03_Scaling_Python

September 26, 2014

1 Scaling Python

2 Python in HPC Tutorial

2.1 Supercomputing 2014

Presenter: Matthew G. Knepley

Contributors: - Aron Ahmadia - Andy R. Terrel - Matthew G. Knepley

Matt Knepley and Aron Ahmadia

2.2 Acknowledgements

- mpi4py is a Cythonized wrapper around MPI originally developed by Lisandro Dalcin, CONICET

2.3 Slideshow Setup Code

```
In [1]: %matplotlib inline
```

2.4 Connecting to MPI via IPython

To run the examples in parallel, you must run the command:

`ipcluster start --engines=MPI --n 4`

Or use notebook Appendix_03_Launch_MPI_Engines (if you are using the VMs or have this configured for your environment).

Then execute the next cell:

```
In [2]: from IPython.parallel import Client
        c = Client()
        view = c[:]

        %load_ext parallelmagic
        view.activate()
        view.block = True
```

2.5 The Message Passing Model

- a process is (traditionally) a program counter for instructions and an address space for data
- processes may have multiple threads (program counters and associated stacks) sharing a single address space
- message passing is for communication among processes, which have separate address spaces
- interprocess communication consists of
- synchronization
- movement of data from one process's address space to another's

---

2.6 Why MPI?

- **communicators** encapsulate communication spaces for library safety
- **datatypes** reduce copying costs and permit heterogeneity
- multiple **communication modes** allow more control of memory buffer management
- extensive **collective operations** for scalable global communication
- **process topologies** permit efficient process placement, user views of process layout
- **profiling interface** encourages portable tools

It Scales!

2.7 MPI - Quick Review

- processes can be collected into **groups**
- each message is sent in a **context**, and must be received in the same context
- a **communicator** encapsulates a context for a specific group
- a given program may have many communicators with any level of overlap
- two initial communicators
- `MPI_COMM_WORLD` (all processes)
- `MPI_COMM_SELF` (current process)

First Example: Hello World

Ths user will have to install mpi4py

```
pip install mpi4py
```

It is crucial here that the proper `mpiexec` for your MPI installation be in the path.
For interactive convenience, we load the parallel magic extensions and make this view the active one for the automatic parallelism magics.
This is not necessary and in production codes likely won't be used, as the engines will load their own MPI codes separately. But it makes it easy to illustrate everything from within a single notebook here.

```
In []: from IPython.parallel import Client
       c = Client()
       view = c[:]

       %load_ext parallelmagic
       view.activate()
       view.block = True
```

Use the autopx magic to make the rest of these cell execute on the engines instead of locally

```
In [3]: %autopx
```

```
%autopx enabled
```

With autopx enabled, the next cell will actually execute *entirely on each engine*:

2.8 Hello World

```
In [4]: from mpi4py import MPI

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

        print("Hello World! I am process %d of %d on %s.\n" % (rank, size, name))
```

```
[stdout:0]
Hello World! I am process 0 of 4 on MATTHEW-KNEPLEYs-MacBook-Air-2.local.

[stdout:1]
Hello World! I am process 3 of 4 on MATTHEW-KNEPLEYs-MacBook-Air-2.local.

[stdout:2]
Hello World! I am process 1 of 4 on MATTHEW-KNEPLEYs-MacBook-Air-2.local.

[stdout:3]
Hello World! I am process 2 of 4 on MATTHEW-KNEPLEYs-MacBook-Air-2.local.
```

2.9 Functionality

- 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
- To launch: mpiexec -n < number of process > -machinefile < hostlist > python < my MPI4Py python script >
- IPython automatically handles calling mpiexec for you with the `ipcluster` command

2.10 MPI Basic (Blocking) Send

```
int MPI_Send(void* buf, int count, MPI_Datatype type,
int dest, int tag, MPI_Comm comm)
```

2.10.1 mpi4py

```
Comm.Send(self, buf, int dest=0, int tag=0)
Comm.send(self, obj=None, int dest=0, int tag=0)
```

2.11 MPI Basic (Blocking) Recv

```
int MPI_Recv(void* buf, int count, MPI_Datatype type,
int source, int tag, MPI_Comm comm, MPI_Status status)
```

2.11.1 mpi4py

```
comm.Recv(self, buf, source=0, tag=0, status=None)
comm.recv(self, obj=None, source=0, tag=0, status=None)
```

2.12 Send/Receive Example (lowercase convenience methods)

```
In [5]: from mpi4py import MPI

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

        if rank == 0:
           data = {'a': 7, 'b': 3.14}
           comm.send(data, dest=1, tag=11)
        elif rank == 1:
           data = comm.recv(source=0, tag=11)
           print data
```

```
[stdout:2] {'a': 7, 'b': 3.14}
```

Send/Receive Example (MPI API on numpy)

```
In [6]: from mpi4py import MPI
        import numpy

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

        # 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=0, tag=77)

        # or take advantage of 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 data
```

```
[stdout:2] [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
```

2.13 Synchronization

```
int MPI_Barrier(MPI_Comm comm)
```

2.13.1 mpi4py

```
comm.Barrier(self)
comm.barrier(self)
```

```
In [7]: from mpi4py import MPI

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

        for r_id in range(comm.Get_size()):
            if rank == r_id:
                print "Hello from proc:", rank
            comm.Barrier()
```

```
[stdout:0] Hello from proc: 0
[stdout:1] Hello from proc: 3
[stdout:2] Hello from proc: 1
[stdout:3] Hello from proc: 2
```

Notice that MPI rank does not coincide with IPython rank

2.14 Timing and Profiling

the elapsed (wall-clock) time between two points in an MPI program can be computed using MPI_Wtime:

```
In [8]: t1 = MPI.Wtime()
        t2 = MPI.Wtime()
        print("time elapsed is: %e\n" % (t2-t1))
```

```
[stdout:0]
time elapsed is: 7.049413e-05

[stdout:1]
time elapsed is: 7.011974e-05

[stdout:2]
time elapsed is: 7.037306e-05

[stdout:3]
time elapsed is: 6.986130e-05
```

2.15 Broadcast Example

```
int MPI_Bcast(void *buf, int count, MPI_Datatype type,
int root, MPI_Comm comm)
```

2.15.1 mpi4py

```
comm.Bcast(self, buf, root=0)
comm.bcast(self, obj=None, root=0)
```

```
In [9]: 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)
        print "bcast finished and data \
         on rank %d is: \n" % comm.rank, data
```

```
[stdout:0]
bcast finished and data  on rank 0 is:
{'key2': ('abc', 'xyz'), 'key1': [7, 2.72, (2+3j)]}
[stdout:1]
bcast finished and data  on rank 3 is:
{'key2': ('abc', 'xyz'), 'key1': [7, 2.72, (2+3j)]}
[stdout:2]
bcast finished and data  on rank 1 is:
{'key2': ('abc', 'xyz'), 'key1': [7, 2.72, (2+3j)]}
[stdout:3]
bcast finished and data  on rank 2 is:
{'key2': ('abc', 'xyz'), 'key1': [7, 2.72, (2+3j)]}
```

2.16 Scatter Example

```
In [10]: 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
         print "data on rank %d is: "%comm.rank, data
```

```
[stdout:0] data on rank 0 is:  1
[stdout:1] data on rank 3 is:  16
[stdout:2] data on rank 1 is:  4
[stdout:3] data on rank 2 is:  9
```

2.17 Gather (and Barrier) Example

```
In [11]: from mpi4py import MPI

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

         data = (rank+1)**2
         print "before gather, data on \
           rank %d is: "%rank, data

         comm.Barrier()
         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
         print "data on rank: %d is: "%rank, data
```

```
[stdout:0]
before gather, data on   rank 0 is:  1
data on rank: 0 is:  [1, 4, 9, 16]
[stdout:1]
before gather, data on   rank 3 is:  16
data on rank: 3 is:  None
[stdout:2]
before gather, data on   rank 1 is:  4
data on rank: 1 is:  None
[stdout:3]
before gather, data on   rank 2 is:  9
data on rank: 2 is:  None
```

2.18 Collective Examples

2.19 Reduce Example

```
In [12]: from mpi4py import MPI
         comm = MPI.COMM_WORLD

         sendmsg = comm.rank
         recvmsg1 = comm.reduce(sendmsg, op=MPI.SUM, root=0)
         recvmsg2 = comm.allreduce(sendmsg)

         print recvmsg2
```

```
[stdout:0] 6
[stdout:1] 6
[stdout:2] 6
[stdout:3] 6
```

Compute Pi Example

In [13]:
```python
from mpi4py import MPI
import math

def compute_pi(n, start=0, step=1):
    h = 1.0 / n
    s = 0.0
    for i in range(start, n, step):
        x = h * (i + 0.5)
        s += 4.0 / (1.0 + x**2)
    return s * h

comm = MPI.COMM_WORLD
nprocs = comm.Get_size()
myrank = comm.Get_rank()
if myrank == 0:
    n = 10
else:
    n = None

n = comm.bcast(n, root=0)

mypi = compute_pi(n, myrank, nprocs)

pi = comm.reduce(mypi, op=MPI.SUM, root=0)

if myrank == 0:
    error = abs(pi - math.pi)
    print ("pi is approximately %.16f, error is %.16f" % (pi, error))
```

```
[stdout:0] pi is approximately 3.1424259850010983, error is 0.0008333314113051
```

Mandelbrot Set Example

In [14]:
```python
def mandelbrot (x, y, maxit):
    c = x + y*1j
    z = 0 + 0j
    it = 0
    while abs(z) < 2 and it < maxit:
        z = z**2 + c
        it += 1
    return it

x1, x2 = -2.0, 1.0
y1, y2 = -1.0, 1.0
w, h = 150, 100
maxit = 127

from mpi4py import MPI
import numpy

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

```
    # number of rows to compute here
    N = h // size + (h % size > rank)

    # first row to compute here
    start = comm.scan(N)-N

    # array to store local result
    Cl = numpy.zeros([N, w], dtype='i')

    # compute owned rows

    dx = (x2 - x1) / w
    dy = (y2 - y1) / h

    for i in range(N):
        y = y1 + (i + start) * dy
        for j in range(w):
            x = x1 + j * dx
            Cl[i, j] = mandelbrot(x, y, maxit)

    # gather results at root (process 0)
    counts = comm.gather(N, root=0)
    C = None
    if rank == 0:
        C = numpy.zeros([h, w], dtype='i')

    rowtype = MPI.INT.Create_contiguous(w)
    rowtype.Commit()

    comm.Gatherv(sendbuf=[Cl, MPI.INT], recvbuf=[C, (counts, None), rowtype],root=0)

    rowtype.Free()
```

We now need to "pull" the C array for plotting so we disable autopx. Make sure to re-enable it later on

```
In [18]: %autopx
```

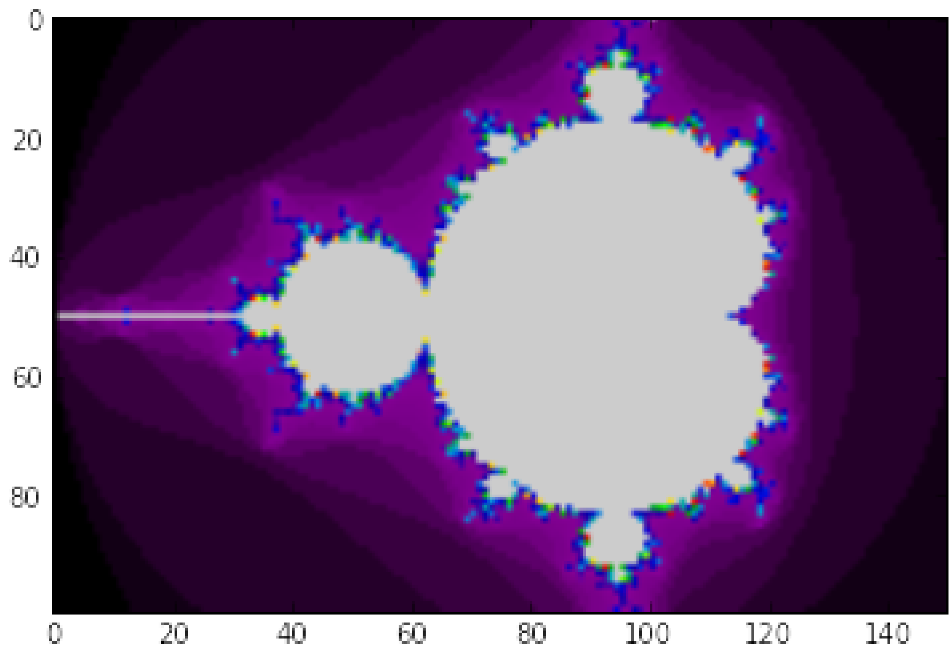
```
%autopx disabled
```

```
In [19]: view['rank']
```

```
Out[19]: [0, 3, 1, 2]
```

```
In [16]: # CC is an array of C from all ranks, so we use CC[0]
         CC = view['C']
         ranks = view['rank']

         # Do the plotting
         from matplotlib import pyplot
         # Some magic to get to MPI4PY rank 0, not necessarily engine rank 0
         pyplot.imshow(CC[ranks.index(0)], aspect='equal')
         pyplot.spectral()
         pyplot.show()
```



Toggle autopx back

```
In [17]: %autopx
```

```
%autopx enabled
```

Advanced Capabilities

Other features supported by mpi4py

- dynamic process spawning: Spawn(), Connect() and Disconnect()
- one sided communication: Put(), Get(), Accumulate()
- MPI-IO: Open(), Close(), Get_view() and Set_view()

More Comprehensive mpi4py Tutorials

* basics - http://mpi4py.scipy.org/docs/usrman/tutorial.html
* advanced - http://www.bu.edu/pasi/files/2011/01/Lisandro-Dalcin-mpi4py.pdf

Interesting Scalable Applications and Tools

- PyTrilinos - http://trilinos.sandia.gov/packages/pytrilinos/

- petsc4py - http://code.google.com/p/petsc4py/
- PyClaw - http://numerics.kaust.edu.sa/pyclaw/
- GPAW - https://wiki.fysik.dtu.dk/gpaw/