E7441: Scientific computing in biology and biomedicine

Parallel programming in PYTHON

Vlad Popovici, Ph.D.

RECETOX

Modes of parallelism

Main modes

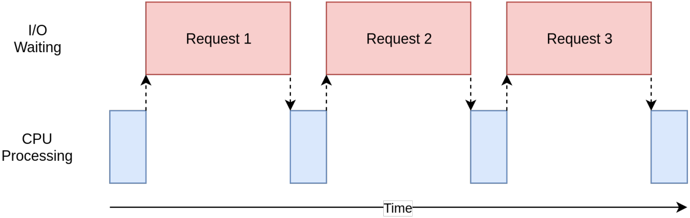
- *embarrassingly parallel*: code that can run independently and the results combined at the end (e.g. apply a function to each element of an array)
- *multithreading*: parallel threads of execution that needs to communicate via shared memory (variables, etc)
- *multiprocessing*: different processes that manage their own memory and share data via message passing

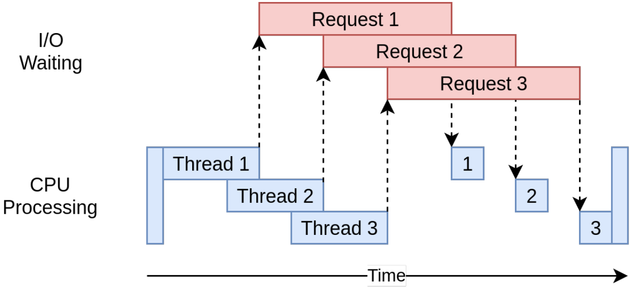
GIL

PYTHON

- interpreted language: source code is compiled into *bytecode* which is interpreted by the interpreter
- there are optimizing implementations (e.g. PYPY) that interpret and compile into optimized *machine code*
- the standard interpreter (*CPython*) executes only one thread at a time
- only the thread which acquired the *Global interpreter Lock* (GIL) (a *mutex*) can execute
- on multi-threaded systems this is a performance-bounding design
- GIL protects the reference count - helps with memory management
- GIL allows integration of non-thread-safe modules written in other languages
- GIL is released at I/O or forced-released at specific intervals

I/O-bound applications





Multiprocessing and multithreading

Example 1 - see the Jupyter notebook.

- not much gain from distributing the computation
- the two threads fight to acquire the GIL
- possible solution: *multiprocessing*: each process has its own interpreter
- external libraries (written in C, etc.) can release the lock and run multi-threaded (e.g. `NumPy`, `SciPy`, etc.)
- non-standard implementations of PYTHON do not necessarily use GIL: JYTHON, IRONPYTHON, and PYPY - they have their own limitations
- checkout "'joblib"' library for a lightweight implementation of parallel processing

MPI - message passing interface

- tasks (cores) have a rank and are numbered 0, 1, 2, 3, etc.
- each task (core) manages its own memory
- tasks communicate and share data by sending messages
- high-level API for distributing and gathering information to/from other tasks
- all tasks typically run the entire code: needs care to avoid doing the same thing

Example 2 - see the Jupyter notebook.

Dask

- scale arrays (`numpy.array`) and data frames (`pandas.Dataframe`) across computing resources
- Dask extension to `scikit-learn`: `Dask-ML`
- transparently manages larger-than-memory arrays and data frames
- transparently scales from desktop to cloud resources

See Example 3 in the Jupyter notebook.

Questions?