GWDG
Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

# Highlights from the Python 3.13 release

Patrick Höhn (patrick.hoehn@uni-goettingen.de)

2024-11-13

# Agenda

- Statistics
- Optional GIL free Python interpreter
- Experimental JIT compiler
- Miscellaneous
- Summary

# Statistics-Release Timeline

- 3.13 development begins: Monday, 2023-05-22
- 3.13.0 alpha 1: Friday, 2023-10-13
- 3.13.0 alpha 2: Wednesday, 2023-11-22
- 3.13.0 alpha 3: Wednesday, 2024-01-17
- 3.13.0 alpha 4: Thursday, 2024-02-15
- 3.13.0 alpha 5: Tuesday, 2024-03-12
- 3.13.0 alpha 6: Tuesday, 2024-04-09
- 3.13.0 beta 1: Wednesday, 2024-05-08 (No new features beyond this point.)
- 3.13.0 beta 2: Wednesday, 2024-06-05
- 3.13.0 beta 3: Thursday, 2024-06-27
- 3.13.0 beta 4: Thursday, 2024-07-18
- 3.13.0 candidate 1: Thursday, 2024-08-01
- 3.13.0 candidate 2: Friday, 2024-09-06
- 3.13.0 candidate 3: Tuesday, 2024-10-01
- 3.13.0 final: Monday, 2024-10-07

# Statistics-Estimated Code Changes

- Contributors: 832
- Most active contributors: Victor Stinner, Serhiy Storchaka, Nikita Sobolev, Erlend E. Aasland, Sam Gross
- Commits: 6803
- Edited files: 14,471
- Additions: 666,843
- Removals: 473,325
- Total lines: 1,140,168

# Multithreading in Python - Example

```python
from concurrent.futures import ThreadPoolExecutor
from time import time

array_size = 1_000_000
count_tasks = 100

def write_by_index(size):
    # some working with local data
    array = [0 for i in range(size)]

start = time()
with ThreadPoolExecutor(max_workers=6) as executor:
    for index in range(count_tasks):
        executor.submit(write_by_index, array_size)
end = time() - start
print(end)
```

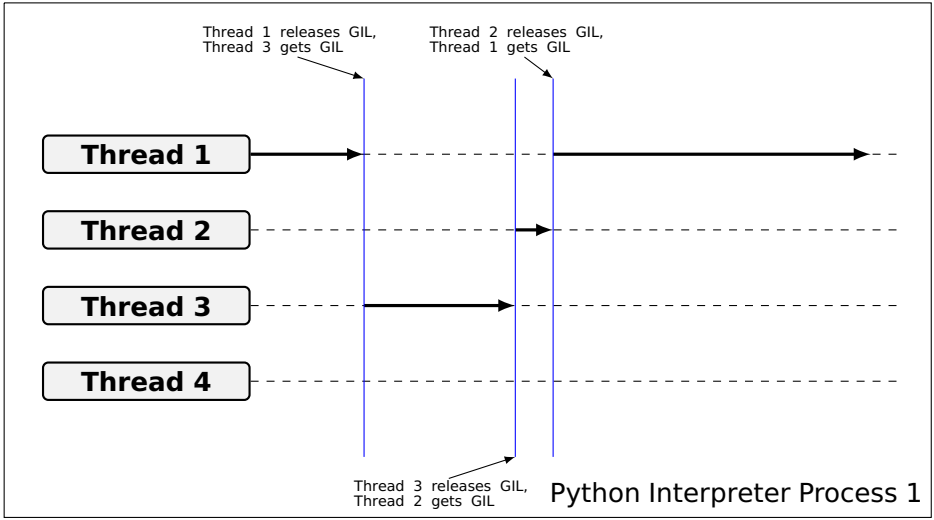
# Multithreading in Python - Example

```python
from concurrent.futures import ThreadPoolExecutor
from time import time

array_size = 1_000_000
count_tasks = 100

def write_by_index(size):
    # some working with local data
    array = [0 for i in range(size)]

start = time()
with ThreadPoolExecutor(max_workers=6) as executor:
    for index in range(count_tasks):
        executor.submit(write_by_index, array_size)
end = time() - start
print(end)
```

threaded: 1.41 s

plain: 1.27 s

# GIL in Action



# What is the GIL?

- Global Interpreter Lock
- Possibly multiple threads in one Python interpreter
- GIL guarantees that only one thread is active simultaneously
- No simultaneous modification of data in different threads

# Why is the GIL important?

- Avoiding race conditions in code
- Preserves thread security
- Simplifies Python Code
- Limits speedup using Python threading module
- Code expects GIL since the first Python releases

# Why should one remove the GIL?

- Starting of threads significantly faster than processes
- GIL limiting factor for scaling in ML applications since processes do not share CUDA contexts
- Communication overhead since serialization needed when communicating between processes
- Limitations in scaling of servers (REST API with multiple requests) or on interference platforms
- No need to rewrite Python code in C++ to circumvent GIL (worse accessibility to researchers)
- GPU-heavy workloads require multi-core processing, e.g. orchestrating GPU from CPUs

# Consequences of removing the GIL

- Enhanced performance on modern multi-core systems
- Performance degradation for single-thread execution
- Special build C-API extensions required for free-threaded build
- Possibly discovery of many undetected bugs
- Possible race conditions in code
- No more guaranteed thread security

# How to run Python without GIL?

- Installation
  - Installation of free-threaded binaries as part of official Windows and MacOS installers
  - Bulid from source with `--disable-gil` option
- Execution
  - environmental variable `PYTHON_GIL`
  - command-line option `-X gil=1`
- Verification
  - `python -VV` and `sys.version`
  - `sys._is_gil_enabled()`

# Installation repeipe for GIL-less CPython on ArchLinux

```
git clone
git checkout
./configure --disable-gil
make
```

# Demo showing advantages and traps of GIL-less Python

## main.py:

```python
from concurrent.futures import ThreadPoolExecutor
from time import time

array_size = 1_000_000
count_tasks = 100

def write_by_index(size):
    # some working with local data
    array = [0 for i in range(size)]

start = time()
with ThreadPoolExecutor(max_workers=6) as executor:
    for index in range(count_tasks):
        executor.submit(write_by_index, array_size)
end = time() - start
print(end)
```

## Execution:

```
PYTHON_GIL=1 python3.13-nogil main.py # 1.89 s
PYTHON_GIL=0 python3.13-nogil main.py # 0.59 s
```

# Demo showing advantages and traps of GIL-less Python

test.py:

```python
import time
import threading
import math

def test_function(num_thread):
    thread_start_time = time.time()
    print(F"Executing thread #{num_thread}...")
    math.factorial(250000)
    thread_execution_time = time.time() - thread_start_time
    print(F"...thread #{num_thread} finalized in {thread_execution_time} seconds")

print("Initializing thread test...")
start_time = time.time()
threads = []
for num_thread in range(5):
    thread = threading.Thread(target=test_function, args=(num_thread,))
    thread.start()
    threads.append(thread)
for thread in threads:
    thread.join()

execution_time = time.time() - start_time
print(f"Finalized! Execution time: {execution_time} seconds")
```

# Demo showing advantages and traps of GIL-less Python

Execution:

```
PYTHON_GIL=1 python3.13—nogil test.py # 2.30 s
PYTHON_GIL=0 python3.13—nogil test.py # 0.65 s
```

# What is a JIT compiler?

- Just In Time (JIT) compiler
- Compilation at run-time instead of build time
- Generally used for interpreted languages
- Possibly improved performance compared to plain interpreter mode

# Limitations of the experimental JIT compiler

- Challenging build with only combination `--enable-experimental-jit --disable-gil` known to work (https://github.com/python/cpython/issues/125246)
- So far limited performance improvements compared to “specializing adaptive interpreter” (PEP 659) introduced in CPython 3.11

# How to use the experimental JIT compiler

- Build LLVM 19 build dependency (https://github.com/python/cpython/blob/main/Tools/jit/README.md) using spack
- Patch LLVM package file to comment out line 90-94
- Load LLVM 19 module in spack
- Patch Tools/jit/_llvm.py to use LLVM version 19
- Build CPython from source with `--enable-experimental-jit --disable-gil` option

# Demo showing disadvantages and traps of the experimental JIT compiler

Test with fibonacci function:

```python
import timeit

def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a

if __name__ == "__main__":
    print(timeit.timeit('fibonacci(100)', globals=globals(), number=1_000_000))
```

Comparison of different Python versions:

- `python test_fib.py # ArchLinux v3.12:  1.54 s`
- `PYTHON_JIT=0 ./python # v3.13 GIL: 2.53 s`
- `PYTHON_JIT=1 ./python # v3.13 GIL JIT: 2.74 s`
- `PYTHON_JIT=1 PYTHON_GIL=0 ./python # v3.13 JIT NOGIL: 2.53 s`

# Other small highlights

- Android and iOS official platforms on Tier 3
- WebAssembly Tier 2
- 2 years full support, 3 years bugfixes
- More flexible interactive shell based on work from PyPy featuring multi-line editing and color support
- Paste Mode to insert full text ignoring empty lines by key "F3"
- History Browse Mode by key "F2"

# Other small highlights

- Compiler strips common leading whitespace from every line in a docstring
- `global` declarations are now permitted in except blocks when global is used in else block. Previously `SyntaxError`
- Change of location of a `.python_history` file via new environmental variable `PYTHON_HISTORY`
- New Module dbm.sqlite3: An SQLite backend for dbm
- New function fma() for fused multiply-add operations

# Summary related to Python Performance

- Experimental GIL-less interpreter
  - Clear potential of performance improvements
  - Many bugs to be uncovered but work in progress
- Experimental JIT compiler
  - So far no convincing performance improvements
  - Very experimental also in terms of compilation

# References


- https://www.heise.de/news/Python-3-13-Bessere-interaktive-Shell-und-endlich-Multithreading-ohne-GIL-9968435.html
- https://dev.to/doctorserone/performance-of-python-with-and-without-gil-4kaa
- https://github.com/python/cpython/issues/118749
- https://pythoninsider.blogspot.com/2024/10/python-3130-release-candidate-3-released.html
- https://docs.python.org/3.13/whatsnew/3.13.html
- https://peps.python.org/pep-0719
- https://peps.python.org/pep-0703

# References


- https://www.youtube.com/watch?v=HxSHIpEQRjs
- https://github.com/brandtbucher/brandtbucher/blob/master/2024/09/26/enabling_cpythons_jit_compiler.pdf
- https://github.com/brandtbucher/brandtbucher/blob/master/2023/10/10/a_jit_compiler_for_cpython.pdf