PP in Python using MPI technology

Mehti Musayev
M4115c academic group

Parallel Programming in Python

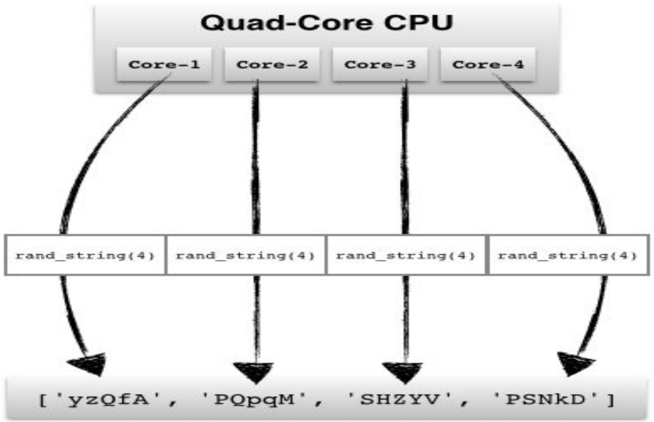
- Standard library:
  - threading
  - multiprocessing
  - concurrent.futures (Python 3)
  - asyncio (Python 3)
- External libraries:
  - mpi4py
  - IPython
  - and much more (parallel python, POSH, celery..)

Standard library: threading

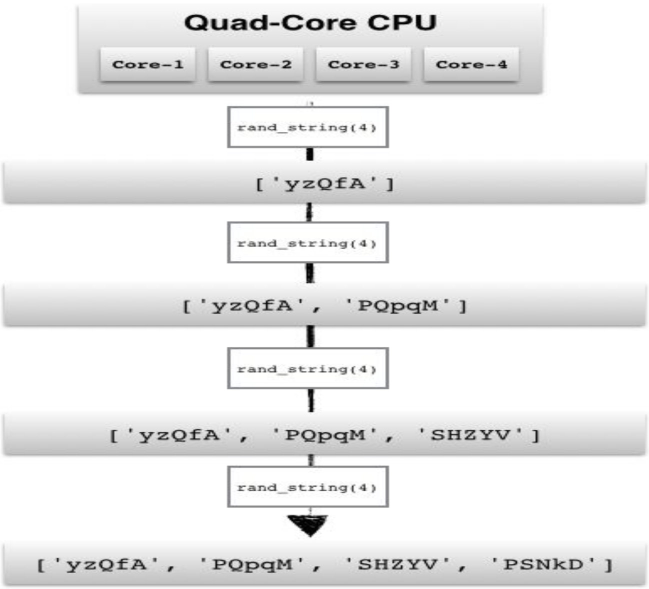
- Main PAIN in Python’s threading - Global Interpreter Lock (GIL)!!!
- It is intended to serialize access to interpreter internals from different threads
- It imposes various restrictions on threads. Namely, you can't utilize multiple CPUs
- Benefits:
  - Increased speed of single-threaded programs
  - Easy integration of C libraries that usually are not thread-safe
- Strongly prefer to use multiprocessing instead of threading in Python

Standard library: multiprocessing imagination

[parallel processing]



[serial processing]



Standard library: multiprocessing

- Pool
- Process
- Context and start methods
- Exchanging objects between processes
- Synchronization between processes
- Sharing state between processes
- Using a pool of workers

Standard library: multiprocessing

```python
from multiprocessing import Pool


def f(x):
    return x * x


if __name__ == '__main__':
    with Pool(5) as p:
        print(p.map(f, [1, 2, 3]))
```

Standard library: concurrent.futures

- Executor Objects (submit, map, shutdown)
- ThreadPoolExecutor
- ProcessPoolExecutor
- Future Objects (cancel, cancelled, running, done, result, exception..)
- Module Functions (wait, as_completed)

Standard library: concurrent.futures

```python
import concurrent.futures
import urllib.request

URLS = ['http://www.foxnews.com/',
        'http://www.cnn.com/',
        'http://europe.wsj.com/',
        'http://www.bbc.co.uk/',
        'http://some-made-up-domain.com/']


# Retrieve a single page and report the URL and contents
def load_url(url, timeout):
    with urllib.request.urlopen(url, timeout=timeout) as conn:
        return conn.read()


# We can use a with statement to ensure threads are cleaned up promptly
with concurrent.futures.ThreadPoolExecutor(max_workers=5) as executor:
    # Start the load operations and mark each future with its URL
    future_to_url = {executor.submit(load_url, url, 60): url for url in URLS}
    for future in concurrent.futures.as_completed(future_to_url):
        url = future_to_url[future]
        try:
            data = future.result()
        except Exception as exc:
            print('%r generated an exception: %s' % (url, exc))
        else:
            print('%r page is %d bytes' % (url, len(data)))
```

Standard library: asyncio

- Event loop
- Coroutines
- Futures
- Tasks

Standard library: asyncio

```
import asyncio


@asyncio.coroutine
def sleep_coroutine(f):
    yield from asyncio.sleep(2)
    f.set_result("Done!")
```

External libraries: mpi4py

- Communicating Python Objects and Array Data
- Communicators
- Point-to-Point Communications
  - Blocking Communications
  - Nonblocking Communications
  - Persistent Communications
- Collective Communications
- Dynamic Process Management
- One-Sided Communications
- Parallel Input/Output

External libraries: mpi4py

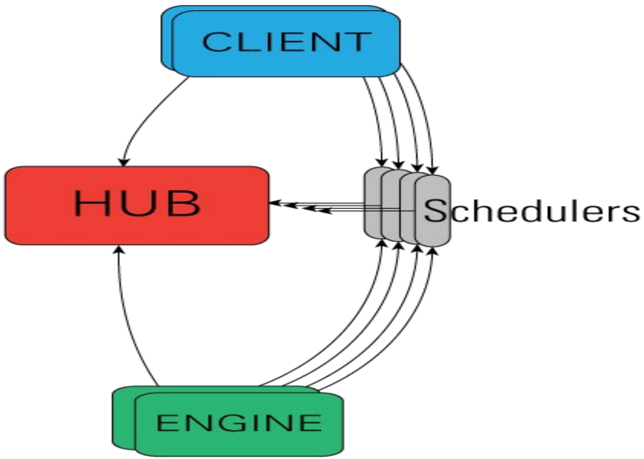
```python
from mpi4py import MPI
import numpy

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
# passing MPI datatypes explicitly
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)
# 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)
```

External libraries: IPython

- Interactively helps to learn Python
- A kernel for Jupiter (tool to run IPython in the browser and widely used in Data Science)
- Interactive data visualization
- **Can be used to run parallel programs using different approaches and tools**

External libraries: IPython

- Single program, multiple data (SPMD) parallelism
- Multiple program, multiple data (MPMD) parallelism
- **Message passing using MPI**
- Task farming
- Data parallel
- Combinations of these approaches
- Custom user-defined approaches



External libraries: IPython with MPI

- Requirements: MPI Implementation (such as OpenMPI or MPICH) and mpi4py package
- Getting started with MPI: `ipcluster start -n 4 --engines=MPIEngineSetLauncher`

```python
from mpi4py import MPI
import numpy as np


def psum(a):
    locsum = np.sum(a)
    rcvBuf = np.array(0.0, 'd')
    MPI.COMM_WORLD.Allreduce([locsum, MPI.DOUBLE],
                             [rcvBuf, MPI.DOUBLE],
                             op=MPI.SUM)
    return rcvBuf
```

- Running: `ipcluster start --profile=mpi -n 4`

Industrial usages

- Data Scientists: mainly IPython (also Apache Spark too)
- Backend developers: multiprocessing, asyncio (Python 3), celery, gevent, rabbitmq/zeromq and much more..
- **Normally,** people **don't** use **threading** from standard library!!!

References

- POSH: http://poshmodule.sourceforge.net/posh/posh.pdf
- About GIL: http://www.dabeaz.com/python/GIL.pdf
- Parallel Programming with Python, Jan Palach
- Python 3 documentation: https://docs.python.org/3/
- IPython Parallel: https://ipyparallel.readthedocs.io/en/latest/index.html
- mpi4py: http://pythonhosted.org/mpi4py/
- Presentation: https://github.com/b5y/parallel_programming

Thank you for attention!

- Email: mehty.musaev@gmail.com
- c.p.: +79110395135
- LinkedIn: https://linkedin.com/in/mehtimusayev