MPI for Python

In-Ho Lee
(이인호)
Korea Research Institute of Standards and Science, Daejeon 305-340
(한국표준과학연구원)

ihlee@kriss.re.kr
http://incredible.egloos.com

KIAS, June 25-27, 2014

contents

Brief introduction to Python

MPI basics

MPI practice

mpi4py

MPI4Py provides an interface very similar to the MPI-2 standard C++ Interface

Focus is in translating MPI syntax and semantics: If you know MPI, MPI4Py is "obvious"

You can communicate Python objects!!

What you lose in performance, you gain in shorter development time

There are hundreds of functions in the MPI standard, not all of them are necessarily available in MPI4Py, most commonly used are

No need to call MPI_Init() or MPI_Finalize()

–MPI_Init() is called when you import the module

–MPI_Finalize() is called before the Python process ends

Python+????

- petsc4py
- mpi4py
- numpy

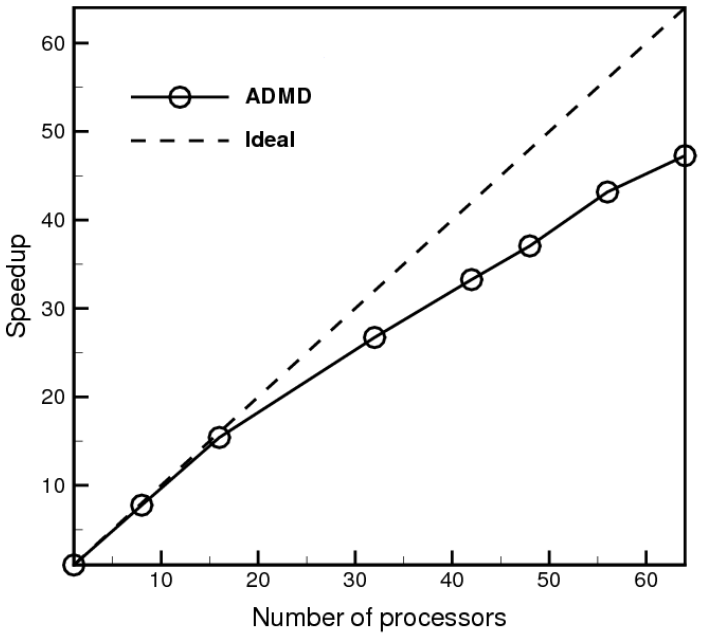
speedup

$$S_p = \frac{T_1}{T_p}$$

***p* is the number of processors.**

***T*$_p$ is the execution time of the algorithm with *p* processors.**

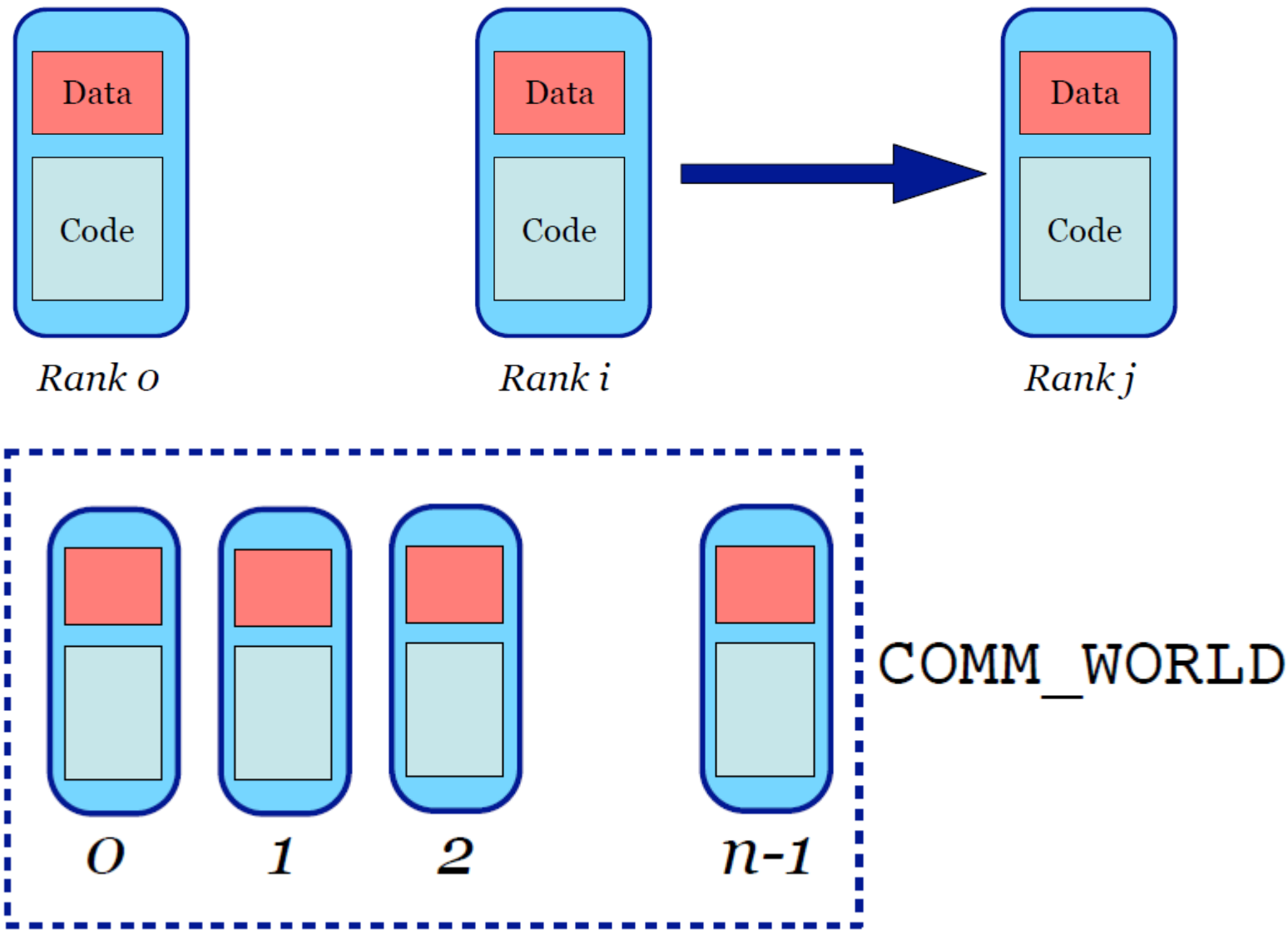
Not CPU time, but wall-clock time reference



http://en.wikipedia.org/wiki/Speedup

Embarrassingly parallel

- little or no communication of results between tasks


Data
Code
Rank o
Data
Code
Rank i
Data
Code
Rank j
0
1
2
n-1
COMM_WORLD

Python program with MPI

- All processors execute the same program
- Every process has a unique rank
- Branch on the rank
- Point-to-point, collective communications
- Blocking, nonblocking communications

Mpi4py INSTALL

https://bitbucket.org/mpi4py/mpi4py/downloads

- $ tar -zxf mpi4py-1.3.1.tar.gz
- $ cd mpi4py-1.3.1
- $ python setup.py build
- $ python setup.py install

- from mpi4py import MPI

Installation

```
$ python
Python 2.6.6 (r266:84292, Jul 10 2013, 22:48:45)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from mpi4py import MPI
>>> quit
Use quit() or Ctrl-D (i.e. EOF) to exit
>>>
[ihlee@tucana-master ~]$
```

mpi4py-1.3.1

```
[ihlee@tucana-master mpi4py_tuto]$ ls /usr/local/mpi4py-1.3.1
total 80
4 build/    4 demo/    8 HISTORY.txt    8 mpi.cfg     4 README.txt   20 setup.py    4 test/
4 conf/     4 docs/    4 LICENSE.txt    4 PKG-INFO    4 setup.cfg     4 src/         4 THANKS.txt
[ihlee@tucana-master mpi4py_tuto]$ ls /usr/local/mpi4py-1.3.1/demo/
total 116
4 compute-pi/      4 helloworld.f90  4 mpe-logging/      4 osu_bw.py          4 sequential/    4 wrap-c/
4 cython/          4 helloworld.py    4 mpi-ref-v1/       4 osu_latency.py     4 spawning/      4 wrap-cython/
4 embedding/       4 init-fini/       4 nxtval/           4 osu_multi_lat.py   4 threads/       4 wrap-f2py/
4 helloworld.c     4 makefile         4 osu_alltoall.py   4 README.txt         4 vampirtrace/   4 wrap-swig/
4 helloworld.cxx   4 mandelbrot/      4 osu_bibw.py       4 reductions/        4 wrap-boost/
[ihlee@tucana-master mpi4py_tuto]$
```

Hello World

```
[ihlee@tucana-master mpi4py_tuto]$ cat /usr/local/mpi4py-1.3.1/demo/helloworld.py
#!/usr/bin/env python
"""
Parallel Hello World
"""

from mpi4py import MPI
import sys

size = MPI.COMM_WORLD.Get_size()
rank = MPI.COMM_WORLD.Get_rank()
name = MPI.Get_processor_name()

sys.stdout.write(
    "Hello, World! I am process %d of %d on %s.₩n"
    % (rank, size, name))
[ihlee@tucana-master mpi4py_tuto]$
```

test

[ihlee@tucana-master mpi4py_tuto]$ cp /usr/local/mpi4py-1.3.1/demo/helloworld.py .

```
[ihlee@tucana-master mpi4py_tuto]$ mpirun -np 5 python helloworld.py
Hello, World! I am process 2 of 5 on master.hpc.
Hello, World! I am process 3 of 5 on master.hpc.
Hello, World! I am process 4 of 5 on master.hpc.
Hello, World! I am process 0 of 5 on master.hpc.
Hello, World! I am process 1 of 5 on master.hpc.
[ihlee@tucana-master mpi4py_tuto]$
```

Issuing at the command line:

```
$ mpiexec -n 5 python demo/helloworld.py
```

or (in the case of older MPI-1 implementations):

```
$ mpirun -np 5 python demo/helloworld.py
```

- Communication of Python objects.
  - high level and very convenient, based in `pickle` serialization
  - can be slow for large data (CPU and memory consuming)

```
<object> ⟶ pickle.dump() ⟶ send()
                                  ↓
<object> ⟵ pickle.load() ⟵ recv()
```

- Communication of array data (e.g. **NumPy** arrays).
  - lower level, slightly more verbose
  - very fast, almost C speed (for messages above 5-10 KB)

```
message = [<object>, (count, displ), datatype]
```

- Broadcasting a Python dictionary:

```
from mpi4py import MPI

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

if rank == 0:
    data = {'key1' : [7, 2.72, 2+3j],
            'key2' : ( 'abc', 'xyz')}
else:
    data = None
data = comm.bcast(data, root=0)
```

- Scattering Python objects:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

if rank == 0:
   data = [(i+1)**2 for i in range(size)]
else:
   data = None
data = comm.scatter(data, root=0)
assert data == (rank+1)**2
```

- Gathering Python objects:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Get_rank()

data = (rank+1)**2
data = comm.gather(data, root=0)
if rank == 0:
   for i in range(size):
      assert data[i] == (i+1)**2
else:
   assert data is None
```

Importing library:

```
from mpi4py import MPI
```

Getting important information:

```
comm = MPI.COMM_WORLD
rank = MPI.COMM_WORLD.Get_rank()
size = MPI.COMM_WORLD.Get_size()
name = MPI.Get_processor_name()
```

Executing in parallel:

```
mpirun -np <P> python <code>.py
```

```
import time
t = time.time()
time.sleep(10)
t2 = time.time()

spendtime = t2 - t
print("Before timestemp: ", t)
print("After timestemp: ", t2)
print("Wait {0} seconds".format(spendtime))
```

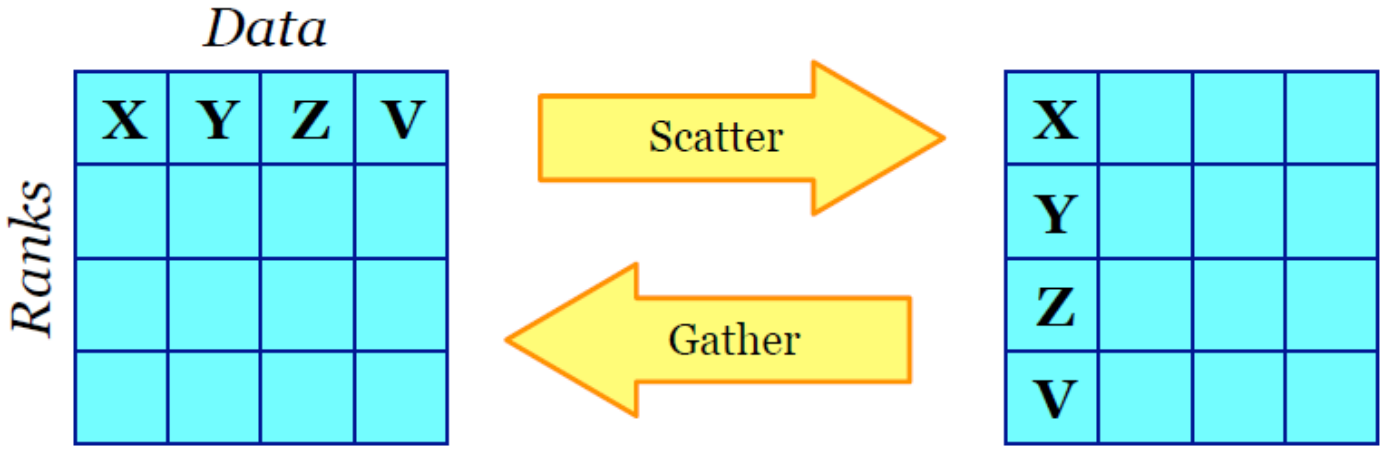
[실행결과]

```
>>>
('Before timestemp: ', 1321780317.218)
('After timestemp: ', 1321780327.218)
Wait 10.0 seconds
```

```
[ihlee@tucana-master ~]$ python
Python 2.6.6 (r266:84292, Jul 10 2013, 22:48:45)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import time
>>> t1=time.time()
>>> t2=time.time()
>>> print t2-t1
7.2504940033
>>>
```

Scatter/Gather

./s_g.py
After Scatter:
[0] [0. 1. 2. 3.]
After Allgather:
[0] [0. 2. 4. 6.]
[ihlee@tucana-master mpi4py_tuto]$ vi s_g.py



```
#!/usr/bin/env python
#--------------
# Loaded Modules
#--------------
import numpy as np
from mpi4py import MPI
#-------------
# Communicator
#-------------
comm = MPI.COMM_WORLD

my_N = 4
N = my_N * comm.size

if comm.rank == 0:
   A = np.arange(N, dtype=np.float64)
else:
   #Note that if I am not the root processor A is an empty array
   A = np.empty(N, dtype=np.float64)
my_A = np.empty(my_N, dtype=np.float64)
#-------------------------
# Scatter data into my_A arrays
#-------------------------
comm.Scatter( [A, MPI.DOUBLE], [my_A, MPI.DOUBLE] )

if comm.rank == 0:
  print "After Scatter:"

for r in xrange(comm.size):
   if comm.rank == r:
      print "[%d] %s" % (comm.rank, my_A)
   comm.Barrier()
```

```
#------------------------
# Everybody is multiplying by 2
#------------------------
my_A *= 2

#----------------------
# Allgather data into A again
#----------------------
comm.Allgather( [my_A, MPI.DOUBLE], [A, MPI.DOUBLE] )

if comm.rank == 0:
   print "After Allgather:"

for r in xrange(comm.size):
    if comm.rank == r:
        print "[%d] %s" % (comm.rank, A)
    comm.Barrier()
```

```
from mpi4py import MPI
import numpy as np

COMM = MPI.COMM_WORLD
RANK = COMM.Get_rank()
SIZE = COMM.Get_size()
N = 10

if(RANK == 0):
    DATA = np.arange(N*SIZE, dtype='i')
    for i in range(1, SIZE):
        SLICE = DATA[i*N:(i+1)*N]
        COMM.Send([SLICE, MPI.INT], dest=i)
    MYDATA = DATA[0:N]
else:
    MYDATA = np.empty(N, dtype='i')
    COMM.Recv([MYDATA, MPI.INT], source=0)
```

```
S = sum(MYDATA)
print RANK, 'has data', MYDATA, 'sum =', S

SUMS = np.zeros(SIZE, dtype='i')
if(RANK > 0):
    COMM.send(S, dest=0)
else:
    SUMS[0] = S
    for i in range(1, SIZE):
        SUMS[i] = COMM.recv(source=i)
    print 'total sum =', sum(SUMS)
```

```
from mpi4py import MPI
comm = MPI.COMM_WORLD

if comm.rank == 0:
    sendmsg = (7, "abc", [1.0,2+3j], {3:4})
else:
    sendmsg = None

recvmsg = comm.bcast(sendmsg, root=0)
```

```
from mpi4py import MPI
comm = MPI.COMM_WORLD

if comm.rank == 0:
    sendmsg = [i**2 for i in range(comm.size)]
else:
    sendmsg = None

recvmsg = comm.scatter(sendmsg, root=0)
```

```
from mpi4py import MPI
comm = MPI.COMM_WORLD

sendmsg = comm.rank**2

recvmsg1 = comm.gather(sendmsg, root=0)

recvmsg2 = comm.allgather(sendmsg)
```

```
from mpi4py import MPI
comm = MPI.COMM_WORLD

sendmsg = comm.rank

recvmsg1 = comm.reduce(sendmsg, op=MPI.SUM, root=0)

recvmsg2 = comm.allreduce(sendmsg)
```

conclusions

- Many examples
- High-level usage of MPI
- Practice