MPI4PY Python
Python in HPC
TACC Training, Oct. 15, 2012

Presenters:
Andy R. Terrel, PhD
Texas Advanced Computing Center
University of Texas at Austin

Yaakoub El Khamra
Texas Advanced Computing Center
University of Texas at Austin

0.1 Interacting with the Tutorial Slides

This tutorial is an interactive worksheet designed to encourage you to try out the lessons during the demonstration. If you are looking at the pdf version, we encourage you to download the updated version (see previous slide) and try the interactive version.

To run the interactive version, you need a good Python environment including:

- IPython version >= 13.0
- Numpy version >= 1.5
- Scipy
- Matplotlib

Move to the directory containing the tarball and execute:

```
$ ipython notebook --pylab=inline
```

We heartily endorse the Anaconda distribution and the Free Enthought Python Distribution.

0.2 Presentation mode

The slide show mode is only supported by an IPython development branch version. To get it I recommend cloning from the official branch, adding Matthias Carreau's remote, fetching and using his branch slideshow_extension2. Here are the commands:

```
git clone git://github.com/ipython/ipython.git # Official clone
cd ipython
git remote add carreau git://github.com/Carreau/ipython.git # Matthias' branch
git fetch carreau # Fetch the branches
git checkout carreau/slideshow_extension2 # Checkout the slideshow extension
python setup.py develop # Install the development version
ipython notebook # Check out the slideshows.
```

0.3 Acknowledgements

- Much of this tutorial adapts slide material from William Gropp, University of Illinois
- mpi4py examples developed by Lisandro Dalcin
- mpi4py is a Cythonized wrapper around MPI originally developed by Lisandro Dalcin, CONICET

1 What is MPI

- Message Passing Interface
- Most useful on distributed memory machines
- Many implementations, interfaces in C/C++/Fortran and Python
- Why python?
- Great for prototyping
- Small to medium codes
- Can I use it for production?
- Yes, if the communication is not very frequent and rapid development is the primary concern

1.1 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!

1.2 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)

1.3 Communicators

- 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)

1.4 Datatypes

- the data in a message to send or receive is described by address, count and datatype
- a datatype is recursively defined as:
- predefined, corresponding to a data type from the language (e.g., `MPI_INT`, `MPI_DOUBLE`)
- a contiguous, strided block, or indexed array of blocks of MPI datatypes
- an arbitrary structure of datatypes
- there are MPI functions to construct custom datatypes

1.5 Tags

- messages are sent with an accompanying user-defined integer tag to assist the receiving process in identifying the message
- messages can be screened at the receiving end by specifying the expected tag, or not screened by using `MPI_ANY_TAG`

1.6 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: mpirun –np < number of process > -machinefile < hostlist > python < my MPI4Py python script >

1.7 Notes about Communicators

- COMM_WORLD is available (MPI.COMM_WORLD)
- To get size: MPI.COMM_WORLD.Get_size() or MPI.COMM_WORLD.size
- To get rank: MPI.COMM_WORLD.Get_rank() or MPI.COMM_WORLD.rank
- To get group (MPI Group): MPI.COMM_WORLD.Get_group() . This returns a Group object
- Group objects can be used with Union(), Intersect(), Difference() to create new groups and new communicators using Create()
- To duplicate a communicator: Clone() or Dup()
- To split a communicator based on a color and key: Split()
- Virtual topologies are supported!
- Cartcomm, Graphcomm, Distgraphcomm fully supported
- Use: Create_cart(), Create_graph()

1.8 First Example: HelloWorld

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.

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

%load_ext parallelmagic
view.activate()
```

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

```
view.block = True
```

```
%autopx
```

```
%autopx enabled
```

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

```
from mpi4py import MPI

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

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

```
[stdout:0]
Helloworld! I am process 0 of 4 on orchid.local.

[stdout:1]
Helloworld! I am process 3 of 4 on orchid.local.

[stdout:2]
Helloworld! I am process 1 of 4 on orchid.local.

[stdout:3]
Helloworld! I am process 2 of 4 on orchid.local.
```

1.9 MPI Basics: (Blocking) Send

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

Python (mpi4py)

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

1.10 MPI Basics: (Blocking) Recv

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

Python (mpi4py)

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

1.11 MPI Basics: Synchronization

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

Python (mpi4py)

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

1.12 Timing and Profiling

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

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

1.13 Send/Receive Example (lowercase convenience methods)

```
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}
```

1.14 Send/Receive Example (MPI API on numpy)

```
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(100, dtype=numpy.float64)
   comm.Send(data, dest=1, tag=13)
elif rank == 1:
   data = numpy.empty(100, dtype=numpy.float64)
   comm.Recv(data, source=0, tag=13)
   print data
```

```
[stdout:2]
[  0.   1.   2.   3.   4.   5.   6.   7.   8.   9.  10.  11.  12.  13.  14.
  15.  16.  17.  18.  19.  20.  21.  22.  23.  24.  25.  26.  27.  28.  29.
  30.  31.  32.  33.  34.  35.  36.  37.  38.  39.  40.  41.  42.  43.  44.
```

```
45.  46.  47.  48.  49.  50.  51.  52.  53.  54.  55.  56.  57.  58.  59.
60.  61.  62.  63.  64.  65.  66.  67.  68.  69.  70.  71.  72.  73.  74.
75.  76.  77.  78.  79.  80.  81.  82.  83.  84.  85.  86.  87.  88.  89.
90.  91.  92.  93.  94.  95.  96.  97.  98.  99.]
```

1.15 Broadcast/Scatter/Gather

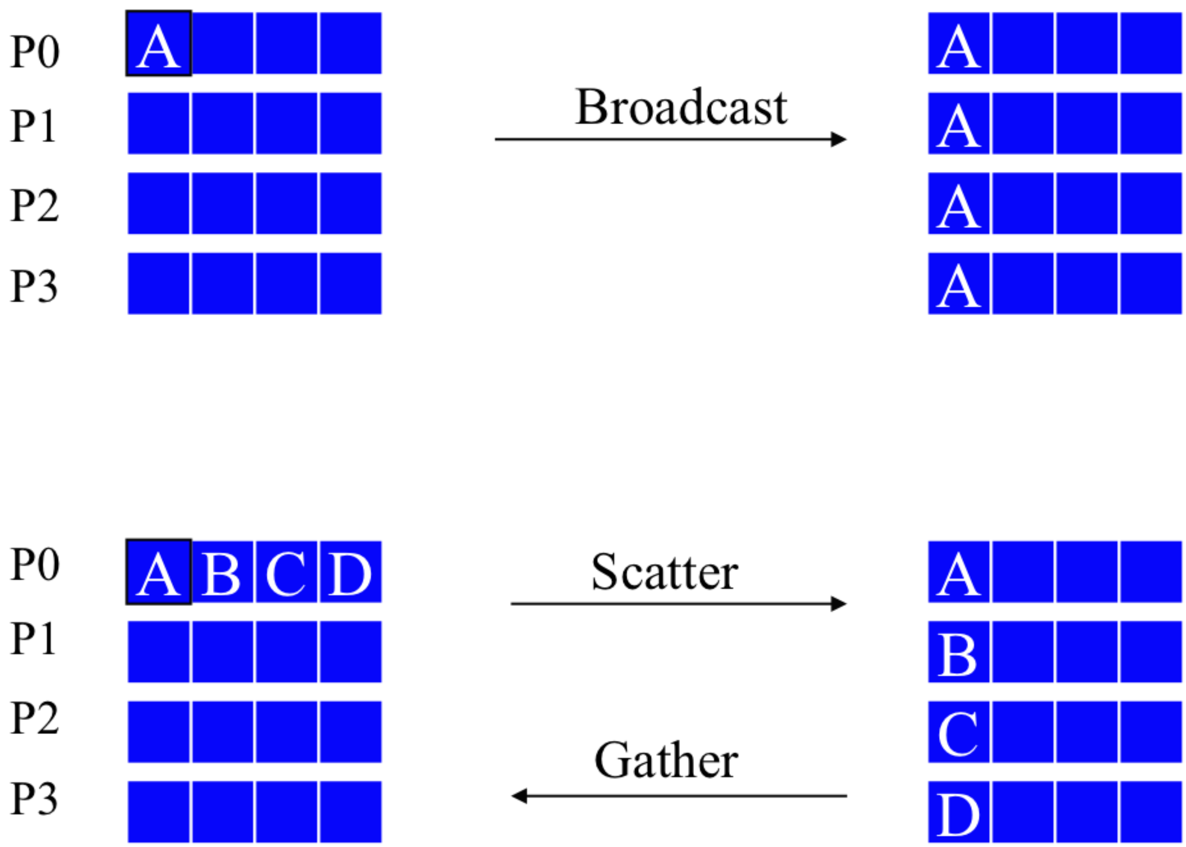


Figure 1: broadcast_scatter_gather

1.16 Broadcast Example

```python
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: "%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)]}
```

1.17 Scatter Example:

```
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
```

1.18 Gather (and Barrier) Example:

```
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
```

1.19 Reduce and Scan

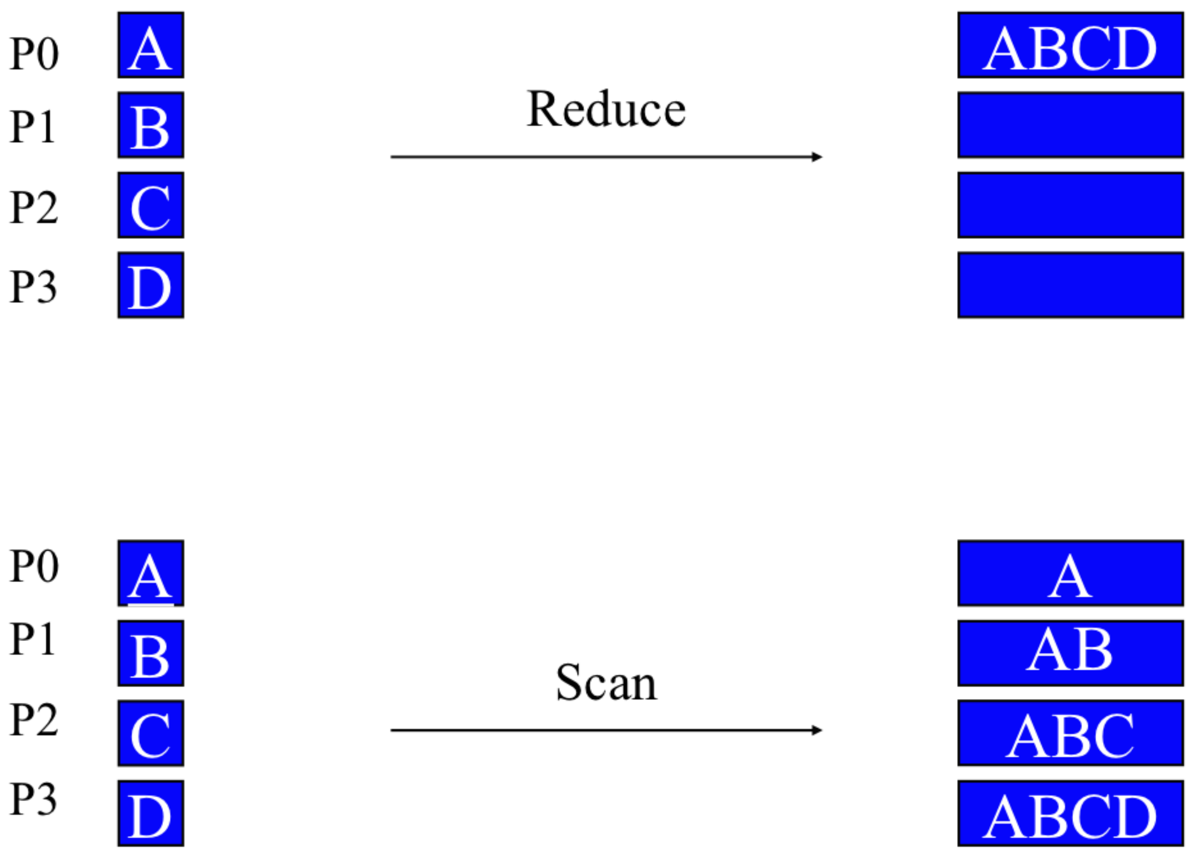


Figure 2: reduce_scan

1.20 Reduce Example:

```
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
```

1.21 Compute Pi Example

```
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
```

1.22 Mandelbrot Set Example

```
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=[C1, MPI.INT], recvbuf=[C, (counts, None), rowtype],root=0)

rowtype.Free()
```

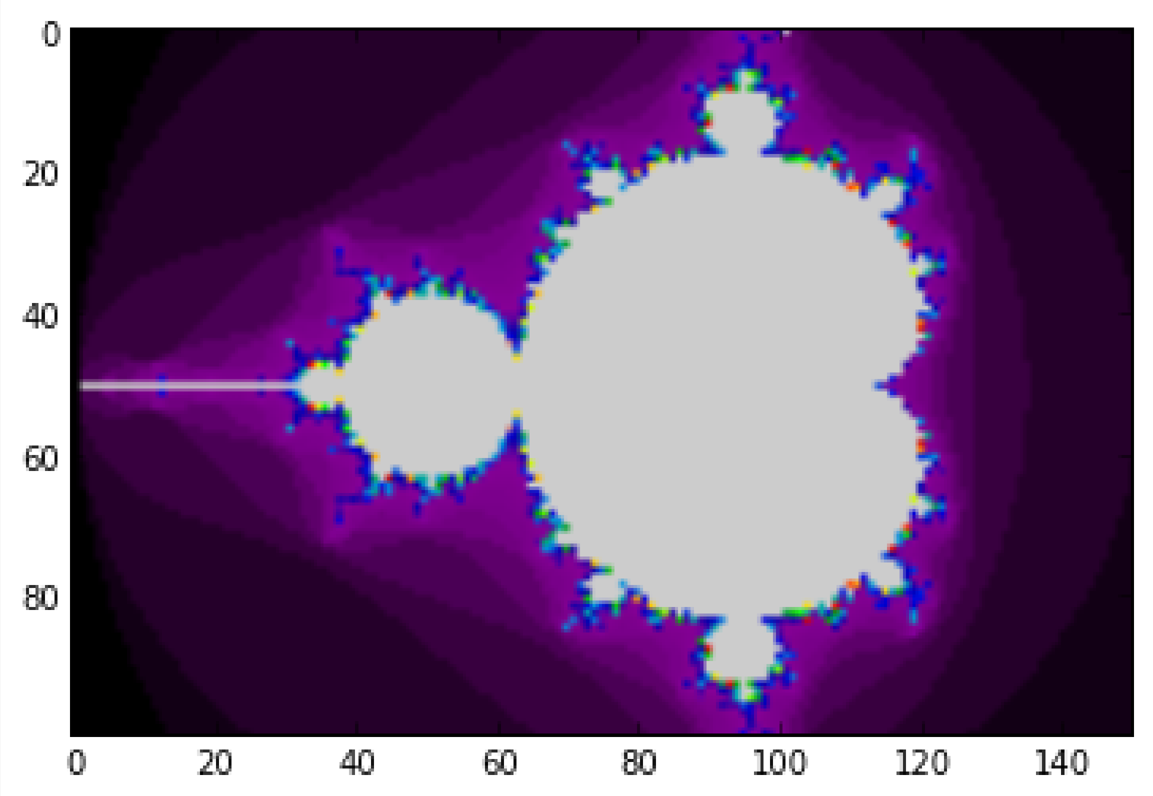
We now need to “pull” the C array for plotting so we disable autopx. Make sure to re-enable it later on

```
%autopx
```

```
%autopx disabled
```

```
# 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 id 0
pyplot.imshow(CC[ranks.index(0)], aspect='equal')
pyplot.spectral()
pyplot.show()
```



Toggle autopx back

```
%autopx
```

```
%autopx enabled
```

1.23 Advanced Capabilities

- MPI4Py supports dynamic processes through spawning: Spawn(), Connect() and Disconnect()
- MPI4PY supports one sided communication Put(), Get(), Accumulate()
- MPI4Py supports MPI-IO: Open(), Close(), Get_view() and Set_view()

1.24 More Comprehensive mpi4py Tutorials

* basics - http://mpi4py.scipy.org/docs/usrman/tutorial.html * advanced - http://www.bu.edu/pasi2011/01/Lisandro-Dalcin-mpi4py.pdf Timing and Profiling

1.25 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/

1.26 –> See Appendix 01 References for more resources