tagm = 101
tagr = 201

if rang == master:
    print('Enter integer number:')
    n = input()
    data = np.array([n],dtype='i')
    for i in range(1,nprocs):
        MPI.COMM_WORLD.Send([data,MPI.INT],dest=i,tag=tagm)
    for i in range(1,nprocs):
        MPI.COMM_WORLD.Recv([data,MPI.INT],source=i,tag=tagr)
        print("Master ",rang," received from slave ",i,": n=",data[0])
else:
```

```
    data = np.empty(1,dtype='i')
    MPI.COMM_WORLD.Recv([data,MPI.INT],source=0,tag=tagm)
    print("Slave ",rang," has received n=",data[0]," from ",master)
    data[0] = data[0]*rang
    MPI.COMM_WORLD.Send([data,MPI.INT],dest=0,tag=tagr)
```

```
(mpi4py3) $ mpirun -np 4 python point_a_point_numpy.py

Enter integer number:
12
Slave 1 has received n= 12 from 0
Slave 3 has received n= 12 from 0
Master 0 received from slave 1 : n= 12
Slave 2 has received n= 12 from 0
Master 0 received from slave 2 : n= 24
Master 0 received from slave 3 : n= 36
```

Communications bloquantes

`MPI_SEND` et `MPI_RECV` sont des communications bloquantes

- L'envoi et la réception d'une donnée attend la fin de la communication avant de continuer l'exécution
- Attention à bien respecter les correspondances de l'enveloppe et des données (source, tag, etc.)

Avantage

- La donnée est entièrement reçue ou envoyée avant qu'elle puisse être modifiée par le programme

Inconvénient

- Il faut attendre la fin de l'opération pour continuer les calculs

MPI_SENDRECV

MPI_SEND et MPI_RECV peuvent être combinés dans un seul échange avec MPI_SENDRECV, cependant

- Toutes les communications se font simultanément
- Les deux processus qui échangent peuvent être le même
- Utile dans une chaîne de processus
- Tous les processus commencent par envoyer son message (aucun n'est en réception)

⇒ Risque de blocage (deadlock) car MPI_SEND et MPI_RECV sont des envois/réception bloquants, suivant l'implémentation MPI

Exemple

Chaque processus envoi un entier (100) dont la valeur est augmentée de son rang, avec son voisin (communication en anneau)

Exécution :

```
mpirun -np 4 ./a.out
```

```
mpirun -np 4 python script.py
```

Résultat :

```
 Process 2 has sent 102 and received 101
 Process 0 has sent 100 and received 103
 Process 3 has sent 103 and received 102
 Process 1 has sent 101 and received 100
```

Exemple en Fortran

```
PROGRAM sendrecv
  USE mpi
  IMPLICIT NONE
  INTEGER :: x,y,nprocs,rang,suiv,prec
  INTEGER :: ierr = 0
  INTEGER, PARAMETER :: tag=110

  CALL MPI_INIT(ierr)
  CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD, rang, ierr)

  suiv = mod(rang+1, nprocs)
  prec = mod(nprocs+rang-1, nprocs)

  y = 0
  x = 100 + rang

  CALL MPI_SENDRECV(x, 1, MPI_INTEGER, suiv,tag, &
                    y, 1, MPI_INTEGER, prec,tag, &
                    MPI_COMM_WORLD, MPI_STATUS_IGNORE, ierr)

  print *,'Process',rang,' has sent',x,' and received ',y

  CALL MPI_FINALIZE(ierr)

END PROGRAM sendrecv
```

Exemple en C

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

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

  int x,y,nprocs,rang,suiv,prec;
  int tag=110;

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &rang);

  suiv = (rang+1)%nprocs;
  prec = (nprocs+rang-1)%nprocs;

  y = 0;
  x = 100 + rang;

  MPI_Sendrecv(&x, 1, MPI_INTEGER, suiv, tag,
               &y, 1, MPI_INTEGER, prec, tag,
               MPI_COMM_WORLD, MPI_STATUS_IGNORE);

  printf("Process %d has sent %d and received %d \n",rang,x,y);

  MPI_Finalize();
  return 0;
}
```

Exemple en Python

```
#!/usr/bin/env python
"""
for python3
"""

import mpi4py.MPI as MPI

comm = MPI.COMM_WORLD
rang = comm.Get_rank()
nprocs = comm.Get_size()

suiv = (rang+1)%nprocs
prec = (nprocs+rang-1)%nprocs

y = 0
x = 100 + rang

y = comm.sendrecv(x,dest=suiv,source=prec)

print("Process ",rang,"has sent ",x,"and received ",y)
```

Exemple en Python avec NumPy

```
#!/usr/bin/env python
"""
for python3
"""

import mpi4py.MPI as MPI
import numpy as np

comm = MPI.COMM_WORLD
rang = comm.Get_rank()
nprocs = comm.Get_size()

suiv = (rang+1)%nprocs
prec = (nprocs+rang-1)%nprocs

y = 0
x = 100 + rang

recv_data = np.empty(1,dtype='i')
data = np.array([x],dtype='i')
comm.Sendrecv([data,MPI.INT],dest=suiv,recvbuf=recv_data,source=prec)

print("Process ",rang,"has sent ",data[0],"and received ",recv_data[0])
```

Optimisation des communications point à point

Les communications bloquantes sont liées à

- de la re-copie dans un espace temporaire (buffer)
- de la synchronisation (avant de continuer le calcul après un envoi, le code attend que la réception correspondante doit initialisée)

Optimiser consiste à **minimiser le temps passé à faire autre chose que des calculs** (l'overhead)

Possibilités :

- Recouvrir les communications par du calcul
- Éviter la recopie de message dans un espace temporaires (buffer)
- Minimiser les surcoûts liés aux appels répétés aux routines de communications

Les comunications peuvent être standards, synchrones, bufferisées ou persistantes.

Communications asynchrones

Le programme continue avant l'initialisation de la réception correspondante : cela permet de calculer pendant le transfert

Fortran

```
CALL MPI_ISEND(message,taille,type,destination,tag,comm,request,ierr)
```

```
CALL MPI_IRECV(message,taille,type,source,tag,comm,request,ierr)
```

C

```
MPI_Isend(void* data, int count, datatype, int dest, int tag, comm, request)
```

```
MPI_Irecv(void* data, int count, datatype, int source, int tag, comm, status)
```

Python

```
MPI.COMM_WORLD.isend(data, dest=#, tag=#)
```

```
MPI.COMM_WORLD.Isend(data, dest=#, tag=#)
```

```
data = MPI.COMM_WORLD.irecv(source=#, tag=#)
```

```
MPI.COMM_WORLD.Irecv(data, source=#, tag=#)
```

Commandes de contrôle associées

Fortran

Attendre jusqu'à la terminaison d'une requête

```
CALL MPI_WAIT(request,status,ierr)
```

Vérifier si une requête est terminée

```
CALL MPI_TEST(request,flag,status,ierr)
```

Contrôler avant réception si une requête est arrivée

```
CALL MPI_PROBE(source,tag,status,comm,ierr)
```

Il existe des variantes asynchrones de ces commandes.

Voir la syntaxe des appels par exemple sur :
Aide mémoire MPI en Fortran, IDRIS

C

Attendre jusqu'à la terminaison d'une requête

```
MPI_Wait(request,status)
```

Vérifier si une requête est terminée

```
MPI_Test(request,flag,status)
```

Contrôler avant réception si une requête est arrivée

```
MPI_Probe(source,tag,comm,flag,status)
```

Voir la syntaxe des appels par exemple sur :
Aide mémoire MPI-2

Python

Attendre jusqu'à la terminaison d'une requête

```
req.wait()
```

```
MPI.Request.Wait(req)
```

Vérifier si une requête est terminée

```
MPI_Test(request,flag,status)
```

Contrôler avant réception si une requête est arrivée

```
MPI_Probe(source,tag,comm,flag,status)
```

Voir la syntaxe des appels par exemple sur :
Documentation mpi4py

Exemple en Python avec NumPy

```
#!/usr/bin/env python
"""
for python3
"""

import mpi4py.MPI as MPI
import numpy as np

comm = MPI.COMM_WORLD
rang = comm.Get_rank()

if (comm.Get_size() != 2):
    if (rang == 0):
        print('Stop.\n Required: 2 MPI processes')
    exit()

n = 10
x = np.arange(n,dtype=np.float64)

data = (1+rang)*x
if rang == 0:
    req = comm.Isend([data,n,MPI.DOUBLE],dest=1,tag=101)
elif rang == 1:
```

```
    print('on task',rang,'before recv: data = ',data)
    req = comm.Irecv(data,source=0,tag=101)
    re = False
    while re == False :
       re = MPI.Request.Test(req)
    print('test result',re)
    re = MPI.Request.Wait(req)
    print('wait result',re)
    print('on task',rang,'after recv: data = ',data)
```

Résultat :

```
on task 1 before recv: data = [ 0. 2. 4. 6. 8. 10. 12. 14. 16. 18.]
test result True
wait result True
on task 1 after recv: data = [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
```

Principales communications usuelles

Fortran	C	Python (Numpy)	type
Envoi	Envoi	Envoi	Envoi
• MPI_SEND	• MPI_Send	• MPI.Send	bloquant, standard
• MPI_ISEND	• MPI_Isend	• MPI.Isend	non bloquant, standard
• MPI_SSEND	• MPI_Ssend	• MPI.Ssend	bloquant, synchrone
• MPI_ISSEND	• MPI_Issend	• MPI.Issend	non bloquant, synchrone
• MPI_BSEND	• MPI_Bsend	• MPI.Bsend	bloquant, bufferisé
• MPI_IBSEND	• MPI_Ibsend	• MPI.Ibsend	non bloquant, bufferisé
Réception	Réception	Réception	Réception
• MPI_RECV	• MPI_Recv	• MPI.Recv	bloquant, standard
• MPI_IRECV	• MPI_Irecv	• MPI.Irecv	non bloquant, standard
Contrôle	Contrôle	Contrôle	Contrôle
• MPI_WAIT	• MPI_Wait	• MPI.Wait	attente op. effectuée

Quelques règles simples

- Initier les réceptions avant les envois

⇒ Mettre les appels à `MPI_IRECV` avant `MPI_SEND`

- Éviter l'usage des buffers

⇒ Utiliser les routines synchrones `MPI_SSEND`

- Recouvrir les communications par des calculs

⇒ Utiliser des communications non bloquantes `MPI_ISEND` et `MPI_IRECV`

- Minimiser les surcoûts liés à des appels répétés avec les mêmes arguments

⇒ Utiliser des communications persistantes

Communications collectives

- C'est une communication qui implique tous les processus du même communicateur (succession de communications point à point).
- Les processus effectuent le même appel, avec des arguments correspondants.
- Il n'y a pas d'étiquette.
- Certaines opérations ont un processus qui est seul destinataire ou seul expéditeur. Nommé par défaut processus `root`.

Principales opérations

Envoi et réception

- Synchronisation
- Diffusion d'une même données depuis `root` vers tous les processus du communicateur
- Distribution des données sur chaque processus
- Regroupement des données sur le `root`

Opérations sur les données entre les processus

- Réduction (max, min, somme, produit, etc.)

Synchronisation

Fortran

```
CALL MPI_BARRIER(MPI_COMM_WORLD,ierr)
```

C

```
MPI_Barrier(MPI_COMM_WORLD)
```

Python

```
comm = MPI.COMM_WORLD
```

```
comm.barrier()
```

```
comm.Barrier()
```

Exemple : mesure du temps occupé par chaque processus

MPI permet de mesurer le temps passé par chaque tâche.

Exécution :

```
mpirun -np 4 ./a.out
```

```
mpirun -np 4 python script.py
```

Résultat :

```
Hello! I am process 0 of 4 on plume.
Hello! I am process 1 of 4 on plume.
Hello! I am process 2 of 4 on plume.
Hello! I am process 3 of 4 on plume.
Time spent by process 0 : 0.000226 sec.
Time spent by process 1 : 0.000225 sec.
Time spent by process 3 : 0.000226 sec.
Time spent by process 2 : 0.000265 sec.
```

(Ici le temps est court car il n'y a pas de message échangés.)

Exemple de timing en Fortran sans MPI_BARRIER

```
PROGRAM hello

  USE mpi
  IMPLICIT NONE

  INTEGER :: numtasks, rank, reslen, ierr = 0
  CHARACTER(MPI_MAX_PROCESSOR_NAME) :: hostname
  REAL*8 :: t_start,t_stop

  CALL MPI_INIT(ierr)

  CALL MPI_COMM_SIZE(MPI_COMM_WORLD, numtasks, ierr)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)

  CALL MPI_GET_PROCESSOR_NAME(hostname, reslen, ierr)
```

```
18
  t_start = MPI_WTIME()
20
  WRITE(*, '(2A,I2,A,I2,3A)') &
22      'Hello! ', &
        'I am process ', rank, &
24      ' of ', numtasks, &
        ' on ', hostname(1:reslen), '.'
26

28  t_stop = MPI_WTIME()

30  WRITE(*, '(A,I2,A,E14.7,A)') 'Time spent by process ',rank,' : ',t_stop-t_start,
       ' sec.'

32  CALL MPI_FINALIZE(ierr)

34 END PROGRAM hello
```

Exécution :

```
mpirun -np 4 ./a.out
```

Résultat

```
Hello! I am process 0 of 8 on plume.
Time spent by process 0 : 0.3466202E-04 sec.
Hello! I am process 2 of 8 on plume.
Time spent by process 2 : 0.5674304E-04 sec.
Hello! I am process 7 of 8 on plume.
Hello! I am process 4 of 8 on plume.
Hello! I am process 5 of 8 on plume.
Time spent by process 5 : 0.3789802E-04 sec.
Hello! I am process 6 of 8 on plume.
Time spent by process 6 : 0.2802198E-04 sec.
Time spent by process 4 : 0.3427902E-04 sec.
Hello! I am process 3 of 8 on plume.
Time spent by process 3 : 0.2747600E-04 sec.
Hello! I am process 1 of 8 on plume.
Time spent by process 1 : 0.3003597E-04 sec.
Time spent by process 7 : 0.3466196E-04 sec.
```

Exemple de timing en Fortran avec MPI_BARRIER

```
PROGRAM hello

  USE mpi
  IMPLICIT NONE

  INTEGER :: numtasks, rank, reslen, ierr = 0
  CHARACTER(MPI_MAX_PROCESSOR_NAME) :: hostname
  REAL*8 :: t_start,t_stop

  CALL MPI_INIT(ierr)

  CALL MPI_COMM_SIZE(MPI_COMM_WORLD, numtasks, ierr)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)

  CALL MPI_GET_PROCESSOR_NAME(hostname, reslen, ierr)
```

```
  ! synchronization point
  CALL MPI_BARRIER(MPI_COMM_WORLD, ierr)
  t_start = MPI_WTIME()

  WRITE(*, '(2A,I2,A,I2,3A)') &
       'Hello! ', &
       'I am process ', rank, &
       ' of ', numtasks, &
       ' on ', hostname(1:reslen), '.'

  CALL MPI_BARRIER(MPI_COMM_WORLD, ierr)
  t_stop = MPI_WTIME()

  WRITE(*, '(A,I2,A,E14.7,A)') 'Time spent by process ',rank,' : ',t_stop-t_start,
       ' sec.'

  CALL MPI_FINALIZE(ierr)

END PROGRAM hello
```

Exécution :

```
mpirun -np 4 ./a.out
```

Résultat :

```
Hello! I am process 1 of 8 on plume.
Hello! I am process 3 of 8 on plume.
Hello! I am process 5 of 8 on plume.
Hello! I am process 7 of 8 on plume.
Hello! I am process 4 of 8 on plume.
Hello! I am process 0 of 8 on plume.
Hello! I am process 2 of 8 on plume.
Hello! I am process 6 of 8 on plume.
Time spent by process 4 : 0.1428364E-02 sec.
Time spent by process 7 : 0.1649751E-02 sec.
Time spent by process 2 : 0.1542790E-03 sec.
Time spent by process 6 : 0.1533800E-03 sec.
Time spent by process 0 : 0.1479295E-02 sec.
Time spent by process 1 : 0.1816410E-02 sec.
Time spent by process 3 : 0.1763733E-02 sec.
Time spent by process 5 : 0.1752706E-02 sec.
```

Exemple de timing en C

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

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

  int numtasks, rank, reslen, rc;
  char hostname[MPI_MAX_PROCESSOR_NAME];
  double t_start,t_stop;

  MPI_Init(&argc,&argv);

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

  MPI_Get_processor_name(hostname, &reslen);

  MPI_Barrier(MPI_COMM_WORLD);
  t_start = MPI_Wtime();
  printf("Hello! I am process %d of %d on %s.\n",rank,numtasks,hostname);

  MPI_Barrier(MPI_COMM_WORLD);
  t_stop = MPI_Wtime();

  printf("Time spent by process %d : %f sec.\n",rank,t_stop-t_start);

  MPI_Finalize();
}
```

Exemple de timing en Python

```
#!/usr/bin/env python
"""
for python3
"""

import mpi4py.MPI as MPI

rank = MPI.COMM_WORLD.Get_rank()
numtasks = MPI.COMM_WORLD.Get_size()
hostname = MPI.Get_processor_name()

MPI.COMM_WORLD.Barrier()
t_start = MPI.Wtime()
mess = "Hello! I am process %d of %d on %s."
print(mess % (rank, numtasks, hostname))

MPI.COMM_WORLD.Barrier()
t_stop = MPI.Wtime()
mess = "Time spent by process %d : %f sec."
print(mess % (rank, t_stop-t_start))
```

Diffusion

Envoi d'une donnée depuis un processus vers tous les autres (broadcast).

Fortran

```
CALL MPI_BCAST(message,taille,type,source,MPI_COMM_WORLD,ierr)
```

C

```
MPI_Bcast(message,taille,type,source,MPI_COMM_WORLD)
```

Python

```
comm = MPI.COMM_WORLD
```

```
data = comm.bcast(data,root=source)
```

```
comm.Bcast(data,root=source)
```

Exemple en Fortran

```
PROGRAM bcast
  USE mpi
  IMPLICIT NONE
  INTEGER :: n,nprocs,rang
  INTEGER :: ierr = 0
  INTEGER, PARAMETER :: idat=91

  CALL MPI_INIT(ierr)
  CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD, rang, ierr)

  IF (rang.eq.0) THEN
      OPEN(unit=idat,file='data.txt')
      READ(idat,*) n
      CLOSE(idat)
  END IF

  CALL MPI_BCAST(n,1,MPI_INT,0,MPI_COMM_WORLD,ierr)

  print *,'Process',rang,' has received ',n

  CALL MPI_FINALIZE(ierr)

END PROGRAM bcast
```

Fichier :

```
cat data.txt
```

```
29
```

Exécution :

```
mpirun -np 4 ./a.out
```

Résultat :

```
 Process 0 has received 29
 Process 1 has received 29
 Process 2 has received 29
 Process 3 has received 29
```

Exemple en C

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

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

  int nprocs,rang;
  int n;

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD, &rang);

  if (rang==0) {
    FILE *fp = fopen("data.txt", "r");
    fscanf(fp, "%d", &n);
    fclose(fp);
  }

  MPI_Bcast(&n,1,MPI_INTEGER,0,MPI_COMM_WORLD);

  printf("Process %d has received %d \n",rang,n);

  MPI_Finalize();
  return 0;
}
```

Exemple en Python

```
#!/usr/bin/env python
"""
for python3
"""

import mpi4py.MPI as MPI

comm = MPI.COMM_WORLD
rang = comm.Get_rank()
nprocs = comm.Get_size()

if rang == 0:
    f = open('data.txt','r')
    n = int(f.read())
else:
    n = None

n = comm.bcast(n,root=0)

print("Process ",rang,"has received ",n)
```

Exemple en Python avec NumPy

```
#!/usr/bin/env python
"""
for python3
"""

import mpi4py.MPI as MPI
import numpy as np

comm = MPI.COMM_WORLD
rang = comm.Get_rank()
nprocs = comm.Get_size()

if rang == 0:
    f = open('data.txt','r')
    n = np.array([int(f.read())], dtype=int)
else:
    n = np.zeros(1, dtype=int)

comm.Bcast(n,root=0)

print("Process ",rang,"has received ",n[0])
```

Distribution sélective de données

Fortran

```
CALL MPI_SCATTER(sendbuf, sendcount, sendtype, &
                 recvbuf, recvcount, recvtype, &
                 root, comm, ierr)
```

C

```
MPI_Scatter(sendbuf, sendcount, sendtype,
            recvbuf, recvcount, recvtype,
            root, comm)
```

Python

```
comm = MPI.COMM_WORLD
```

```
data = comm.scatter(sendbuff, root)
```

```
comm.Scatter([senddata, data_size, data_type], [recvdata, data_size, data_type], root)
```

Exemple : distribution d'un tableau sur 4 processus

Exemple inspiré de `https://computing.llnl.gov/tutorials/mpi`.

Objectif : distribuer des données 4×5 sur 4 processus.

$$\begin{pmatrix} 1.0 & 2.0 & 3.0 & 4.0 & 4.5 \\ 5.0 & 6.0 & 7.0 & 8.0 & 8.5 \\ 9.0 & 10.0 & 11.0 & 12.0 & 12.5 \\ 13.0 & 14.0 & 15.0 & 16.0 & 16.5 \end{pmatrix}$$

Exécution :

```
mpirun -np 4 ./a.out
```

```
mpirun -np 4 python script.py
```

Résultat :

```
rank= 0 Results: 1.00000000 2.00000000 3.00000000 4.00000000 4.50000000
rank= 1 Results: 5.00000000 6.00000000 7.00000000 8.00000000 8.50000000
rank= 2 Results: 9.00000000 10.0000000 11.0000000 12.0000000 12.5000000
rank= 3 Results: 13.0000000 14.0000000 15.0000000 16.0000000 16.5000000
```

Exemple en Fortran

```
program scatter
   include 'mpif.h'

   integer SIZE_X,SIZE_Y
   parameter(SIZE_X=5,SIZE_Y=4)
   integer numtasks, rank, sendcount, recvcount, source, ierr

   real*4 sendbuf(SIZE_X,SIZE_Y), recvbuf(SIZE_X)
   data sendbuf /1.0, 2.0, 3.0, 4.0, 4.5, &
                 5.0, 6.0, 7.0, 8.0, 8.5, &
                 9.0, 10.0, 11.0, 12.0, 12.5, &
                 13.0, 14.0, 15.0, 16.0, 16.5/

   call MPI_INIT(ierr)
   call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
   call MPI_COMM_SIZE(MPI_COMM_WORLD, numtasks, ierr)

   if (numtasks .eq. SIZE_Y) then

      source = 1
      sendcount = SIZE_X
      recvcount = SIZE_X
```

```
      call MPI_SCATTER(sendbuf, sendcount, MPI_REAL, &
                       recvbuf, recvcount, MPI_REAL, &
                       source, MPI_COMM_WORLD, ierr)

      print *, 'rank= ',rank,' Results: ',recvbuf

   else
      print *, 'Must specify',SIZE_Y,' processors. Terminating.'
   endif

   call MPI_FINALIZE(ierr)

   end
```

Exemple en C

```
#include "mpi.h"
#include <stdio.h>
#define SIZE_X 5
#define SIZE_Y 4

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

  int numtasks, rank, sendcount, recvcount, source;
  float sendbuf[SIZE_Y][SIZE_X] = {
    {1.0, 2.0, 3.0, 4.0, 4.5},
    {5.0, 6.0, 7.0, 8.0, 8.5},
    {9.0, 10.0, 11.0, 12.0, 12.5},
    {13.0, 14.0, 15.0, 16.0, 16.5} };
  float recvbuf[SIZE_X];

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

  if (numtasks == SIZE_Y) {
    // define source task and elements to send/receive, then perform collective
         scatter
    source = 1;
    sendcount = SIZE_X;
    recvcount = SIZE_X;
```

```
    MPI_Scatter(sendbuf,sendcount,MPI_FLOAT,recvbuf,recvcount,
                MPI_FLOAT,source,MPI_COMM_WORLD);

    printf("rank= %d Results: %f %f %f %f %f\n",rank,recvbuf[0],
           recvbuf[1],recvbuf[2],recvbuf[3],recvbuf[4]);
    }
  else
    printf("Must specify %d processors. Terminating.\n",SIZE_Y);

  MPI_Finalize();
  return 0;
}
```

Exemple en Python

```
#!/usr/bin/env python
"""
for python3
"""

import mpi4py.MPI as MPI

comm = MPI.COMM_WORLD
rang = comm.Get_rank()
nprocs = comm.Get_size()

if rang == 0:
   data = [[1.0, 2.0, 3.0, 4.0, 4.5], \
           [5.0, 6.0, 7.0, 8.0, 9.5], \
           [9.0, 10.0, 11.0, 12.0, 12.5], \
           [13.0, 14.0, 15.0, 16.0, 16.5]]
else:
   data = None

data = comm.scatter(data, root=0)
print('rank=',rang,'Results:',data)
```

Exemple en Python avec NumPy

```
#!/usr/bin/env python
"""
for python3
"""

import mpi4py.MPI as MPI
import numpy as np

comm = MPI.COMM_WORLD
rang = comm.Get_rank()
nprocs = comm.Get_size()

my_N = 5
N = my_N * nprocs

if rang == 0:
   data = np.array([[1.0, 2.0, 3.0, 4.0, 4.5], \
           [5.0, 6.0, 7.0, 8.0, 9.5], \
           [9.0, 10.0, 11.0, 12.0, 12.5], \
           [13.0, 14.0, 15.0, 16.0, 16.5]], dtype=np.float64)
else:
   data = np.empty(N, dtype=np.float64)

recv_data = np.empty(my_N, dtype=np.float64)
comm.Scatter([data,my_N,MPI.DOUBLE], [recv_data,my_N,MPI.DOUBLE], root=0)
print('rank=',rang,'Results:',recv_data)
```

Distribution sélective de données inhomogènes

Un tableau supplémentaire donne le pas des paquets à distribuer.

Fortran

```
CALL MPI_SCATTERV(sendbuf, sendcount, pas, sendtype, &
                  recvbuf, recvcount, recvtype, &
                  root, comm, ierr)
```

C

```
MPI_Scatterv(sendbuf, sendcount, pas, sendtype,
             recvbuf, recvcount, recvtype,
             root, comm)
```

Python

```
comm = MPI.COMM_WORLD
```

```
  comm.Scatterv([senddata, data_size, pas, data_type], recvbuff, root)
```

Exemple : distribution d'un tableau inhomogène sur 4 processus

Objectif : distribuer les 4 lignes de ce tableau sur 4 processus.

$$\begin{pmatrix} 1.0 & 2.0 & 3.0 & & \\ 5.0 & 6.0 & 7.0 & 8.0 & 8.5 \\ 9.0 & & & & \\ 13.0 & 14.0 & & & \end{pmatrix}$$

Exécution :

```
mpirun -np 4 ./a.out
```

```
mpirun -np 4 python script.py
```

Résultat :

```
rank= 0 Results: [1. 2. 3.]
rank= 2 Results: [9.]
rank= 1 Results: [5. 6. 7. 8. 8.5]
rank= 3 Results: [13. 14.]
```

Exemple en Fortran

```
program scatterv

   use mpi
   implicit none

   integer, parameter :: SIZE_V=11,SIZE_Y=4
   integer :: i, numtasks, rank, ierr

   real*4 :: sendbuf(SIZE_V)
   integer :: chunk(0:SIZE_Y-1), displ(0:SIZE_Y-1), my_chunk
   real*4, dimension(:), allocatable :: recvbuf
   data sendbuf /1.0, 2.0, 3.0, &
                5.0, 6.0, 7.0, 8.0, 8.5, &
                9.0, &
                13.0, 14.0 /

   call MPI_INIT(ierr)
   call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
   call MPI_COMM_SIZE(MPI_COMM_WORLD, numtasks, ierr)

   if (numtasks .eq. SIZE_Y) then
```

```
    if (rank .eq. 0) then
      chunk = (/3, 5, 1, 2/)
      displ(0) = 0
      do i=1, SIZE_Y-1
        displ(i) = displ(i-1) + chunk(i-1)
      end do
    end if
    call MPI_BCAST(chunk,SIZE_Y,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
    call MPI_BCAST(displ,SIZE_Y,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
    call MPI_BARRIER(MPI_COMM_WORLD,ierr)

    my_chunk = chunk(rank)
    print *, 'RK= ',rank,' chunk ',my_chunk
    allocate(recvbuf(my_chunk))
    call MPI_SCATTERV(sendbuf, chunk, displ, MPI_REAL, &
                      recvbuf, my_chunk, MPI_REAL, &
                      0, MPI_COMM_WORLD, ierr)

    print *, 'rank= ',rank,' Results: ',recvbuf

    deallocate(recvbuf)
  else
    print *, 'Must specify',SIZE_Y,' processors. Terminating.'
  end if

  call MPI_FINALIZE(ierr)

  end program scatterv
```

Exemple en C

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define SIZE_X 5
#define SIZE 4

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

  int numtasks, rank;
  int sum = 0;
  float sendbuf[11] = { 1.0, 2.0, 3.0,
    5.0, 6.0, 7.0, 8.0, 8.5,
    9.0,
    13.0, 14.0 };
  int chunk[SIZE] = {3, 5, 1, 2};
  float recvbuf[SIZE_X];

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

  int *displ = malloc(sizeof(int)*numtasks);
  if (numtasks == SIZE) {
```

```
    for (int i = 0; i < numtasks; i++) {
      displ[i] = sum;
      sum += chunk[i];
    }
    if (rank == 0) {
      for (int i = 0; i < numtasks; i++) {
        printf("chunk[%d] = %d\tdispl[%d] = %d\n",i,chunk[i],i,displ[i]);
      }
    }

    MPI_Scatterv(&sendbuf,chunk, displ, MPI_FLOAT,&recvbuf,chunk[rank],
                 MPI_FLOAT,0,MPI_COMM_WORLD);

    MPI_Barrier(MPI_COMM_WORLD);
    printf("rank= %d ", rank);
    for (int i = 0; i < chunk[rank]; i++) {
        printf("%f\t", recvbuf[i]);
    }
    printf("\n");
  } else {
    printf("Must specify %d processors. Terminating.\n",SIZE);
  }

  MPI_Finalize();

  free(displ);

  return 0;
}
```

Exemple en Python

```
#!/usr/bin/env python
""" for python3 """

import mpi4py.MPI as MPI

comm = MPI.COMM_WORLD
rang = comm.Get_rank()
nprocs = comm.Get_size()

if rang == 0:
   data = [[1.0, 2.0, 3.0], \
           [5.0, 6.0, 7.0, 8.0, 8.5], \
           [9.0], \
           [13.0, 14.0]]
else:
   data = None

data = comm.scatter(data, root=0)
print('rank=',rang,'Results:',data)
```

Attention : ici ce sont les lignes qui sont distribuées (il y a le même nombre de lignes que de processus)

```
>>> len(data)
4
```

Exemple en Python avec NumPy

```
#!/usr/bin/env python
""" for python3 """

import mpi4py.MPI as MPI
import numpy as np

comm = MPI.COMM_WORLD
rang = comm.Get_rank()
nprocs = comm.Get_size()

if (nprocs != 4):
    if (rang == 0):
        print('Stop.\n Required: 4 MPI processes')
    exit()
```

```
N = 11
if rang == 0:
   data = np.array([1.0, 2.0, 3.0, \
          5.0, 6.0, 7.0, 8.0, 8.5, \
          9.0, \
          13.0, 14.0], dtype=np.float64)
   chunk = np.array([3, 5, 1, 2], dtype=np.int)
   disp = np.insert(np.cumsum(chunk), 0, 0)[0:-1]
else:
   data = np.zeros(N, dtype=np.float64)
   chunk = np.zeros(nprocs, dtype=np.int)
   disp = np.zeros(nprocs, dtype=np.int)

comm.Bcast(chunk, root = 0)
comm.Bcast(disp, root = 0)

dest = np.zeros(chunk[rang], dtype=np.float64)
comm.Scatterv([data,chunk,disp,MPI.DOUBLE], dest)
print('rank=',rang,'Results:',dest)
```

- La distribution de données réparties se fait avec les commandes de type `SCATTER` et `SCATTERV`.
- La collecte utilise les commandes `GATHER` et `GATHERV`.
- La collecte par tous les processus de données distribuées est basé sur les commandes `ALLGATHER` et `ALLGATHERV`.
- `ALLTOALL` et `ALLTOALLV` permettent à tous les processus de faire autant d'envoi que de réception.

Remarque sur la décomposition de domaine

Il est préférable de travailler avec une décomposition du domaine qui assure le meilleur équilibrage de charge entre les différents processus.

- Soit N la taille du tableau structuré à décomposer entre les différents processus.
- Soit p le nombre total de processus.

En supposant que $p \leq N$, la répartition des charges est équilibrée si le reste de la division de N par p est nul, $r = mod(N, p)$.
Sinon, nécessairement $r < N/p$ et il est intéressant de répartir le surplus sur l'ensemble des processus, en respectant la localité spatiale ad-hoc à la structure de donnée initiale.
Une solution consiste donc à augmenter d'un élément les r premières partitions (tout en décalant les curseurs de début et de fin de chaque partie).

Exemple en Fortran

```
INTEGER, parameter :: N=1001
INTEGER :: rank, nprocs, ierr
INTEGER :: nstart, nstop, npart, ncount, nrem

CALL MPI_INIT(ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
CALL MPI_Comm_size(MPI_COMM_WORLD, nprocs, ierr)

ncount = N/nprocs
nrem = MOD(N,nprocs)
IF (rank < nrem) THEN
  nstart = rank * (ncount + 1)
  nstop = nstart + ncount
ELSE
  nstart = rank * ncount + nrem
  nstop = nstart + (ncount - 1)
END IF

npart = nstop-nstart+1
```

Exemple en C

```
int nprocs, rank;

MPI_Init(&argc,&argv);

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

int N=1001;

int count = N/nprocs;
int remainder = N%nprocs;
int start, stop;

if (rank < remainder) {
  // The first 'remainder' ranks get 'count + 1' tasks each
  start = rank * (count + 1);
  stop = start + count;
} else {
  // The remaining 'size - remainder' ranks get 'count' task each
  start = rank * count + remainder;
  stop = start + (count - 1);
}

int npart = stop-start+1;
```

Exemple en Python

```
#!/usr/bin/env python
""" for python3 """

# Give me my start and end index of an array of size N using my rank
# from A. Gomez TACC
def partition (rank, size, N):
    n = N//size + ((N%size ) > rank )
    s = rank * (N//size )
    if (N%size ) > rank:
        s += rank
    else:
        s += N%size
    return s,s+n-1
```

Opérations de réduction

Elles effectuent des opérations sur les données transférées, avec éventuellement diffusion du résultat (i.e. combinées avec un broadcast).

Fortran

```
CALL MPI_REDUCE(SENDBUF, RECVBUF, COUNT, DATATYPE, OP, ROOT, COMM, ierr)
CALL MPI_ALLREDUCE(SENDBUF, RECVBUF, COUNT, DATATYPE, OP, COMM, ierr)
```

C

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

Python

```
comm = MPI.COMM_WORLD
```

```
comm.reduce(sendobj=None, recvobj=None, op=MPI.SUM, root=0)
comm.allreduce(sendobj=None, recvobj=None, op=MPI.SUM)
```

```
comm.Reduce(sendbuf, recvbuf, op=MPI.SUM, root=0)
comm.Allreduce(sendbuf, recvbuf, op=MPI.SUM)
```

Principales opérations de réduction

`MPI.MIN`	min
`MPI.MAX`	max
`MPI.SUM`	somme
`MPI.PROD`	produit
`MPI.MAXLOC`	index valeur max
`MPI.MINLOC`	index valeur min

(syntaxe `Python`)

Exemple d'utilisation de la réduction

Faire la somme des éléments (unitaires) d'un vecteur de taille $N = 1001$. Le vecteur est décomposé en p partitions, p étant le nombre de processus initialisés par MPI.

Exécution pour $p = 4$:

```
mpirun -np 4 ./a.out
```

```
mpirun -np 4 python script.py
```

Résultat :

```
[ 0 ] part : 251.000000
[ 2 ] part : 250.000000
[ 3 ] part : 250.000000
[ 1 ] part : 250.000000
Sum : 1001.000000
```

Exemple en Fortran

```
PROGRAM main

  USE mpi
  IMPLICIT NONE

  INTEGER, parameter :: N=1001
  INTEGER :: ierr, i, rank, nprocs
  INTEGER :: nstart, nstop, npart, ncount, nrem
  DOUBLE PRECISION, allocatable :: vec(:)
  DOUBLE PRECISION :: local_sum, total_sum

  CALL MPI_INIT(ierr)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
  CALL MPI_Comm_size(MPI_COMM_WORLD, nprocs, ierr)

  ncount = N/nprocs
  nrem = MOD(N,nprocs)
  IF (rank < nrem) THEN
    nstart = rank * (ncount + 1)
    nstop = nstart + ncount
  ELSE
    nstart = rank * ncount + nrem
    nstop = nstart + (ncount - 1)
  END IF
  npart = nstop-nstart+1

  allocate(vec(npart))
```

```
  DO i=1,npart
    vec(i) = 1.0D0
  END DO
  local_sum = 0.0D0
  DO i=1,npart
    local_sum = local_sum + vec(i)
  END DO
  WRITE(*,'(A,I3,A,F15.8)') "[",rank,"] part : ",local_sum
  CALL MPI_BARRIER(MPI_COMM_WORLD,ierr)

  CALL MPI_REDUCE(local_sum, total_sum, 1, MPI_DOUBLE, MPI_SUM, 0 , MPI_COMM_WORLD
       , ierr)

  IF (rank.eq.0) THEN
    WRITE(*,*) "Sum : ", total_sum
  END IF

  deallocate(vec)

  CALL MPI_FINALIZE(ierr)

END PROGRAM
```

Exemple en C

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

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

  int i, N;
  int nprocs, rank;

  MPI_Init(&argc,&argv);

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

  double *vec;
  double local_sum,sum;

  N=1001;

  int count = N/nprocs;
  int remainder = N%nprocs;
  int start, stop;
```

```
  if (rank < remainder) {
    start = rank * (count + 1);
    stop = start + count;
  } else {
    start = rank * count + remainder;
    stop = start + (count - 1);
  }

  int npart = stop-start+1;
  vec = malloc(sizeof(double)*npart);

  for (i=0; i<npart; i++) {
    vec[i] = 1.0;
  }
  local_sum = 0.0;
  for (i=0; i<npart; i++) {
    local_sum += vec[i];
  }
  printf("[ %d ] part : %g\n",rank,local_sum);
  MPI_Barrier(MPI_COMM_WORLD);

  MPI_Reduce(&local_sum,&sum,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);

  if (rank == 0) {
    printf("Sum : %g\n",sum);
  }
  MPI_Finalize();
  return 0;
}
```

Exemple en Python

```
#!/usr/bin/env python
""" for python3 """

import mpi4py.MPI as MPI
import numpy as np
import part as part

comm = MPI.COMM_WORLD
nprocs = comm.size
rank = comm.rank

# Define the size of the problem
N = 1001
start,end = part.partition(rank,nprocs,N)
vec = np.ones((end-start+1),dtype=np.float64)

# Calculate the local sum of local vectors
local_sum = vec.sum()
print("[ %d ] part : %f"%(rank, local_sum))
comm.barrier()

# Get the global sum
global_sum = comm.reduce(local_sum, op=MPI.SUM, root=0)
if rank == 0:
    print("Sum : %f"%(global_sum))
```

Exemple en Python avec NumPy

```
#!/usr/bin/env python
""" for python3 """

import mpi4py.MPI as MPI
import numpy as np
import part as part

comm = MPI.COMM_WORLD
nprocs = comm.size
rank = comm.rank

# Define the size of the problem
N = 1001
start,end = part.partition(rank,nprocs,N)
vec = np.ones((end-start+1),dtype=np.float64)

# Calculate the local sum of local vectors
local_sum = vec.sum()
print("[ %d ] part : %f"%(rank, local_sum))
comm.Barrier()

# Get the global sum
global_sum = np.zeros(1, dtype='float64')
comm.Reduce(local_sum, global_sum, op=MPI.SUM, root=0)
if rank == 0:
    print("Sum : %f"%(global_sum[0]))
```

Autres fonctionnalités

La bibliothèque `MPI` permet de faire d'autres types d'opérations que celles exposées dans cette introduction, notamment

- D'échanger des types de données dérivés, hétérogènes
- De décomposer, créer des communicateurs
- De spécifier des topologies adaptées
- Des entrées/sorties parallèles

Références

- Parallélisme sur machines à mémoire distribuée, Bastien Di Pierro, LyonCalcul, 2016
- Calcul parallèle avec MPI, Guy Moebs, Univ. Nantes, 2010
- Formations MPI, IDRIS
- https://computing.llnl.gov/tutorials/mpi
- https://www.mpi-forum.org/docs
- Documentation OpenMPI
- Documentation mpi4py