mpi4py

HPC Python

Antonio Gómez-Iglesias
agomez@tacc.utexas.edu

April 17, 2015

What is MPI

- Message Passing Interface
- Most useful on distributed memory machines
- Many implementations, interfaces in C/C++/Fortran
- Why Python?
  - Easy!
  - Great for prototyping
  - Small to medium codes

Can I use it for production?

Yes, if the communication is not very frequent and performance is not the primary concern

MPI

- An MPI Program is launched as separate processes (tasks)
- Each task has its own address space
  - Requires partitioning data across tasks: if you don't do anything, you just run the same thing **N** times
  - Data is explicitly moved from task to task (message passing)
  - Two classes of message passing
    - Point-to-Point messages involving only two tasks
    - Collective messages involving a set of tasks

Why MPI

Universality

- Works on separate processors connected by any network (and even on shared memory systems)
- Matches the hardware of most of today's parallel supercomputers

Performance/Scalability

- Scalability is the most compelling reason why message passing will remain a permanent component of HPC
- As modern systems increase core counts, management of the memory hierarchy (including distributed memory) is the key to extracting the highest performance
- Each message passing process only directly uses its local data $\rightarrow$ compilers and cache management hardware function without contention

mpi4py

- Great implementation of MPI on Python (there are others)
- mpi4py provides an interface very similar to the MPI Standard C++ Interface
- If you know MPI, ~~mpi4py is easy~~
- You can communicate Python objects
- What you lose in performance, you gain in shorter development time

Functionality

- There are hundreds of functions in the MPI standard: you don't need to know all of them
- Important particularity of mpi4py: no need to call `MPI_Init()` or `MPI_Finalize()`

To launch

- On Stampede, using python to run mpi4py is not recommended
- Use this:

```
ibrun python-mpi <my_mpi4py_python_script>
```

Where to find python-mpi

```
$TACC_PYTHON_LIB/python2.7/site-packages/mpi4py/bin/python-mpi
```

- Include this in your script:

```
export PATH=$PATH:$TACC_PYTHON_LIB/python2.7/site-packages/mpi4py/bin
```

Communicators

Predefined Instances

- `COMM_WORLD`: all the processes involved
- `COMM_SELF`: contains just the calling process
- `COMM_NULL`: no communicator

```
comm = MPI.COMM_WORLD
```

Information

```
rank = comm.Get_rank()
size = comm.Get_size()
```

A First Example

Hello | examples/4_mpi4py/hello.py

```
1 from mpi4py import MPI
2
3 comm = MPI.COMM_WORLD
4
5 print "Hello! I'm rank %02d from %02d" % (comm.rank, comm.size)
6
7 print "Hello! I'm rank %02d from %02d" % (comm.Get_rank(),
      comm.Get_size())
8
9 print "Hello! I'm rank %02d from %02d" %
      (MPI.COMM_WORLD.Get_rank(), MPI.COMM_WORLD.Get_size())
```

```
> ibrun python-mpi hello.py
```

Data Communication

- Python objects can be communicated with the send and receive methods of the communicator

`send(data, dest, tag)`

- data: Python object to send
- dest: destination rank
- tag: id given to the message

`data = receive(source, tag)`

- source: source rank
- tag: id given to the message
- data is provided as return value

- Destination and source ranks as well as tags have to match

Point to Point

examples/4_mpi4py/p2p.py

```
1  from mpi4py import MPI
2
3  comm = MPI.COMM_WORLD
4  assert comm.size == 2
5
6  if comm.rank == 0:
7      sendmsg = 123
8      comm.send(sendmsg, dest=1, tag=11)
9      recvmsg = comm.recv(source=1, tag=22)
10     print "[%02d] Received message: %s" % (comm.rank, recvmsg)
11 else:
12     recvmsg = comm.recv(source=0, tag=11)
13     print "[%02d] Received message: %d" % (comm.rank, recvmsg)
14     sendmsg = "Message from 1"
15     comm.send(sendmsg, dest=0, tag=22)
```

```
> ibrun -np 2 python-mpi p2p.py
```

Under the Hood

- Python objects are converted to byte streams (send)
- The byte stream is converted back to Python object (receive)
- This conversion (serialization) introduces an overhead!
- NumPy arrays are communicated with very little overhead. But only with upper case methods:
  - `Send(data, dest, tag)`
  - `Recv(data, source, tag)`
- When receiving the data array has to exist in the time of call

Remember upper/lower case

- `send/recv`: general Python objects, slow
- `Send/Recv`: continuous arrays, fast

Point to Point with Numpy

examples/4_mpi4py/p2p_numpy.py

```
1  from mpi4py import MPI
2  import numpy
3
4  comm = MPI.COMM_WORLD
5  assert comm.size == 2
6
7  rank = comm.rank
8
9  # pass explicit MPI datatypes
10 if rank == 0:
11     data = numpy.arange(10, dtype='i')
12     comm.Send([data, MPI.INT], dest=1, tag=77)
13 elif rank == 1:
14     data = numpy.empty(10, dtype='i')
15     comm.Recv([data, MPI.INT], source=0, tag=77)
16     print "[%02d] Received: %s" % (rank, data)
17 # automatic MPI datatype discovery
18 if rank == 0:
19     data = numpy.arange(10, dtype=numpy.float64)
20     comm.Send(data, dest=1, tag=13)
21 elif rank == 1:
22     data = numpy.empty(10, dtype=numpy.float64)
23     comm.Recv(data, source=0, tag=13)
24     print "[%02d] Received: %s" % (rank, data)
```

```
> ibrun -np 2 python-mpi p2p_numpy.py
```

Advanced Point to Point

- Tag can be any tag!
- Source can be any source!
  - Use class `Status`
- Communication can be non-blocking:
  - Use class `Request`
  - `Isend/Irecv (isend)`: return immediately
  - `Test/Testany/Testall (test/testany/testall)`: check if one/any/all pending requests finished
  - `Wait/Waitany/Waitall (wait/waitany/waitall)`: wait until one/any/all pending requests finish
  - You can even cancel a request

Advanced Point to Point

Status | examples/4_mpi4py/status.py

```python
from mpi4py import MPI
import numpy

comm = MPI.COMM_WORLD
assert comm.size == 2

rank = comm.rank
status = MPI.Status()

# pass explicit MPI datatypes
if rank == 0:
    data = numpy.arange(1000, dtype='i')
    comm.Send([data, MPI.INT], dest=1, tag=77)
elif rank == 1:
    data = numpy.empty(1000, dtype='i')
    comm.Recv([data, MPI.INT],
          source=MPI.ANY_SOURCE,
          tag=MPI.ANY_TAG, status=status)
    source = status.Get_source()
    tag = status.Get_tag()
    print "[%02d] Received data from source %d
          with tag %d" % (rank, source, tag)
```

Request | examples/4_mpi4py/request.py

```python
if rank==0:
    requests = [MPI.REQUEST_NULL for i in
          range(0,size)]
    d = np.zeros (size, dtype='i')
    print "[%02d] Original data %s " % (rank, d)
    #Request data from a set of processes
    for i in range (1, size):
        requests[i] = comm.Irecv([d[i:],1,
              MPI.INT], i, MPI.ANY_TAG)

    status = [MPI.Status() for i in
          range(0,size)]
    #Wait for all the messages
    MPI.Request.Waitall(requests, status)
    for i in range(1,size):
        source = status[i].source
        tag = status[i].tag
        assert d[i] == source ; assert d[i] ==
              tag
    print "[%02d] Received data %s " % (rank, d)
else:
    data = np.array ([rank])
    time.sleep (np.random.random_sample())
    request = comm.Isend ([data[:], 1, MPI.INT]
          , 0, rank)
    request.Wait()
```

Advanced Point to Point

Asynchronous | examples/4_mpi4py/comm_asynchronous.py

```
1  from mpi4py import MPI
2  import time
3
4  comm = MPI.COMM_WORLD
5  assert comm.size == 2
6
7  rank = comm.rank
8  start = MPI.Wtime()
9  if rank == 0:
10     sendmsg = 123
11     target = 1
12 else:
13     target = 0
14
15 if rank==0:
16     time.sleep(2)
17     request = comm.isend(sendmsg, dest=target, tag=11)
18     request.Wait()
19 else:
20     while not comm.Iprobe(source=target, tag=11):
21         print "[%02d] Waiting for message " % rank
22         time.sleep (0.5)
23     time.sleep(0.5)
24     recvmsg = comm.recv (source=target, tag=11)
25     print "[%02d] Message received %s " % (rank, str(recvmsg))
26 comm.Barrier()
27 end = MPI.Wtime()
28 if rank==0:
29     print "Total time %f" % (end-start)
```

Reduction

Reduction | examples/4_mpi4py/reduction.py

```
1  import numpy as np
2  from mpi4py import MPI
3  comm = MPI.COMM_WORLD
4  size = comm.size
5  rank = comm.rank
6
7  #Give me my start and end index of an array of size N using my rank
8  def partition(rank, size, N):
9      n = N//size + ((N % size) > rank)
10     s = rank * (N//size)
11     if (N % size) > rank:
12         s += rank
13     else:
14         s += N % size
15     return s, s+n
16
17 #Define the size of the problem
18 N = 1000
19 start, end = partition(rank, size, N)
20
21 #Calculate the local sum of all the integers from start to end
22 local_sum = sum(range(start,end))
23 #Get the global sum
24 global_sum = comm.reduce (local_sum, op=MPI.SUM, root=0)
25 if rank == 0:
26     print "[%02d] Received: %d -- Correct: %d" % (rank, global_sum, np.arange(N).sum())
```

RMA

RMA | examples/4_mpi4py/rma.py

```
1  from mpi4py import MPI
2  import numpy as np
3
4  comm = MPI.COMM_WORLD
5  rank = comm.rank
6
7  n = np.zeros (10, dtype=np.int)
8  if rank==0:
9      win = MPI.Win.Create (n, comm=MPI.COMM_WORLD)
10 else:
11     win = MPI.Win.Create (None, comm=MPI.COMM_WORLD)
12
13 if rank==0:
14     print "[%02d] Original data %s" % (rank, n)
15
16 win.Fence()
17 if rank!=0:
18     data = np.arange(10, dtype = np.int)
19     win.Accumulate (data, 0, op=MPI.SUM)
20 win.Fence()
21
22 if rank==0:
23     print "[%02d] Received data %s" % (rank, n)
24
25 win.Free()
```

Collectives

- Multiple processes within the same communicator exchange messages and possibly perform operations
- Always blocking, no tags (organized by calling order)
- Typical operations: `Broadcast`, `Scatter`, `Gather`, `Reduction`

Broadcast

With Numpy | examples/4_mpi4py/bcast.py

```
1  from mpi4py import MPI
2  import numpy
3
4  comm = MPI.COMM_WORLD
5
6  if comm.rank == 0:
7      #rank 0 has data
8      A = numpy.arange(10, dtype=numpy.float64)
9  else:
10     #all other have an empty array
11     A = numpy.empty(10, dtype=numpy.float64)
12
13 # Broadcast from rank 0 to everybody
14 comm.Bcast( [A, MPI.DOUBLE] , root=0)
15
16 print "[%02d] %s" % (comm.rank, A)
```

Dictionary | examples/4_mpi4py/bcast_dict.py

```
1  from mpi4py import MPI
2
3  comm = MPI.COMM_WORLD
4
5  if comm.rank == 0:
6     data = {'key1' : [7, 2.72, 2+3j],
7             'key2' : ( 'abc', 'xyz')}
8  else:
9     data = None
10
11 data = comm.bcast(data, root=0)
12 print "[%02d] %s" % (comm.rank, data)
```

Scatter & Gather

Scatter | examples/4_mpi4py/scatter.py

```
1  from mpi4py import MPI
2
3  comm = MPI.COMM_WORLD
4  size = comm.size
5  rank = comm.rank
6
7  if rank == 0:
8      data = [i for i in range(size)]
9      print "[%02d] Original data: %s" % (rank,
           data)
10 else:
11     data = None
12 data = comm.scatter(data, root=0)
13 assert data == rank
14
15 print "[%02d] Data received: %d" % (rank, data)
```

Gather | examples/4_mpi4py/gather.py

```
1  from mpi4py import MPI
2
3  comm = MPI.COMM_WORLD
4  size = comm.size
5  rank = comm.rank
6  data = (rank+1)**2
7  print "[%02d] Sending value: %d " % (rank, data)
8
9  data = comm.gather(data, root=0)
10 if rank == 0:
11     for i in range(size):
12         assert data[i] == (i+1)**2
13 else:
14     assert data is None
15
16 if rank == 0:
17     print "[%02d] After gather: %s " % (rank,
           data)
```

Split Processes in 2 Communicators

Communicators | examples/4_mpi4py/communicator.py

```
1  from mpi4py import MPI
2
3  world_rank = MPI.COMM_WORLD.rank
4  world_size = MPI.COMM_WORLD.size
5
6  color = world_rank%2
7  if (color==0):
8      key = +world_rank
9  else:
10     key = -world_rank
11
12 newcomm = MPI.COMM_WORLD.Split(color, key)
13
14 newcomm_rank = newcomm.rank
15 newcomm_size = newcomm.size
16 for i in range(world_size):
17     MPI.COMM_WORLD.Barrier()
18     if (world_rank==i):
19         print "Global: rank %d of %d. New comm: rank %d of %d" % (world_rank, world_size,
               newcomm_rank, newcomm_size)
20
21 newcomm.Free()
```

License