How To Overcome the GIL Limitations

(While Staying In Python Ecosphere)

Francesc Alted

Numexpr Author
Barcelona Music and Audio Technology

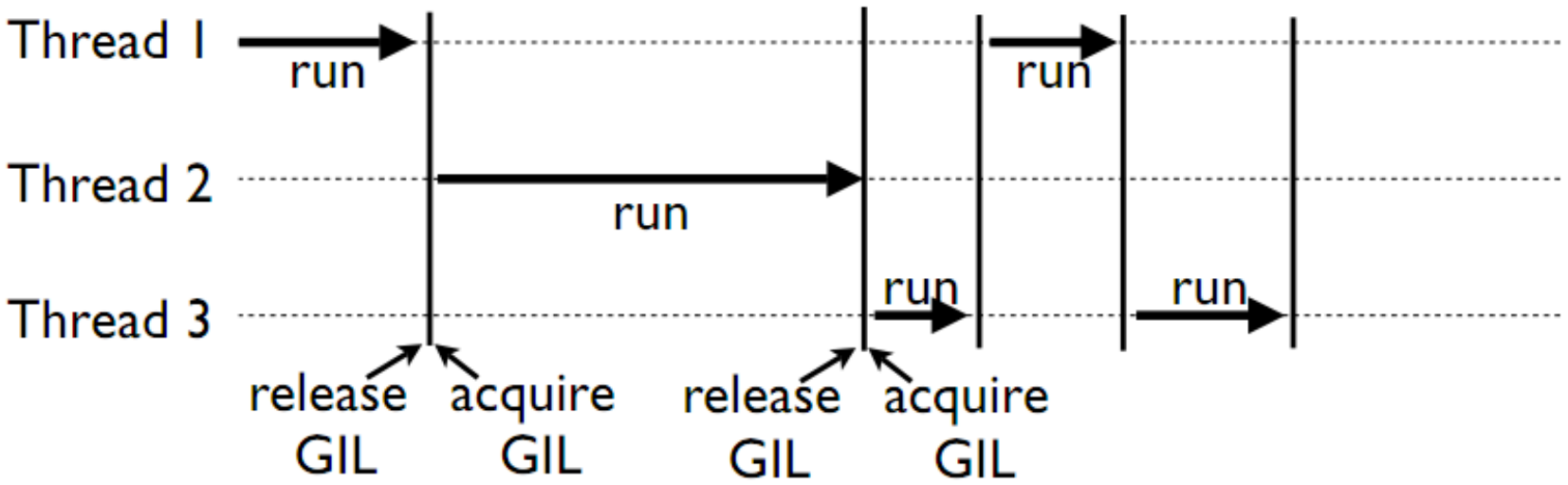
Advanced Scientific Programming in Python
2011 Summer School, St Andrews, Scotland

Outline

What the "GIL" Is?

- GIL stands for "Global Interpreter Lock".
- Its main goal is to avoid race conditions to happen among different Python threads, allowing the Python interpreter to work flawlessly in threading environments.
- This is achieved by preventing threads to use the Python interpreter **simultaneously** while they run.

How the GIL Works



Actually prevents threads from running in **parallel** in Python.

If GIL Is So Bad, Why It Was Introduced?

- Actually, it is **not that bad**:
  - Module developers do not need to worry about evil threading effects.
  - It is unproblematic (and helpful!) when running several I/O bounded threads *simultaneously.*
  - C extensions usually releases the GIL during operation, allowing to use several threads concurrently.

The GIL is only a problem when tackling CPU-bounded problems in Python, where it effectively prevents threads from running in parallel.

If GIL Is So Bad, Why It Was Introduced?

- Actually, it is **not that bad**:
  - Module developers do not need to worry about evil threading effects.
  - It is unproblematic (and helpful!) when running several I/O bounded threads *simultaneously.*
  - C extensions usually releases the GIL during operation, allowing to use several threads concurrently.

The GIL is only a problem when tackling CPU-bounded problems in Python, where it effectively prevents threads from running in parallel.

Ways To Avoid the GIL

- Use threaded extensions in C where GIL is not a problem (Numexpr, NumPy with MKL, SciPy with FFTW...):
  - Pro: powerful and very easy to use.
  - Con: not general enough (not a programming language).
- Use the `multiprocess` module:
  - Pro: completely compatible with the threading module.
  - Con: data is not shared among processes.
- Use Cython's parallel features:
  - Pro: uses native threads internally, so data is shared naturally.
  - Con: Cython is compiled.

Cython Parallel Features

Since Cython 0.15, it actually supports a couple of ways to create true CPU-bounded parallel apps:

- Using Python's `threading` module.
- Use `prange` statement in loops.

You should always remember to release the GIL explicitely.

Cython Parallel Features

Since Cython 0.15, it actually supports a couple of ways to create true CPU-bounded parallel apps:

- Using Python's `threading` module.
- Use `prange` statement in loops.

You should always remember to release the GIL explicitely.

Releasing the GIL In Cython

Cython allows to explicitly release the GIL in code sections:

```
cdef np.ndarray x[double]
cdef double beta
cdef long i
with nogil:
    for i in range(1000):
        x[i] = sin(i * beta)
```

...or in complete functions:

```
cdef int myfunc(np.ndarray[double] x, double beta) nogil:
    cdef long i
    for i in range(1000):
        x[i] = sin(i * beta)
```

The code sections in `nogil` blocks cannot call code in Python space. Else, Cython will abort compilation.

Using Cython's prange

'Canonical' and easiest way to parallelize loops in Cython:

```
for i in prange(N, nogil=True):
    x[i] = sin(i * beta)
```

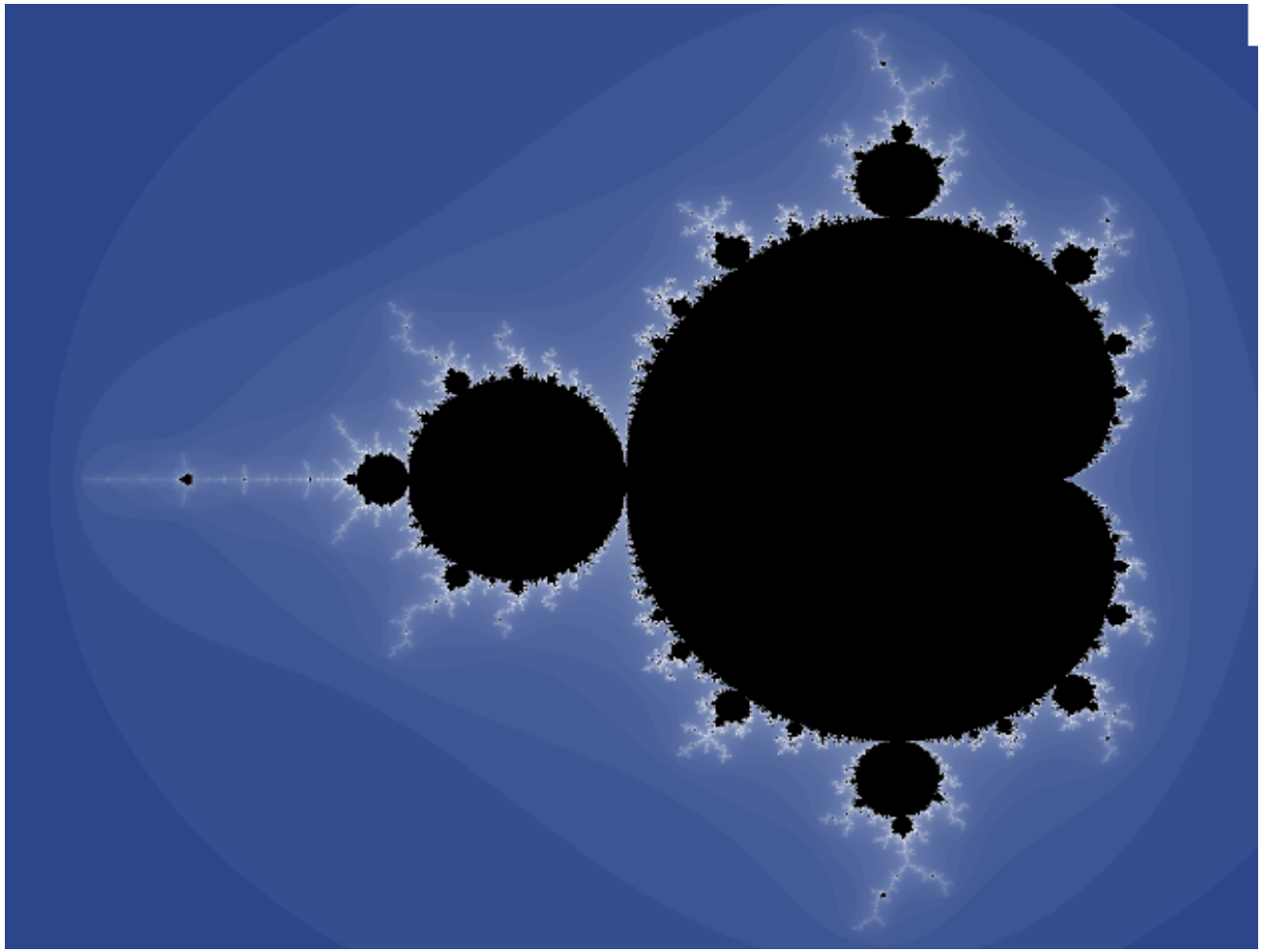
- This releases the GIL explicitely in the prange loop.
- Cython's prange makes use of OpenMP in the C output.

A Few Notes About OpenMP

OpenMP is an open standard API for programming threads in C, C++ and Fortran.

- It is a set of:
  - Compiler directives.
  - Library routines.
  - Environment variables that influence run-time behaviour.
- Supported by most of current compilers: GCC, Microsoft MSVC, Intel ICC...
- It only runs efficiently in shared-memory multicore platforms.

An Example Of Parallel Code with Cython



Summary

- The GIL was introduced in Python for a good reason, and it is not going to disappear anytime soon.
- There are several solutions that allow to circumvent GIL limitations for threading apps. Threaded extensions, `multiprocess` and Cython are among the most interesting ones.
- Cython is the recommended way to go when you need to program your solution and really want to squeeze all the performance out of your cores in shared memory configurations.

More Info

- David Beazley
  Understanding the Python GIL
  http://www.dabeaz.com/GIL/
- The Cython crew
  Cython: C-Extensions for Python
  http://cython.org

What's Next

In the following exercises you will:

- Get familiar with some parallel code in Cython.
- Check which speed-ups can you achieve in multi-core processors.
- Learn curious things about scalability in different scenarios.

Thank You!

Contact:

faltet@pytables.org