Intro to concurrency / parallelism with Python

Michael Hirsch

CEDAR 2019 Workshop

Why discuss Python?

- Python is popular in heliophysics
- Other languages scale better (Go, Haskell)
- Concepts learned in Python apply to other languages

- Let’s review some basic terminology with informal examples

Concurrent / Parallel

- Concurrency example: Air traffic controller
  - Many planes in the air
  - Can direct one plane at a time
  - Planes can be handed off to other controllers on other frequencies
  - Planes let controller know when done with flight leg
- Parallel example: Aircraft pilot
  - Keeps critical gauges in view, ears on radio, reacts immediately
  - Has to keep all critical parameters in balance to avoid loss of control
  - Operates control surfaces simultaneously for desired flight path

Pipelining

- Example: JIT supply chain management – automotive assembly
- Many different vendors and parts needed to build a car
- If we wait too long to order a part, vendor can’t supply them fast enough
- If we order too many parts at once, we can’t store them all

Concurrency example: media file processing

- Let’s say Spotify decides it wants to reprocess all its media files with a new streaming codec (better quality & efficiency, say)
- Spotify decides it wants to do A/B testing with its most demanding classical music listeners, rather than convert all files at once (too expensive)
- We will use off-the-shelf tool (FFprobe) known to run robustly with many media types and corrupted files to find music of the desired Genre and Artists
- No compliers needed, just pure Python and the FFmpeg executables
- This is less efficient, but fastest to deploy

```
from pathlib import Path
import asyncio  # built into Python

async def main(path: Path):
    async for meta in probe.get_meta(path.iterdir()):
        filename = meta['format']['filename']
        artists = [s[‘artist’] for s in meta['streams’]]

async def get_meta(files: Iterable[Path]) -> AsyncGenerator
    futures = (ffprobe(file) for file in files)
    for future in asyncio.as_completed(futures):
        meta = await future
        yield meta
```

Pipeline dataflow management: Queues

- Sometimes we let data flow constrain resource usage
- Process each blob of information as it’s needed
- Queues can be part of the pipeline, to manage resources

Example: N CPU cores—for CPU-bound tasks, there is usually little benefit to running more than N tasks

- Running too many CPU-bound tasks leads CPU to thrash with context switching, use excessive RAM
- Queues can be used with pipelines to set a fixed number of concurrent tasks (e.g. Ntask = Ncpucores)

```python
import asyncio  # built into Python
from pathlib import Path

async def main(path: Path):
    q = asyncio.Queue()
    for filename in path.iterdir():
        await q.put(filename)

    tasks = [asyncio.create_task(convert(q)) for i in range(4)]
    await q.join()

    await asyncio.gather(*tasks, return_exceptions=True)

asyncio.run(main(Path(‘~/Music’).expanduser()))
```

```
async def convert(queue: asyncio.Queue, cmd: Sequence[str]):

    while True:
        file = await queue.get()
        cmd += file
        proc = await asyncio.create_subprocess_exec(*cmd)

        ret = await proc.wait()

        if ret != 0:
            print(f’{file} conversion failure',file=sys.stderr)

        queue.task_done()
```

Coroutines vs. ThreadPool vs. ProcessPool

- Coroutines in Python are managed by a single thread
- Coroutines can launch processes e.g. `asyncio.create_subprocess_exec`
- Some tasks are more amenable to concurrent.futures.ThreadPoolExecutor
  - Launches up to a specific number of threads
  - Heavier weight (RAM, CPU) than coroutines in many cases, but may be easier/more suitable
- Other tasks are suited for concurrent.futures.ProcessPoolExecutor
  - Similar API to ThreadPoolExecutor, yet the most heavyweight in overhead

My goals

- To understand benefits from concurrency, need to work with some practical examples (see reference slide)
- Don’t have to run to Go, Haskell, C++ for things that can be done in Python
- Best language can be one you program fastest in, if it doesn’t have too many performance, code style or licensing downsides

Python concurrency example repos

These are real working code

The repos are all under https://github.com/scivision

- findssh
- asyncio-subprocess-examples
- asyncio-subprocess-ffmpeg
- Pyfindfiles
- gitMC