mpi4py

HPC Python

Antonio Gómez-Iglesias
agomez@tacc.utexas.edu

October 30th, 2014

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, avoiding complexities of process-shared data, and allowing compilers and cache management hardware to 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

```
ibrun python-mpi <my mpi4py python script>
```

python-mpi

```
/opt/apps/intel14/mvapich2_2_0/python/2.7.6/lib/python2.7/site-packages/mpi4py/bin/python-mpi

export
PATH=$PATH:/opt/apps/intel14/mvapich2_2_0/python/2.7.6/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

```
from mpi4py import MPI

comm = MPI.COMM_WORLD

print "Hello! I'm rank %02d from %02d" % (comm.rank, comm.size)

print "Hello! I'm rank %02d from %02d" % (comm.Get_rank(), comm.Get_size())

print "Hello! I'm rank %02d from %02d" % (MPI.COMM_WORLD.Get_rank(), MPI.COMM_WORLD.Get_size())
```

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

```
from mpi4py import MPI

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

if comm.rank == 0:
    sendmsg = 123
    comm.send(sendmsg, dest=1, tag=11)
    recvmsg = comm.recv(source=1, tag=22)
    print "[%02d] Received message: %s" % (comm.rank, recvmsg)
else:
    recvmsg = comm.recv(source=0, tag=11)
    print "[%02d] Received message: %d" % (comm.rank, recvmsg)
    sendmsg = "Message from 1"
    comm.send(sendmsg, dest=0, tag=22)
```

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

```python
from mpi4py import MPI
import numpy

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

rank = comm.rank

# pass explicit MPI datatypes
if rank == 0:
    data = numpy.arange(10, dtype='i')
    comm.Send([data, MPI.INT], dest=1, tag=77)
elif rank == 1:
    data = numpy.empty(10, dtype='i')
    comm.Recv([data, MPI.INT], source=0, tag=77)
    print "[%02d] Received: %s" % (rank, data)
# automatic MPI datatype discovery
if rank == 0:
    data = numpy.arange(10, dtype=numpy.float64)
    comm.Send(data, dest=1, tag=13)
elif rank == 1:
    data = numpy.empty(10, dtype=numpy.float64)
    comm.Recv(data, source=0, tag=13)
    print "[%02d] Received: %s" % (rank, data)
```

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

```python
from mpi4py import MPI
import time

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

rank = comm.rank
start = MPI.Wtime()
if rank == 0:
    sendmsg = 123
    target = 1
else:
    target = 0

if rank==0:
    time.sleep(2)
    request = comm.isend(sendmsg, dest=target, tag=11)
    request.Wait()
else:
    while not comm.Iprobe(source=target, tag=11):
        print "[%02d] Waiting for message " % rank
        time.sleep (0.5)
    time.sleep(0.5)
    recvmsg = comm.recv (source=target, tag=11)
    print "[%02d] Message received %s " % (rank, str(recvmsg))
comm.Barrier()
end = MPI.Wtime()
if rank==0:
    print "Total time %f" % (end-start)
```

Reduction

Reduction | examples/4_mpi4py/reduction.py

```python
import numpy as np
from mpi4py import MPI
comm = MPI.COMM_WORLD
size = comm.size
rank = comm.rank

#Give me my start and end index of an array of size N using my rank
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

#Define the size of the problem
N = 1000
start, end = partition(rank, size, N)

#Calculate the local sum of all the integers from start to end
local_sum = sum(range(start,end))
#Get the global sum
global_sum = comm.reduce (local_sum, op=MPI.SUM, root=0)
if rank == 0:
    print "[%02d] Received: %d -- Correct: %d" % (rank, global_sum, np.arange(N).sum())
```

RMA

RMA | examples/4_mpi4py/rma.py

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.rank

n = np.zeros (10, dtype=np.int)
if rank==0:
    win = MPI.Win.Create (n, comm=MPI.COMM_WORLD)
else:
    win = MPI.Win.Create (None, comm=MPI.COMM_WORLD)

if rank==0:
    print "[%02d] Original data %s" % (rank, n)

win.Fence()
if rank!=0:
    data = np.arange(10, dtype = np.int)
    win.Accumulate (data, 0, op=MPI.SUM)
win.Fence()

if rank==0:
    print "[%02d] Received data %s" % (rank, n)

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

```python
from mpi4py import MPI
import numpy

comm = MPI.COMM_WORLD

if comm.rank == 0:
    #rank 0 has data
    A = numpy.arange(10, dtype=numpy.float64)
else:
    #all other have an empty array
    A = numpy.empty(10, dtype=numpy.float64)

# Broadcast from rank 0 to everybody
comm.Bcast( [A, MPI.DOUBLE] , root=0)

print "[%02d] %s" % (comm.rank, A)
```

Dictionary | examples/4_mpi4py/bcast_dict.py

```python
from mpi4py import MPI

comm = MPI.COMM_WORLD

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

data = comm.bcast(data, root=0)
print "[%02d] %s" % (comm.rank, data)
```

Scatter & Gather

Scatter | examples/4_mpi4py/scatter.py

```python
from mpi4py import MPI

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

if rank == 0:
    data = [i for i in range(size)]
    print "[%02d] Original data: %s" % (rank,
            data)
else:
    data = None
data = comm.scatter(data, root=0)
assert data == rank

print "[%02d] Data received: %d" % (rank, data)
```

Gather | examples/4_mpi4py/gather.py

```python
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.size
rank = comm.rank
data = (rank+1)**2
print "[%02d] Sending value: %d " % (rank, data)

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

if rank == 0:
    print "[%02d] After gather: %s " % (rank,
            data)
```

Split Processes in 2 Communicators

Communicators | examples/4_mpi4py/communicator.py

```python
from mpi4py import MPI

world_rank = MPI.COMM_WORLD.rank
world_size = MPI.COMM_WORLD.size

color = world_rank%2
if (color==0):
    key = +world_rank
else:
    key = -world_rank

newcomm = MPI.COMM_WORLD.Split(color, key)

newcomm_rank = newcomm.rank
newcomm_size = newcomm.size
for i in range(world_size):
    MPI.COMM_WORLD.Barrier()
    if (world_rank==i):
        print "Global: rank %d of %d. New comm: rank %d of %d" % (world_rank, world_size,
               newcomm_rank, newcomm_size)

newcomm.Free()
```

License