Nayeon Shin

Professor Lerner

Independent Study

16 December 2023

Comparative Analysis of Concurrency in C and Python

Concurrency is the ability of a program to execute multiple independent tasks or processes at the same time. It boosts performance for handling diverse operations in multi-processor or multi-core systems. The two fundamental methods for concurrency are multithreading and multiprocessing. In C programming language, developers achieve fast concurrency using the POSIX threads API for Unix-like operating systems and the Windows API for the Windows operating system. On the other hand, Python provides the multiprocessing and multithreading modules utilized across various operating systems. Despite Python being primarily written in C, it exhibits suboptimal execution speed in multithreading. This performance lag is attributable to Python's Global Interpreter Lock (GIL), a mutex that prevents the concurrent execution of multiple threads. This study assesses the impact of GIL on multithreading in Python by comparing its performance with C.

Hardware & Software Specifications

The hardware under research is a Dell Latitude 5490 notebook featuring an Intel Core i5-8350U CPU running at a base frequency of 1.70GHz. The notebook has 16 gigabytes of DDR4 Synchronous Unbuffered system memory, comprising a single SODIMM module

operating at 2400MHz. The Intel Corp. CPU is a 64-bit processor with four cores, enabling eight threads.

The software environment is a Ubuntu Server 22.04.2 distribution. The system utilizes the Cubic 2023-05-30 09:42 configuration. Furthermore, the technology stack includes Python 3.10.6 and multithreaded Python without the Global Interpreter Lock. Unfortunately, the build configuration (--disable-gil) has not yet landed for CPython 3.12 to let it run Python code without the GIL and with the necessary changes to make the interpreter thread-safe. Thus, this investigation instead leverages a GitHub repository that provides an alternative implementation of Python without the GIL, based on an experimental rebase of the same version of Python.

Implementation of Concurrent Programs

The study presents two programs, one in C and the other in Python, designed to evaluate execution times in a shared scenario using multithreading and multiprocessing. Each program runs ten iterations, where two processes and two threads independently calculate the sum of a given range of numbers. The elapsed times for both multiprocessing and multithreading are recorded and displayed using clock_gettime() and time.time() functions. The comprehensive source code and further information are available in the GitHub repository: c-vs-python-concurrency.

Results

Language - Operation	Average Total Elapsed Time (in seconds)
C - Multiprocessing	1.50
C - Multithreading	1.50
Python - Multiprocessing	6.89
Python - Multithreading	12.83
Python (GIL disabled) - Multiprocessing	8.70
Python (GIL disabled) - Multithreading	8.62

In the C implementation, both multiprocessing and multithreading operations consistently show average elapsed times of 1.50 seconds. This uniformity in performance demonstrates C's efficiency in handling parallel tasks, reflecting its low-level system resource management capabilities.

Conversely, Python exhibits longer total elapsed times for both multiprocessing and multithreading compared to C. When the GIL is enabled, multiprocessing operations in Python take an average of 6.89 seconds, indicating a notable performance lag compared to C. Multithreading in Python, under the influence of the GIL, shows an even more pronounced delay, with an average elapsed time of 12.83 seconds. This significant increase can be attributed to the GIL's limitation on executing multiple threads concurrently, a critical factor in multithreaded, CPU-bound tasks.

When the GIL is disabled, the average elapsed time for multiprocessing operations increases slightly to 8.70 seconds. In contrast, the performance of multithreading in Python improves significantly, recording an average elapsed time of 8.62 seconds. This reduction from

12.83 seconds (with GIL enabled) highlights the GIL's restrictive impact on Python's multithreading capabilities. The absence of the GIL allows for truly parallel execution of threads, enhancing the overall efficiency of multithreaded processes.

Global Interpreter Lock (GIL)

The Global Interpreter Lock (GIL) is a mutex mechanism integral to CPython, the reference implementation of Python. Introduced as a pragmatic solution to concurrency management, the GIL ensures that only one thread executes Python bytecodes at any given time. This approach was adopted to simplify the handling of concurrent operations and to cater to the non-thread-safe nature of Python's memory management. When CPython was first developed in 1989, this design decision was more tenable, primarily due to the prevalent computing environment characterized by single-core processors. In such a case, the GIL's restriction of sequential bytecode execution did not significantly impede performance and offered a more straightforward approach to handling shared resources, such as memory. It obviated the need for complex locking mechanisms for memory access, thereby simplifying the development and maintenance of the Python interpreter.

However, the evolution of computing hardware, particularly the shift towards multi-core processors, has exposed the limitations of the GIL in a multithreading context. In modern multi-core environments, the ability to execute multiple threads concurrently is essential to maximize computational efficiency. The GIL, by allowing only one thread to execute at a time, becomes a bottleneck in such scenarios, especially for CPU-bound tasks. This limitation is less pronounced in I/O-bound tasks, where the GIL releases control during I/O operations, allowing other threads to run. Nonetheless, for compute-intensive applications, the GIL's presence means

that multithreading in Python cannot leverage the full potential of modern multicore processors. This constraint significantly hinders the performance of Python applications designed to benefit from parallelism at the CPU level.

Efforts to address the GIL issue in Python have been ongoing, yet they encounter considerable complexities. The challenge lies in the deep integration of the GIL within Python's architecture and its impact on the broader ecosystem. Removing or modifying the GIL is not merely a technical adjustment but a change that affects the fundamental behavior of Python programs. Many existing libraries and features rely on the GIL for thread safety and consistency. A transition away from the GIL would necessitate a comprehensive review and potential redesign of these components to ensure they operate correctly in a GIL-free environment. Furthermore, such a change must consider backward compatibility, as Python's extensive user base has a vast amount of existing code that assumes the presence of the GIL. The Python community has provided other approaches, including alternative Python implementations like Jython and IronPython, which do not have a GIL. However, these implementations do not offer a direct solution for CPython.

Conclusion

In contrast to the C programming language, Python faces a limitation in achieving effective multithreading due to the presence of the Global Interpreter Lock (GIL). The GIL is a mutex that safeguards access to Python objects, restricting multiple threads from executing Python bytecodes simultaneously. This single-threaded execution hinders true parallelism in multithreading. The findings from this study's experiments illustrate that enhancing parallelism in multithreading is achievable by disabling the GIL, albeit with a slight compromise in the

execution time of individual processes. The design of CPython, the official Python implementation, initially tackled memory management issues with prevalent single-core CPUs. However, it presents a significant drawback in the context of today's ubiquitous multi-core CPUs. Despite the challenges in eliminating the GIL from the Python ecosystem, doing so would enable the realization of true parallelism in multithreading.

Works Cited

"GlobalInterpreterLock - Python Wiki." n.d. Wiki.python.org. https://wiki.python.org/moin/GlobalInterpreterLock.

Wikipedia Contributors. 2019. "Python (Programming Language)." Wikipedia. Wikimedia Foundation. May 4, 2019. https://en.wikipedia.org/wiki/Python_(programming_language).