Due to global interpreter lock (GIL), python threads do not provide efficient multi-core execution, unlike other languages such as golang. Asynchronous programming in python is focusing on single core execution. Happily, Nexedi python code base already supports high performance multi-core execution either through multiprocessing with distributed shared memory or by relying on components (NumPy, MariaDB) that already support multiple core execution.

The impatient reader will find bellow a table that summarises current options to achieve high performance multi-core performance in python.

High Performance multi-core Python Cheat Sheet

Use python whenever you can rely on any of...	**Use golang whenever you need at the same time...**
• multiprocessing • cython's nogil option • multi-core execution within wendelin.core distributed shared memory • cython's parallel module	• low latency • high concurrency • multi-core execution within single userspace shared memory • something else than cython's parallel module or nogil option

Here are a set of rules to achieve high performance on a multi-core system in the context of Nexedi stack.

- **use ERP5's CMFActivity by default, since it can solve 99% of your concurrency and performance problems on a cluster or multi-core system**;
- use parallel module in cython if you need to use all cores with high performance within a single transaction (ex. HTTP request);
- use threads in python together with nogil option in cython in order to use all cores on a multithreaded python process (ex. Zope);
- use cython to achieve C/C++ performance without mastering C/C++ as long as you do not need multi-core asynchronous programming;
- rewrite some python in cython to get performance boost on selected portions of python libraries;
- use numba to accelerate selection portions of python code with JIT that may also work in a dynamic context (ex. Wendelin python scripts);
- use a good WSGI http server behind a enough http2/QUIC server and you will be able to serve thousands of parallel http requests with low latency;
- asyncio, coroutines, greenlets, etc. may be useful to improve readability of asynchronous code through abstractions or syntactic sugar (async / await) compared to a hand-made state machine, but only outside ERP5;
- use golang if you need to develop a (1) low latency (2) highly concurrent (3) multi-core service such as a database that (4) can not be implemented through parallel module in cython.

Here are now some anti-rules:

- asyncio, coroutines, greenlets, etc. have no direct effect on performance and do not help using multiple cores efficiently;
- asynchronous systems may lead to slower performance (ex. Gevent) than systems with a blocking design (ex. Meinheld) - this is especially true whenever an async server acts as a frontend to a blocking server in the backend and increases congestion by not limiting the number of requests forwarded to the backend;
- multiple threads are not useful in Nexedi code except for I/O because anything that requires multiple threads has been delegated to NumPy or to the database;
- golang is not useful for concurrency within Nexedi stack context if low latency is not required, or if multiprocess is sufficient or if parallel module in cython can be applied.

And if we are talking about improving performance of existing code, always consider a way to determine where the bottlenecks are located before investing too much time in possibly useless optimisation effort, either by using profiling tools such as pprofile+zpprofile (already part of ERP5) or by doing quick-and-dirty experiments through minor code changes that are only meant to demonstrate the reality of a bottleneck improvement. Profiling was for example used to demonstrate that proxy roles in Python scripts lead to 30% slow down. Quick-and-dirty experiments through minor code changes were used to demonstrate how proxy fields in ERP5 Form could provide actual higher performance than previous implementation before starting their full implementation.

With this set of rules and anti-rules in mind, we can expect that Nexedi core development languages will be in the next 10 years:

1. python (backend)
2. SQL (backend)
3. javascript (frontend)
4. cython (accelerated backend)
5. C/C++ (database and high performance frontend)
6. golang (database and high performance frontend)

golang language is likely to replace C or C++ for low level services such as database or http servers. It will be a slow evolution since we do not expect to replace MariaDB any time soon and there is no urgency in replace Apache or NGINX with Caddy for example.

For new projects that combine (1) low latency (2) high concurrency (3) multi-core (4) and can not be implemented through parallel module in cython, golang should be the default choice at Nexedi. This is the case of the new version of NEO storage for Wendelin project. NEO needs (1) low latency, (2) high concurrency to reduce congestion whenever objects of different size are being accessed in a context of serialization from NEO client side, (3) multi-core to compress and uncompress data quickly and (4) seems difficult to implement with parallel module in cython.

The same reasoning seems to be applicable to new projects outside Nexedi. BesidesCaddy, we observe an increasing number of projects being developed in golang rather than in python, C or C++. Even though we have no strong personal preference, we recognize that golang is simpler to learn and use than C or C++ for many python programmers in the sense that it simplifies memory management, concurrency and security without losing control on low level system access. It should thus be a good choice for a company such as Nexedi since its global corporate adoption leads to a fairly low learning cost compared to C++ (or Rust).

Acknowledgements: Kirill Smelkov, Vincent Pelletier and Yusei Tahara for their contributions and suggestions to this post.

Links

- Efficiently Exploiting Multiple Cores with Python
- Numba (JIT for python)
- A million requests per second with Python (Japronto)
- Combining Coroutines with Threads and Processes (concurrent.futures and asyncio)
- How Celery fixed Python's GIL problem
- goless: Go-style Python
- uvloop: Blazing fast Python networking
- coroutines in cython
- Cython in Wikipedia
- Building cython code
- PyParallel
- Even with async/await, raw promises are still key to writing optimal concurrent javascript
- Nim language
- Caddy http/https server (golang) and comparison to nginx (and video)