PARALLEL PROGRAMMING IN PYTHON 3.13

Mantvydas **Deltuva**

Justinas **Teselis**

Patras, 2025

CONTENTS

PARALLEL PROGRAMMING

Python has several modules to implement concurrency and parallelism, each suited for specific use cases. Whether dealing with I/O-bound or CPU-bound tasks, these modules provide the tools to manage threads, processes, and tasks effectively.

MODULE THREADING

Python threads are most effective for I/O-bound tasks therefore threading module can achieve concurrency in Python for tasks such as reading files, handling network requests or other tasks, where program has to wait for external resources. Memory space is shared across threads, making data sharing simple but prone to race conditions. To ensure thread-safe execution, module provides synchronization primitives such as locks, semaphores. This module is constrained by the GIL for CPU-bound tasks which limits true parallelism and its effectiveness.

MODULE MULTIPROCESSING

Python processes are most effective for CPU-bound tasks therefore multiprocessing module can achieve parallelism in Python for tasks such as numerical computations, data processing or other tasks, where program can benefit from multi-core CPU architectures by creating separate processes, each with its own memory space and CPU resources. To have clear communication with processes there are supported tools for it like pools, queue or pipe. This module overcomes the GIL limitation.

MODULE concurrent.futures

For unified usage of threads and processes there is concurrent.futures module. It simplifies the implementation of parallelism regarding management of threads, processes and tasks. It also includes pool executioners with helpful utility functions such as submit, map or as_completed. Threads and processes have the same characteristics as mentioned before, where threads are dependent on GLI and processes has overhead regarding memory.

GLOBAL INTERPRETER LOCK (GIL)

Global Interpreter Lock (GIL) is a mutex (mutual exclusion lock) that enforces rules for native threads for executing Python code simultaneously. This way it simplifies memory management, garbage collection and ensures thread safety. However, the main drawback is that it prevents achieving true multi-threaded applications.

GLOBAL INTERPRETER LOCK (GIL)

The GIL was introduced in Python 1.5 to simplify the implementation of it by:

- **Reducing complexity** - the lock eliminates the need for complex synchronization mechanisms across threads and thus makes it easier for development. Internal state remains consistent by preventing multiple threads from executing at the same time;
- **Managing memory** - the GIL ensures that memory allocation reference counts are updated safely (one by other) without race conditions;
- **Supporting extensions** - many extensions for Python depend on the GIL to avoid implementing their own thread-safety mechanisms. For extensions developers this reduces the complexity of thread-safe code.

GLOBAL INTERPRETER LOCK (GIL)

Despite its advantages, the GIL provides certain drawbacks:

- **Latency in I/O-bound scenarios** - context switching among threads introduces latency, which can degrade performance for applications with many concurrent and repeating I/O tasks;
- **Underutilization of multi-core systems** - today's hardware with multiple cores is not fully utilized due to the single-threaded code execution bottleneck by the GIL;
- **“True“ multi-threading** - as GIL only allows one thread to execute Python code at a time, the performance of multi-threaded applications designed for CPU-bound tasks has internal limitations by the language.
- **Workarounds** - developers often use multiprocessing or rewriting critical sections in other languages to overcome GIL-related performance issues. Development overhead and complexity increases by the size of project.

PYTHON 3.13

Python 3.13 was released on October 7, 2024, and it introduced significant new features and optimizations. The most notable changes include a revamped interactive interpreter, an experimental Just-In-Time (JIT) compiler, and an experimental free-threaded mode, which we will explore further.

FREE-THREADED MODE

Free-threaded mode addresses one of the biggest Python‘s limitations by disabling GIL, therefore, enabling threads to execute in parallel across multiple CPU cores, fully utilising multi-core processors. This feature is particularly advantageous for workloads that require multi-threading, such as simulations, data analysis, and other compute-intensive operations. With the enablement of true parallelism, significant performance improvements can be achieved, particularly in scenarios designed with threading in mind.

RUN WITHOUT GIL

To use free-threaded mode, developers must run their applications with a specialized executable, called python3.13t (or python3.13t.exe). Pre-built binaries for free-threaded mode are available in official Windows and macOS installers. Alternatively, developers can also build CPython from source with the *--disable-gil* option for customized configurations.

DEMONSTRATION