multiprocessing and mpi4py

Python for computational science
28 - 30 May 2012
CINECA

m.cestari@cineca.it

Bibliography

- multiprocessing

  http://docs.python.org/library/multiprocessing.html

  http://www.doughellmann.com/PyMOTW/multiprocessing/

- mpi4py

  http://mpi4py.scipy.org/docs/usrman/index.html

Introduction (1)

- Global Interpterer Lock, **GIL**, allows only a single thread to run in the interpreter
  - ➔ this clearly kills the performance

- There are different ways to overcome this limit and enhance the overall performance of the program
  - ➔ multiprocessing (python 2.6)
  - ➔ mpi4py (SciPy)

multiprocessing (1)

- Basically it works by forking new processes and dividing the work among them exploiting (all) the cores of the system

```
import multiprocessing as mp

def worker(num):
    """thread worker function"""
    print 'Worker:', num
    return

if __name__ == '__main__':
    jobs = []
    for i in range(5):
        p = mp.Process(target=worker, args=(i,))
        jobs.append(p)
        p.start()
```

multiprocessing (2)

- processes can communicate to each other via queue, pipes

- Poison pills to stop the process

- Let's see an example...

multiprocessing (3)

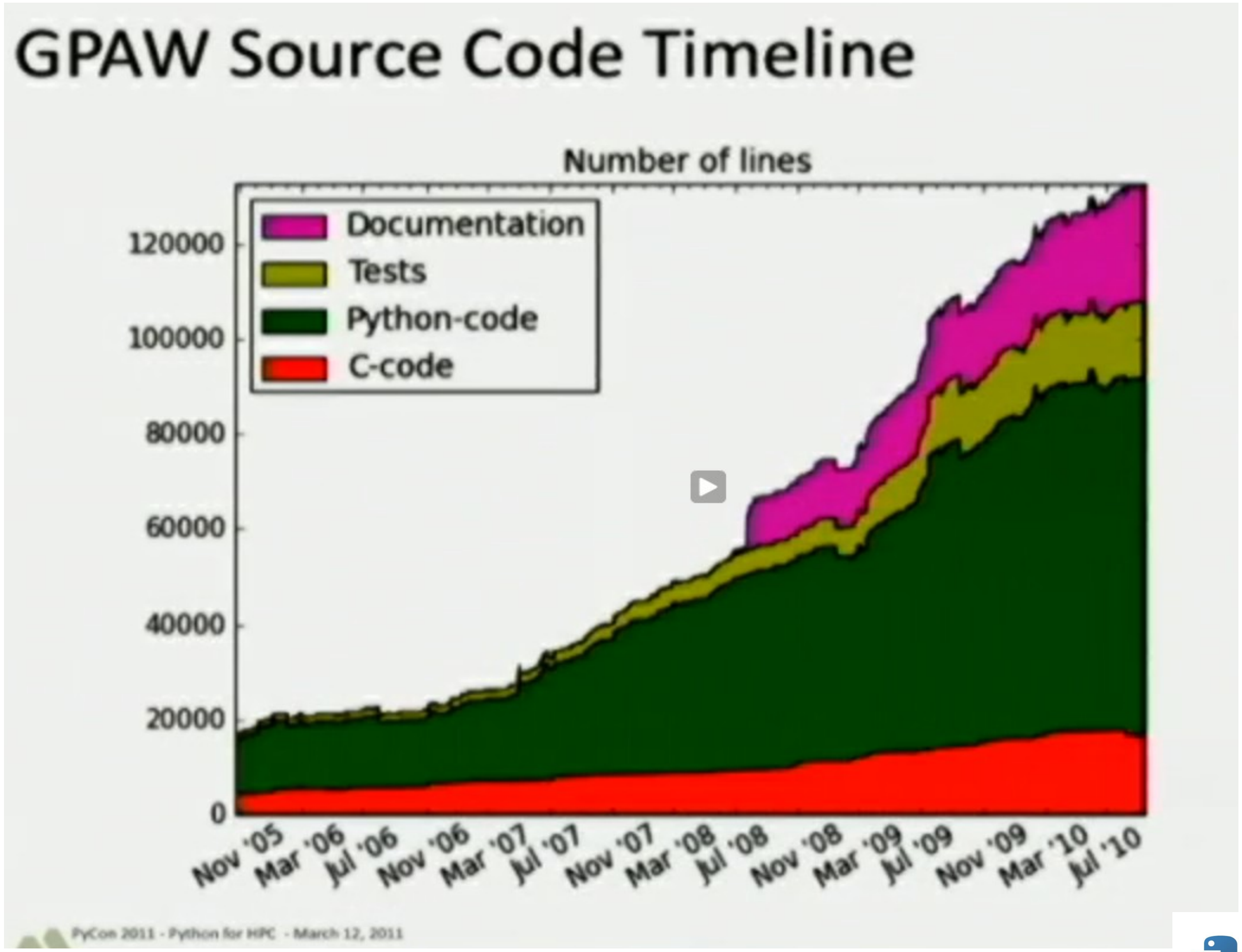
```
import multiprocessing as mp
import os

def worker(num, input_queue):
  """thread worker function"""
  print 'Worker:', num
  for inp in iter(input_queue.get, 'stop'):
    print 'executing %s' % inp
    os.popen('./mmul.x < '+inp+' >'+inp+'.out' )
  return

if __name__ == '__main__':
  input_queue = mp.Queue() # queue to allow IPC
  for i in range(4):
    input_queue.put('in'+str(i))   # the queue contains the
                                     name of the inputs
  for i in range(4):
    input_queue.put('stop')  # add a poison pill for each
                               process
  for i in range(4):
    p = mp.Process(target=worker, args=(i, input_queue))
    p.start()
```

Introduction (1)

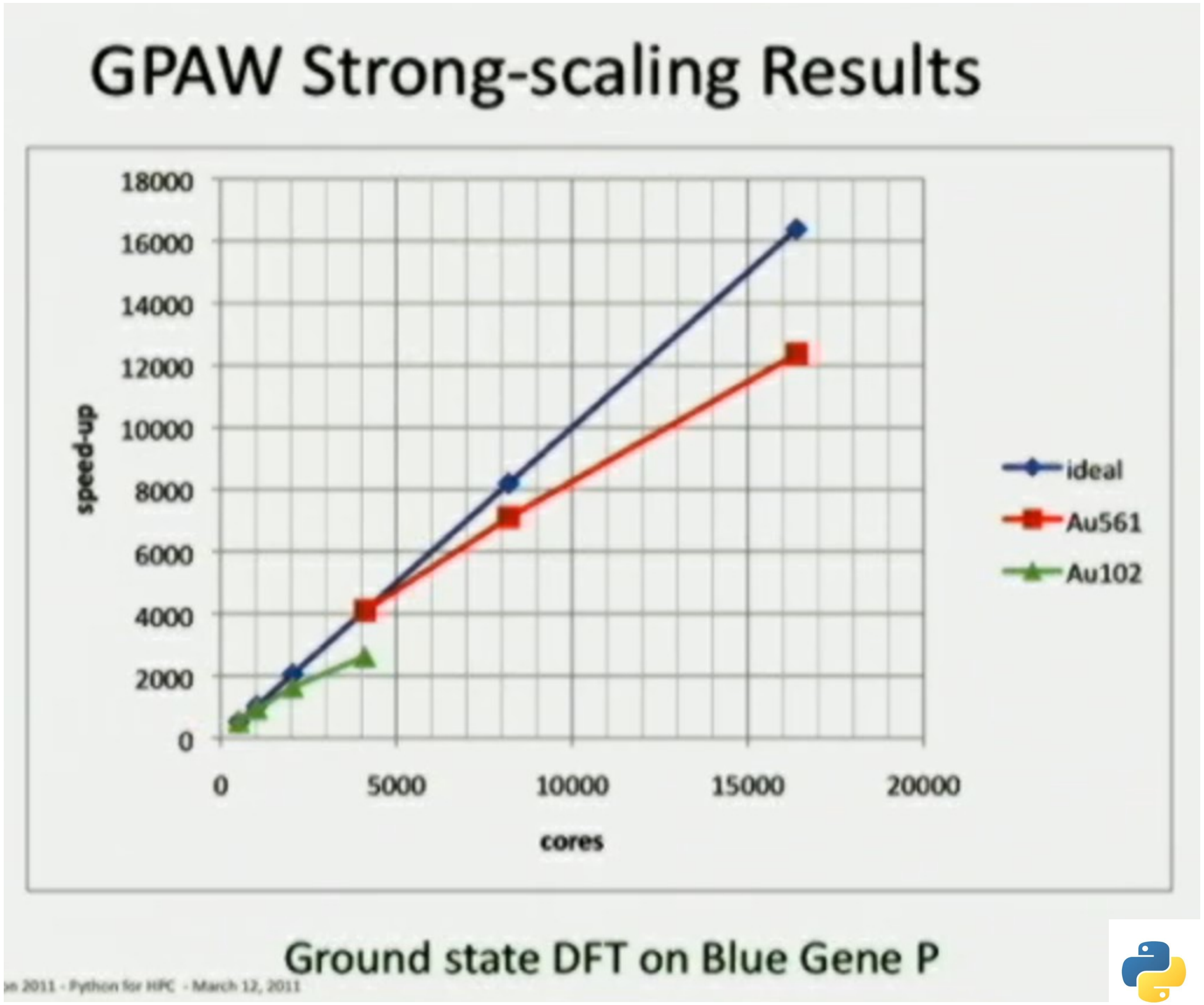
- **mpi4py** allows Python to sneak (its way) into HPC field
- For example:
    - the infrastructure of the program (MPI, error handling, ...) can be written in Python
    - Resource-intensive kernels can still be programmed in compiled languages (C, Fortran)
- new-generation massive parallel HPC systems (like bluegene) already have the Python interpreter on their thousands of compute nodes

Introduction (2)

- Some codes already have this infrastructure: i
  GPAW



Introduction (2)


GPAW Strong-scaling Results

Ground state DFT on Blue Gene P

What is MPI

- the Message Passing Interface, MPI, is a **standardized** and **portable** message-passing system designed to function on a wide variety of parallel computers.
- Since its release, the MPI specification has become the leading **standard for message-passing libraries** for parallel computers.
- mpi4py wraps the native MPI library

Performance

- Message passing with **mpi4py** generally is close to C performance from medium size to long arrays
- the overhead is about 5 % (near C-speed)
- This performance may entitle you to skip C/Fortran code in favor of Python
- To reach this communication performance you need use special syntax

Performance (2)

```
t0 = MPI.Wtime()

data = numpy.empty(10**7,
dtype=float)


if rank == 0:

    data.fill(7) # with 7

else:

    pass


MPI.COMM_WORLD.Bcast([data,
MPI.DOUBLE], root=0)

t1 = MPI.Wtime() - t0
```

```
t0 = MPI.Wtime()


if rank == 0:

    data = numpy.empty(10**7,
                 dtype=float)

    data.fill(7)

else:

    data = None


data=MPI.COMM_WORLD.bcast(data
, root=0)


t1 = MPI.Wtime() - t0
```

Performance (3)

```
numpy array comm tempo 0.068648815155

numpy array comm tempo 0.0634961128235

numpy array cPickle tempo 0.197563886642

numpy array cPickle tempo 0.18874502182
```

- example on the left is about 3 times faster than that one on the right
- The faster example exploits direct array data communication of buffer-provider objects (e.g., NumPy arrays)
- The slower example employs a pickle-based communication of **generic** Python object

Error handling

- Error handling is supported. Errors originated in native MPI calls will throw an instance of the exception class **Exception**, which derives from standard exception RuntimeError

mpi4py API

- mpi4py API libraries can be found the web site

  http://mpi4py.scipy.org/docs/apiref/index.html

To summarize

- mpi4py and multiprocessing can be a viable option to make your serial code parallel
- Message passing performance are close to compiled languages (like C, Fortran)
- Massive parallel systems, like bluegene, already have Python on their compute nodes