Tutoriel : utilisation de MPI en Python

Camille Coti

14 octobre 2016

Il existe des bindings MPI non-officiels pour Python. On peut par exemple utiliser mpi4py [1].

1 Lancement d'un programme

Les scripts Python sont interprétés par l'interpréteur Python. On le passe donc en argument du mpiexec ou srun / salloc, qui va lancer sur les machines distantes l'interpréteur qui lui-même va exécuter le script. On lance donc :

```
1  $ mpiexec -n 8 python helloWorld.py
```

2 Appel de fonctions MPI

On charge le module mpi4py en début de programme pour pouvoir utiliser les fonctions MPI :

```
1 from mpi4py import MPI
```

Attention : il ne fait pas faire de `MPI_Init` ni de `MPI_Finalize`. On commence directement avec des appels de fonction MPI.

2.1 Premier exemple : HelloWorld

Les opérations sont toujours effectuées sur un communicateur donné. Les communicateurs définis par la norme sont fournis : `MPI.COMM_WORLD`, `MPI.COMM_SELF` et `MPI.COMM_NULL`.

On récupère un objet de type `mpi4py.MPI.Intracomm` et on appelle les méthodes de cet objet.

Concrètement, la classe Intracomm hérite les méthodes de communications point-à-point et collectives de la classe Comm [2].

Un petit exemple complet affichant le rang de chaque processus et le nombre de processus :

```
1  #!/bin/python
2
3  from mpi4py import MPI
4
5  def main():
6      comm = MPI.COMM_WORLD
7      rank = comm.Get_rank()
8      size = comm.Get_size()
9      print "hello from " + str( rank ) + " in " + str( size )
10
11
12 if __name__ == "__main__":
13     main()
```

Son exécution :

1. `http://mpi4py.scipy.org/docs/usrman/index.html`
2. documentation de la classe Comm : `http://mpi4py.scipy.org/docs/apiref/mpi4py.MPI.Comm-class.html`

```
$ mpiexec -n 8 python helloWorld.py
hello from 0 in 8
hello from 4 in 8
hello from 1 in 8
hello from 2 in 8
hello from 3 in 8
hello from 5 in 8
hello from 6 in 8
hello from 7 in 8
```

2.2 Communications point-à-point

Il existe deux familles de routines de communications :

— pour envoyer des objets Python : le nom de la routine commence par une minuscule

— pour envoyer des buffers (plus rapide) : le nom de la routine commence par une majuscule

Attention : il existe des fonctions d'envoi non-bloquantes (Isend/isend), mais pas de fonctions de réception.

Exemple avec des objets Python : le processus de rang 0 envoie une chaine de caractères au processus de rang 1

```
#!/bin/python

from mpi4py import MPI

def main():
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()
    TAG = 40
    if 0 == rank:
        token = "Bonjour"
        comm.send( token, dest = 1, tag = TAG )
    if 1 == rank:
        token = comm.recv( source = 0, tag = TAG )
    print str( rank ) + "] Token : " + token

if __name__ == "__main__":
    main()
```

Pour envoyer un buffer, on utilise les tableaux de NumPy. À l'envoi on prépare un tableau, qui permet de packer les données pour que la bibliothèque MPI puisse directement les envoyer. En réception on doit préparer un tableau vide dans lequel la bibliothèque MPI mettra les données reçues.

Les buffers d'envoi et de réception sont passés en arguments des routines de communications MPI. Ils peuvent être passés directement, auquel cas le type de données manipulées est découvert automatiquement. On peut également fournir le type de données envoyées : dans ce cas, on passe un tuple contenant le buffer et son type. Les types de base sont des attributs de la classe MPI (`MPI.INT`, `MPI.DOUBLE`, etc).

Exemple avec un tableau contenant un entier :

```
  #!/bin/python

from mpi4py import MPI
import numpy
import os

def main():
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()
```

```
    TAG = 40
    if 0 == rank:
        token = numpy.arange( 1, dtype = 'i' )
        token[0] = os.getpid()
        comm.Send( [token, MPI.INT], dest = 1, tag = TAG )
    if 1 == rank:
        token = numpy.empty( 1, dtype = 'i' )
        comm.Recv( [token, MPI.INT], source = 0, tag = TAG )
    print str( rank ) + "] Token : " + str( token )

if __name__ == "__main__":
    main()
```

En cas de découverte automatique des types, les appels des fonctions d'envoi et réception deviennent :

```
comm.Send( token, dest = 1, tag = TAG )
comm.Recv( token, source = 0, tag = TAG )
```

Pour les fonctions de réception (recv et Recv), on peut donner comme émetteur `MPI.ANY_SOURCE` et/ou comme tag `MPI.ANY_TAG`. On récupère alors les informations dans le status. Attention, il faut au préalable l'avoir instancié.

```
#!/bin/python

from mpi4py import MPI

def main():
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()
    token = rank
    TAG = 40

    if 0 == rank:
        comm.send( token, 1, 40 )
    if 1 == rank:
        st = MPI.Status()
        token = comm.recv( source = MPI.ANY_SOURCE, tag = MPI.ANY_TAG,
                           status = st )
        src = st.source
        tag = st.tag
        print "Recv from " + str( src ) + " with tag " + str( tag )

if __name__ == "__main__":
    main()
```

2.3 Communications collectives

Les communications collectives suivent la mÃªme logique :

— communication d'objets Python : nom de routine commençant par une minuscule

— communication d'un buffer : nom de routine commençant par une majuscule

À noter que la racine des communications collectives en ayant une est un argument optionnel, par défaut initialisé à 0.

Un petit exemple avec une diffusion : fonction bcast. Attention : la fonction est appelée de la même façon pour tous les processus, mais les processus autres que la racine doivent récupérer le résultat dans ce qui est retourné par l'appel de la fonction, et non pas dans l'argument passé en paramètre.

```
#!/bin/python
```

```
2
3  from mpi4py import MPI
4  import os
5
6  def main():
7      comm = MPI.COMM_WORLD
8      rank = comm.Get_rank()
9      if 0 == rank:
10         token = os.getpid()
11     else:
12         token = None
13     token = comm.bcast( token, root = 0 )
14
15     print str( rank ) + "] Token : " + str( token )
16
17 if __name__ == "__main__":
18     main()
```

De la même façon, avec une réduction, le résultat se trouve dans le buffer retourné par la fonction appelée dans le cas d'un MPI.reduce(). Dans le cas de données à type explicite (`MPI.Reduce()`), le buffer contenant le résultat et le type de données sont passés dans un tuple de la même façon que le buffer d'envoi.

```
1  #!/bin/python
2
3  from mpi4py import MPI
4  import numpy
5  import os
6
7  def main():
8      comm = MPI.COMM_WORLD
9      rank = comm.Get_rank()
10     token = numpy.arange( 10, dtype = 'i' )
11     for i in range( 0, 9 ) :
12         token[i] = os.getpid()
13     maximum = numpy.empty( 10, dtype = 'i' )
14     comm.Reduce( [token, MPI.INT], [maximum, MPI.INT],
15                  op = MPI.MAX, root = 0 )
16
17     print str( rank ) + "] Token : " + str( token[0] )
18     comm.Barrier()
19     if 0 == rank:
20         print str( rank ) + "] Global max : " + str( maximum[0] )
21
22 if __name__ == "__main__":
23     main()
```

2.4 Spawning d'un nouveau processus

Attention, la fonction `MPI_Comm_get_parent()` devient `Get_parent()` de l'objet communicateur.

```
1  #!/bin/python
2
3  from mpi4py import MPI
4  import numpy
5  import os
```

```
6  import sys
7
8  def main():
9      comm = MPI.COMM_WORLD
10     rank = comm.Get_rank()
11     interpreteur = sys.executable
12     script = os.path.basename(sys.argv[0])
13
14     parentcomm = comm.Get_parent()
15
16     if parentcomm == MPI.COMM_NULL:
17         print str( rank ) + "] spawning a new process"
18         comm.Spawn( interpreteur, args=[script] )
19     else:
20         print "I am the newly spawned process"
21
22 if __name__ == "__main__":
23     main()
```

3 Gestion des exceptions

Les bibliothèques MPI retournent des codes d'erreur spécifiques lorsque quelque chose se passe mal. Par exemple, OpenMPI (qui est l'implémentation MPI disponible sur Magi) retourne `MPI_ERR_TRUNCATE` lorsque le message reçu est tronqué, etc. En Python, ces erreurs sont remontées en soulevant des exceptions.

Par exemple, si le rang du destinataire d'un message est erroné, la fonction d'envoi `MPI_Send()` d'OpenMPI retourne le code d'erreur `MPI_ERR_RANK`. La fonction Python send() soulève alors une exception, et on regarde dans cette exception pour en savoir plus.

```
1  #!/bin/python
2
3  from mpi4py import MPI
4
5  def main():
6      comm = MPI.COMM_WORLD
7      rank = comm.Get_rank()
8      token = rank
9
10     if 0 == rank:
11         try:
12             # On essaye d'envoyer a un rang de processus qui n'existe pas
13             comm.send( token, dest = MPI.ANY_SOURCE, tag = 40 )
14         except MPI.Exception, ierr:
15             print "exception soulevee"
16             print ierr.Get_error_string()
17
18 if __name__ == "__main__":
19     main()
```

Le code ci-dessus donne l'affichage suivant :

```
1  exception soulevee
2  MPI_ERR_RANK: invalid rank
```

Plus d'information sur les codes d'erreur d'Open MPI : `http://www.netlib.org/utk/papers/mpi-book/node178.html`.