Aide mémoire MPI en Fortran

Version 4.6 – 01 juin 2015

1 Environnement

1.1 Initialisation de l'environnement MPI

```
integer :: code
call MPI_INIT(<OUT> code)
```

1.2 Rang du processus

```
integer :: comm, rang, code
call MPI_COMM_RANK(<IN> comm, <OUT> rang, <OUT> code)
```

1.3 Nombre de processus

```
integer :: comm, nb_procs, code
call MPI_COMM_SIZE(<IN> comm, <OUT> nb_procs, <OUT> code)
```

1.4 Désactivation de l'environnement MPI

```
integer :: code
call MPI_FINALIZE(<OUT> code)
```

1.5 Arrêt d'un programme MPI

```
integer :: comm, error, code
call MPI_ABORT(<IN> comm, <IN> error, <OUT> code)
```

1.6 Prise de temps

```
real(kind=8) :: temps
temps=MPI_WTIME()
```

2 Communications point à point

2.1 Envoi de message

```
<type et attribut>:: message
integer :: longueur, type, rang_dest
integer :: etiquette, comm, code
call MPI_SEND(
        <IN>    message,
        <IN>    longueur,
        <IN>    type,
        <IN>    rang_dest,
        <IN>    etiquette,
        <IN>    comm,
        <OUT>   code)
```

2.2 Envoi non bloquant de message

```
<type et attribut>:: message
integer :: longueur, type, rang_dest
integer :: etiquette, comm, requete, code
call MPI_ISEND(
        <IN>    message,
        <IN>    longueur,
        <IN>    type,
        <IN>    rang_dest,
        <IN>    etiquette,
        <IN>    comm,
        <OUT>   requete,
        <OUT>   code)
```

2.3 Réception de message

```
<type et attribut>:: message
integer :: longueur, type, rang_source
integer :: etiquette, comm, code
integer, dimension(MPI_STATUS_SIZE) :: statut
call MPI_RECV(
        <OUT>   message,
        <IN>    longueur,
        <IN>    type,
        <IN>    rang_source,
        <IN>    etiquette,
        <IN>    comm,
        <OUT>   statut,
        <OUT>   code)
```

2.4 Réception non bloquant de message

```
<type et attribut>:: message
integer :: longueur, type, rang_source
integer :: etiquette, comm, requete, code
call MPI_IRECV(
        <OUT>   message,
        <IN>    longueur,
        <IN>    type,
        <IN>    rang_source,
        <IN>    etiquette,
        <IN>    comm,
        <OUT>   requete,
        <OUT>   code)
```

2.5 Envoi et réception de message

```
<type et attribut>:: message_emis, message_recu
integer :: longueur_message_emis, type_message_emis
integer :: etiquette_message_emis, etiquette_message_recu
integer :: longueur_message_recu, type_message_recu
integer :: rang_source, rang_dest, comm, code
integer, dimension(MPI_STATUS_SIZE) :: statut
call MPI_SENDRECV(
        <IN>    message_emis,
        <IN>    longueur_message_emis,
        <IN>    type_message_emis,
        <IN>    rang_dest,
        <IN>    etiquette_message_emis,
        <OUT>   message_recu,
        <IN>    longueur_message_recu,
        <IN>    type_message_recu,
        <IN>    rang_source,
        <IN>    etiquette_message_recu,
        <IN>    comm,
        <OUT>   statut,
        <OUT>   code)
.....

<type et attribut>:: message_emis_recu
integer :: longueur, type, rang_dest, rang_source
integer :: etiquette_message_emis, etiquette_message_recu
integer :: comm, code
integer, dimension(MPI_STATUS_SIZE) :: statut
call MPI_SENDRECV_REPLACE(
        <INOUT>   message_emis_recu,
        <IN>      longueur,
        <IN>      type,
        <IN>      rang_dest,
        <IN>      etiquette_message_emis,
        <IN>      rang_source,
        <IN>      etiquette_message_recu,
        <IN>      comm,
        <OUT>     statut,
        <OUT>     code)
```

2.6 Attente de la fin d'une communication non bloquante

```
integer :: requete, code
integer, dimension(MPI_STATUS_SIZE) :: statut
call MPI_WAIT(<IN> requete, <OUT> statut, <OUT> code)
```

2.7 Test de la fin d'une communication non bloquante

```
integer :: requete, code
logical :: drapeau
integer, dimension(MPI_STATUS_SIZE) :: statut
call MPI_TEST(
        <IN>    requete,
        <OUT>   drapeau,
        <OUT>   statut,
        <OUT>   code)
```

3 Communications collectives

3.1 Diffusion générale

```
<type et attribut>:: message
integer :: longueur, type, rang_source, comm, code
call MPI_BCAST(
        <INOUT>   messsage,
        <IN>      longueur,
        <IN>      type,
        <IN>      rang_source,
        <IN>      comm,
        <OUT>     code)
```

3.2 Diffusion séléctive

```
<type et attribut>:: message_a_repartir, message_recu
integer :: longueur_message_emis, longueur_message_recu
integer :: type_message_emis, type_message_recu
integer :: rang_source, comm, code
call MPI_SCATTER(
        <IN>    message_a_repartir,
        <IN>    longueur_message_emis,
        <IN>    type_message_emis,
        <OUT>   message_recu,
        <IN>    longueur_message_recu,
        <IN>    type_message_recu,
        <IN>    rang_source,
        <IN>    comm,
        <OUT>   code)
```

3.3 Collecte

```
<type et attribut>:: message_emis, message_collecte
integer :: longueur_message_emis, longueur_message_recu
integer :: type_message_emis, type_message_recu
integer :: rang_dest, comm, code
call MPI_GATHER(
        <IN>    message_emis,
        <IN>    longueur_message_emis,
        <IN>    type_message_emis,
        <OUT>   message_collecte,
        <IN>    longueur_message_recu,
        <IN>    type_message_recu,
        <IN>    rang_dest,
        <IN>    comm,
        <OUT>   code)
```

.....

```
<type et attribut>:: message_a_repartir, message_recu
integer :: longueur_message_emis, longueur_message_recu
integer :: type_message_emis, type_message_recu
integer :: comm, code
call MPI_ALLGATHER(
        <IN>    message_emis,
        <IN>    longueur_message_emis,
        <IN>    type_message_emis,
        <OUT>   message_collecte,
        <IN>    longueur_message_recu,
        <IN>    type_message_recu,
        <IN>    comm,
        <OUT>   code)
```

3.4 Collecte et diffusion

```
<type et attribut>:: message_a_repartir, message_collecte
integer :: longueur_message_emis, longueur_message_recu
integer :: type_message_emis, type_message_recu
integer :: comm, code
call MPI_ALLTOALL(
        <IN>    message_a_repartir,
        <IN>    longueur_message_emis,
        <IN>    type_message_emis,
        <OUT>   message_collecte,
        <IN>    longueur_message_recu,
        <IN>    type_message_recu,
        <IN>    comm,
        <OUT>   code)
```

3.5 Réduction

```
<type et attribut>:: message_emis, message_recu
integer :: longueur, type, rang_dest
integer :: operation, comm, code
call MPI_REDUCE(
        <IN>    message_emis,
        <OUT>   message_recu,
        <IN>    longueur,
        <IN>    type,
        <IN>    operation,
        <IN>    rang_dest,
        <IN>    comm,
        <OUT>   code)
```

operation ≡ MPI_MAX | MPI_MIN | MPI_SUM | MPI_PROD | MPI_BAND | MPI_BOR | MPI_BXOR | MPI_LAND | MPI_LOR | MPI_LXOR

.....

```
<type et attribut>:: message_emis, message_recu
integer :: longueur, type, operation, comm, code
call MPI_ALLREDUCE(
        <IN>    message_emis,
        <OUT>   message_recu,
        <IN>    longueur,
        <IN>    type,
        <IN>    operation,
        <IN>    comm,
        <OUT>   code)
```

3.6 Synchonisation globale

```
integer :: comm, code
call MPI_BARRIER(<IN> comm, <OUT> code)
```

4 Types dérivés

4.1 Types contigus

```
integer :: nb_elements, ancien_type, nouveau_type, code
call MPI_TYPE_CONTIGUOUS(
        <IN>    nb_elements,
        <IN>    ancien_type,
        <OUT>   nouveau_type,
        <OUT>   code)
```

4.2 Types avec un pas constant

```
integer :: nb_elements, longueur_bloc
integer :: pas, ancien_type, nouveau_type, code
call MPI_TYPE_VECTOR(
        <IN>    nombre_bloc,
        <IN>    nbr_elt_par_bloc,
        <IN>    pas,
        <IN>    type_elt,
        <OUT>   nouveau_type,
        <OUT>   code)
```

.....

```
integer :: nb_elements,longueur_bloc
integer(MPI_ADDRESS_KIND) :: pas
integer :: ancien_type, nouveau_type, code
call MPI_TYPE_CREATE_HVECTOR(
        <IN>    nombre_bloc,
        <IN>    nbr_elt_par_bloc,
        <IN>    pas,
        <IN>    type_elt,
        <OUT>   nouveau_type,
        <OUT>   code)
```

4.3 Types à pas variable

```
integer :: nb_elements, code
integer, dimension(nb_elements) :: longueur_bloc, pas
integer :: ancien_type, nouveau_type
call MPI_TYPE_INDEXED(
        <IN>    nb_elements,
        <IN>    longueur_bloc,
        <IN>    pas,
        <IN>    ancien_type,
        <OUT>   nouveau_type,
        <OUT>   code)
```

4.4 Types sous-tableau

```
integer :: nb_dims, adresse_debut, ordre
integer :: ancien_type, nouveau_type, code
integer, dimension(nb_dims) :: profil_tab, profil_sous_tab
integer, dimension(nb_dims) :: adresse_debut
call MPI_TYPE_CREATE_SUBARRAY(
        <IN>    nb_dims,
        <IN>    profil_tab,
        <IN>    profil_sous_tab,
        <IN>    adresse_debut,
        <IN>    ordre,
        <IN>    ancien_type,
        <OUT>   nouveau_type,
        <OUT>   code)
```

4.5 Types hétérogènes

```
integer :: nb_elements, nouveau_type, code
integer, dimension(nb_elements) :: longueur_bloc
integer(MPI_ADDRESS_KIND),dimension(nb_elements) :: pas
integer, dimension(nb_elements) :: ancien_types
call MPI_TYPE_CREATE_STRUCT(
        <IN>    nb_elements,
        <IN>    longueur_bloc,
        <IN>    pas,
        <IN>    ancien_types,
        <OUT>   nouveau_type,
        <OUT>   code)
```

4.6 Validation des types

```
integer :: type, code
call MPI_TYPE_COMMIT(<IN> type, <OUT> code)
```

4.7 Étendue

```
integer :: type, code
integer(MPI_ADDRESS_KIND) :: borne_inf_alignee
integer(MPI_ADDRESS_KIND) :: taille_alignee
call MPI_TYPE_GET_EXTENT(
        <IN> type,
        <OUT> borne_inf_alignee,
        <OUT> taille_alignee,
        <OUT> code)
```

.....

```
integer :: ancien_type, nouveau_type, code
integer(MPI_ADDRESS_KIND) :: nouvelle_borne_inf
integer(MPI_ADDRESS_KIND) :: nouvelle_taille
call MPI_TYPE_CREATE_RESIZED(
        <IN> ancien_type,
        <IN> nouvelle_borne_inf,
        <IN> nouvelle_taille,
        <OUT> nouveau_type,
        <OUT> code)
```

4.8 Taille d'un type

```
integer :: type, taille, code
call MPI_TYPE_SIZE(<IN> type, <OUT> taille, <OUT> code)
```

5 Communicateur

5.1 Partitionnement d'un communicateur

```
integer :: comm, couleur, cle
integer :: nouveau_comm, code
call MPI_COMM_SPLIT(
        <IN>    comm,
        <IN>    couleur,
        <IN>    cle,
        <OUT>   nouveau_comm,
        <OUT>   code)
```

5.2 Création d'une topologie cartésienne

```
integer :: comm, nb_dims, nouveau_comm, code
integer, dimension(nb_dims) :: dims
logical :: periodique, reorganise
call MPI_CART_CREATE(
        <IN>    comm,
        <IN>    nb_dims,
        <IN>    dims,
        <IN>    periodique,
        <IN>    reorganise,
        <OUT>   nouveau_comm,
        <OUT>   code)
```

5.3 Distribution des processus

```
integer :: nb_procs, nb_dims, code
integer, dimension(nb_dims) :: dims
call MPI_DIMS_CREATE(
        <IN>      nb_procs,
        <IN>      nb_dims,
        <INOUT>   dims,
        <OUT>     code)
```

5.4 Rang d'un processus

```
integer :: comm, rang, code, nb_dims
integer, dimension(nb_dims) :: coords
call MPI_CART_RANK(
        <IN>    comm,
        <IN>    coords,
        <OUT>   rang,
        <OUT>   code)
```

5.5 Coordonnées d'un processus

```
integer :: comm, rang, nb_dims, code
integer, dimension(nb_dims) :: coords
call MPI_CART_COORDS(
        <IN>    comm,
        <IN>    rang,
        <IN>    nb_dims,
        <OUT>   coords,
        <OUT>   code)
```

5.6 Rang des voisins

```
integer :: comm, direction, sens
integer :: rang_source, rang_dest, code
call MPI_CART_SHIFT(
        <IN>    comm,
        <IN>    direction,
        <IN>    pas,
        <OUT>   rang_source,
        <OUT>   rang_dest,
        <OUT>   code)
```

5.7 Subdivision d'une topologie

```
integer :: comm, comm_sub, code, nb_dims
logical, dimension(nb_dims) :: dim_sub
call MPI_CART_SUB(
        <IN>    comm,
        <IN>    dim_sub,
        <OUT>   comm_sub,
        <OUT>   code)
```

6 MPI-IO

6.1 Ouverture d'un fichier

```
integer :: comm, attribut, info, descripteur, code
character(len=*) :: fichier
call MPI_FILE_OPEN(
        <IN>    comm,
        <IN>    fichier,
        <IN>    attribut,
        <IN>    info,
        <OUT>   descripteur,
        <OUT>   code)
```

6.2 Fermeture d'un fichier

```
integer :: descripteur, code
call MPI_FILE_CLOSE(
        <INOUT>   descripteur,
        <OUT>     code)
```

6.3 Changement de la vue

```
integer :: descripteur, type_derive, motif, info, code
integer(kind=MPI_OFFSET_KIND) :: deplacement_initial
character(len=*) :: mode
call MPI_FILE_SET_VIEW(
        <INOUT>   descripteur,
        <IN>      deplacement_initial,
        <IN>      type_derive,
        <IN>      motif,
        <IN>      mode,
        <IN>      info,
        <OUT>     code)
```

6.4 Pointeur individuels

6.4.1 Positionnement du pointeur

```
integer :: descripteur, mode_seek, code
integer(kind=MPI_OFFSET_KIND) :: position_fichier
call MPI_FILE_SEEK(
        <INOUT>   descripteur,
        <IN>      position_fichier,
        <IN>      mode_seek,
        <OUT>     code)
```

6.4.2 Lecture non bloquante

```
integer :: descripteur, nb_valeurs, type_derive, requete
integer :: code
<type et attributs>:: valeurs
call MPI_FILE_IREAD(
        <IN>    descripteur,
        <OUT>   valeurs,
        <IN>    nb_valeurs,
        <IN>    type_derive,
        <OUT>   requete,
        <OUT>   code)
```

6.4.3 Lecture

```
integer :: descripteur, nb_valeurs, type_derive, code
<type et attributs>:: valeurs
integer, dimension(MPI_STATUS_SIZE) :: statut
call MPI_FILE_READ(
        <IN>    descripteur,
        <OUT>   valeurs,
        <IN>    nb_valeurs,
        <IN>    type_derive,
        <OUT>   statut,
        <OUT>   code)
```

6.4.4 Lecture collective

```
integer :: descripteur, nb_valeurs, type_derive, code
<type et attributs>:: valeurs
integer, dimension(MPI_STATUS_SIZE) :: statut
call MPI_FILE_READ_ALL(
        <IN>    descripteur,
        <OUT>   valeurs,
        <IN>    nb_valeurs,
        <IN>    type_derive,
        <OUT>   statut,
        <OUT>   code)
```

6.4.5 Écriture

```
integer :: descripteur, nb_valeurs, type_derive, code
<type et attributs>:: valeurs
integer, dimension(MPI_STATUS_SIZE) :: statut
call MPI_FILE_WRITE(
        <IN>    descripteur,
        <OUT>   valeurs,
        <IN>    nb_valeurs,
        <IN>    type_derive,
        <OUT>   statut,
        <OUT>   code)
```

6.4.6 Écriture collective

```
integer :: descripteur, nb_valeurs, type_derive, code
<type et attributs>:: valeurs
integer, dimension(MPI_STATUS_SIZE) :: statut
call MPI_FILE_WRITE_ALL(
        <IN>    descripteur,
        <OUT>   valeurs,
        <IN>    nb_valeurs,
        <IN>    type_derive,
        <OUT>   statut,
        <OUT>   code)
```

6.5 Adresses explicites

6.5.1 Lecture non bloquante

```
integer :: descripteur, nb_valeurs, type_derive, requete
integer :: code
integer(kind=MPI_OFFSET_KIND) :: position_fichier
<type et attributs>:: valeurs
call MPI_FILE_IREAD_AT(
        <IN>    descripteur,
        <IN>    position_fichier,
        <OUT>   valeurs,
        <IN>    nb_valeurs,
        <IN>    type_derive,
        <OUT>   requete,
        <OUT>   code)
```

6.5.2 Lecture

```
integer :: descripteur, nb_valeurs, type_derive, code
integer(kind=MPI_OFFSET_KIND) :: position_fichier
<type et attributs>:: valeurs
integer, dimension(MPI_STATUS_SIZE) :: statut
call MPI_FILE_READ_AT(
        <IN>    descripteur,
        <IN>    position_fichier,
        <OUT>   valeurs,
        <IN>    nb_valeurs,
        <IN>    type_derive,
        <OUT>   statut,
        <OUT>   code)
```

6.5.3 Lecture collective

```
integer :: descripteur, nb_valeurs, type_derive, code
integer(kind=MPI_OFFSET_KIND) :: position_fichier
<type et attributs>:: valeurs
integer, dimension(MPI_STATUS_SIZE) :: statut
call MPI_FILE_READ_AT_ALL(
        <IN>    descripteur,
        <IN>    position_fichier,
        <OUT>   valeurs,
        <IN>    nb_valeurs,
        <IN>    type_derive,
        <OUT>   statut,
        <OUT>   code)
```

6.5.4 Écriture

```
integer :: descripteur, nb_valeurs, type_derive, code
integer(kind=MPI_OFFSET_KIND) :: position_fichier
<type et attributs>:: valeurs
integer, dimension(MPI_STATUS_SIZE) :: statut
call MPI_FILE_WRITE_AT(
        <IN>    descripteur,
        <IN>    position_fichier,
        <IN>    valeurs,
        <IN>    nb_valeurs,
        <IN>    type_derive,
        <OUT>   statut,
        <OUT>   code)
```

6.6 Pointeurs partagés

6.6.1 Positionnement du pointeur

```
integer :: descripteur, mode_seek, code
integer(kind=MPI_OFFSET_KIND) :: position_fichier
call MPI_FILE_SEEK_SHARED(
        <INOUT>   descripteur,
        <IN>      position_fichier,
        <IN>      mode_seek,
        <OUT>     code)
```

6.6.2 Lecture

```
integer :: descripteur, nb_valeurs, type_derive, code
<type et attributs>:: valeurs
integer, dimension(MPI_STATUS_SIZE) :: statut
call MPI_FILE_READ_SHARED(
        <IN>    descripteur,
        <OUT>   valeurs,
        <IN>    nb_valeurs,
        <IN>    type_derive,
        <OUT>   statut,
        <OUT>   code)
```

6.6.3 Lecture collective

```
integer :: descripteur, nb_valeurs, type_derive, code
<type et attributs>:: valeurs
integer, dimension(MPI_STATUS_SIZE) :: statut
call MPI_FILE_READ_ORDERED(
        <IN>    descripteur,
        <OUT>   valeurs,
        <IN>    nb_valeurs,
        <IN>    type_derive,
        <OUT>   statut,
        <OUT>   code)
```

6.6.4 Lecture collective non bloquante

```
integer :: descripteur, nb_valeurs, type_derive, code
<type et attributs>:: valeurs
call MPI_FILE_READ_ORDERED_BEGIN(
        <IN>    descripteur,
        <OUT>   valeurs,
        <IN>    nb_valeurs,
        <IN>    type_derive,
        <OUT>   code)

integer :: descripteur, code
<type et attributs>:: valeurs
integer, dimension(MPI_STATUS_SIZE) :: statut
call MPI_FILE_READ_ORDERED_END(
        <IN>    descripteur,
        <OUT>   valeurs,
        <OUT>   statut,
        <OUT>   code)
```

7 Constantes symboliques

MPI_ANY_TAG, MPI_ANY_SOURCE, MPI_SUCCESS, MPI_STATUS_IGNORE
MPI_CHARACTER, MPI_LOGICAL, MPI_INTEGER,
MPI_REAL, MPI_DOUBLE_PRECISION, MPI_COMPLEX,
MPI_COMM_NULL, MPI_COMM_WORLD,
MPI_PROC_NULL, MPI_STATUS_SIZE, MPI_UNDEFINED