Group 7: GIL-free Python

Code: `github.com/sueszli/nogil`

Python is really popular. In October 2024, Python surpassed JavaScript for the first time as the most popular language on GitHub's Octoverse[1]. This high-level, interpreted, garbage-collected and dynamically-typed language emphasizes code readability and simplicity both in its implementation and syntax. This emphasis on simplicity also reflects itself in the language's community, often referred to as the "Pythonic" way of doing things.

But the GIL is a problem. However, the language's simplicity comes at a cost: While memory and network-bound tasks can be efficiently managed through colored functions[2] and `asyncio`, Python is notoriously slow for compute-bound tasks due to the Global Interpreter Lock (GIL)[3]. The GIL is a mutex in Python's most popular implementation, CPython, that protects access to Python objects, preventing multiple threads from executing Python bytecodes simultaneously and therefore limits the language's performance on multi-core systems - often making it over 1000x slower than its system-level counterparts on common benchmarking algorithms[4].

Or as Rob Pike put it in 2012[5]:

> *"The computing landscape today is almost unrelated to the environment in which the languages being used, mostly C++, Java, and Python, had been created. The problems introduced by multicore processors, networked systems, massive computation clusters, and the web programming model were being worked around rather than addressed head-on. Moreover, the scale has changed: today's server programs comprise tens of millions of lines of code, are worked on by hundreds or even thousands of programmers, and are updated literally every day. To make matters worse, build times, even on large compilation clusters, have stretched to many minutes, even hours."*

There are workarounds. These limitations have led to the development of various workarounds, such as the development of competing superset languages such as the `taichi`[6] and `mojo`[7], optimizing Python interpreters like `PyPy`[8] and `Numba`[9], proposing to introduce multiple lightweight sub-interpreters[10] [11], or even making the GIL entirely optional[12] [13] [14], as proposed in PEP 703[15].

The latter approach, making the GIL optional, was recently accepted in Python 3.13 and is currently in the experimental stage. There is no guarantee of this feature being included in the final release, but it is a step in the right direction.

Let's find out how they work. Chosen algorithm: hashcat

- on password storage: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
- we use a simpler one
- no algorithmic optimizations (e.g. rainbow tables, bloom filters, etc.) just brute-force

Cpython dependency `Python.h`: https://github.com/python/cpython/blob/main/Include/Python.h

Experiments

command	mean	stddev	median	user	system	min	max
plain: itertools.py	0.5674692	0.0119883	0.5700655	0.5599028	0.0074184	0.5496380	0.5865690
plain: lib.py	0.1003698	0.0026018	0.0995592	0.0950988	0.0051505	0.0976092	0.1086287
plain: plain.py	0.5631182	0.0085463	0.5607433	0.5574100	0.0056163	0.5564152	0.5863659
multiprocessing: imap_unordered.py	0.2258019	0.0085966	0.2235880	0.5456730	0.1661910	0.2184310	0.2449692
multiprocessing: imap.py	0.2328316	0.0065632	0.2306093	0.5554529	0.1625106	0.2235672	0.2426373
multiprocessing: map_async.py	0.4528332	0.0248580	0.4485649	1.0100400	0.1108107	0.4314743	0.5167882
multiprocessing: map.py	0.4400746	0.0043315	0.4405771	0.9853628	0.1084339	0.4329375	0.4467715
multithreading: GIL=1 executor.py	0.3696592	0.0103798	0.3658508	0.3597924	0.0092238	0.3621005	0.3968615
multithreading: GIL=0 executor.py	0.2102704	0.0104267	0.2121045	0.4389796	0.0105094	0.1962787	0.2321854
multithreading: GIL=1 workers.py	0.1304648	0.0071726	0.1292655	0.1178397	0.0081775	0.1242261	0.1582329
multithreading: GIL=0 workers.py	0.1677349	0.0076839	0.1685633	0.1931002	0.0202853	0.1429964	0.1760283

[1]https://github.blog/news-insights/octoverse/octoverse-2024/#the-most-popular-programming-languages
[2]https://langdev.stackexchange.com/questions/3430/colored-vs-uncolored-functions
[3]Wang, Z., Bu, D., Sun, A., Gou, S., Wang, Y., & Chen, L. (2022). An empirical study on bugs in python interpreters. IEEE Transactions on Reliability, 71(2), 716-734.
[4]https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html
[5]https://go.dev/talks/2012/splash.article
[6]https://www.taichi-lang.org/
[7]https://docs.modular.com/mojo/stdlib/python/python.html
[8]https://www.pypy.org/
[9]https://numba.pydata.org/
[10]https://peps.python.org/pep-0554/
[11]https://peps.python.org/pep-0683/
[12]https://peps.python.org/pep-0703/
[13]https://discuss.python.org/t/a-steering-council-notice-about-pep-703-making-the-global-interpreter-lock-optional-in-cpython/30474
[14]https://engineering.fb.com/2023/10/05/developer-tools/python-312-meta-new-features/
[15]https://peps.python.org/pep-0703/

command	mean	stddev	median	user	system	min	max
ctypes: invoke_hashcat.py	0.0934947	0.0031416	0.0929726	0.0882496	0.0049164	0.0891827	0.0996272
ctypes: invoke_hashcat.py (openmp)	0.1021338	0.0056378	0.1003631	0.0986943	0.0083828	0.0976012	0.1269725
cpython: invoke_hashcat.py	0.1006056	0.0043579	0.0997623	0.0950439	0.0052310	0.0943297	0.1081794

Target hash: `aaa`

Warmup: 3 runs

Docker with Python 3.13t experimental build

Addendum

All experiments were conducted on a consumer-grade laptop with the following specifications:

```
$ system_profiler SPSoftwareDataType SPHardwareDataType

Software:

    System Software Overview:

      System Version: macOS 14.6.1 (23G93)
      Kernel Version: Darwin 23.6.0
      Boot Volume: Macintosh HD
      Boot Mode: Normal
      Computer Name: Yahya's MacBook Pro
      User Name: Yahya Jabary (sueszli)
      Secure Virtual Memory: Enabled
      System Integrity Protection: Enabled
      Time since boot: 79 days, 22 hours, 26 minutes

Hardware:

    Hardware Overview:

      Model Name: MacBook Pro
      Model Identifier: Mac14,10
      Model Number: Z174001ABD/A
      Chip: Apple M2 Pro
      Total Number of Cores: 12 (8 performance and 4 efficiency)
      Memory: 16 GB
      System Firmware Version: 10151.140.19
      OS Loader Version: 10151.140.19
      Serial Number (system): VCYQD0HH0G
      Hardware UUID: BEA4D09D-6651-54E1-A3F7-7FB78A7BF1AB
      Provisioning UDID: 00006020-001A284901E8C01E
      Activation Lock Status: Disabled

$ docker compose exec main lscpu
Architecture:                        x86_64
CPU op-mode(s):                      32-bit
Byte Order:                          Little Endian
CPU(s):                              12
On-line CPU(s) list:                 0-11
Thread(s) per core:                  1
Core(s) per socket:                  12
Socket(s):                           1
Vendor ID:                           0x61
Model:                               0
Stepping:                            0x0
BogoMIPS:                            48.00
Vulnerability Gather data sampling:  Not affected
Vulnerability Itlb multihit:         Not affected
Vulnerability L1tf:                  Not affected
Vulnerability Mds:                   Not affected
Vulnerability Meltdown:              Not affected
Vulnerability Mmio stale data:       Not affected
Vulnerability Reg file data sampling: Not affected
Vulnerability Retbleed:              Not affected
Vulnerability Spec rstack overflow:  Not affected
Vulnerability Spec store bypass:     Mitigation; Speculative Store Bypass disabled via prctl
Vulnerability Spectre v1:            Mitigation; __user pointer sanitization
Vulnerability Spectre v2:            Not affected
Vulnerability Srbds:                 Not affected
Vulnerability Tsx async abort:       Not affected
Flags:                               fp asimd evtstrm aes pmull sha1 sha2 crc32 atomics fphp asimdhp cpuid asimdrdm jscvt fcma lrcpc dcp
                                      ilrcpc flagm ssbs sb paca pacg dcpodp flagm2 frint
```

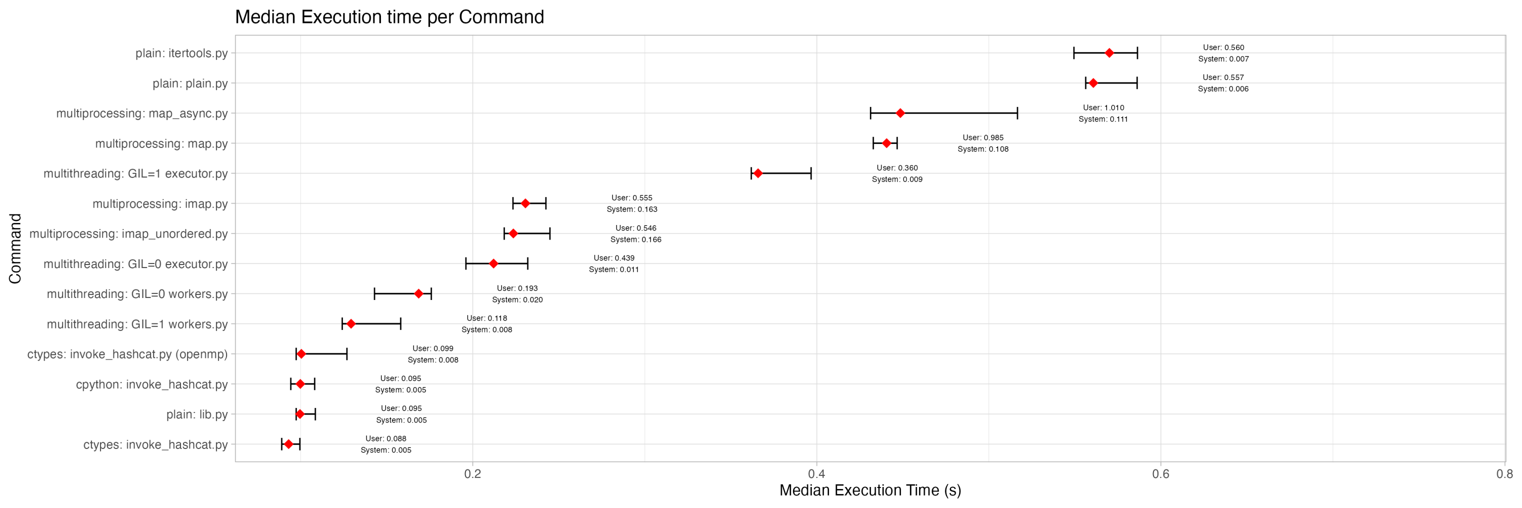

Figure 1: Performance Overview