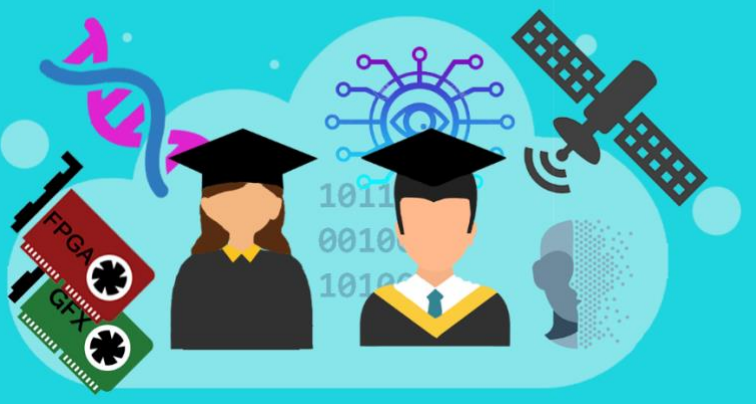
Diploma Thesis

Microproccessors and Digital Systems Laboratory



Dynamic Memory Allocation of Python Programming Language

The Python programming language has become one of the most widely used languages in modern software development, owing to its simplicity, readability, and extensive ecosystem of libraries. However, this convenience often comes at the cost of runtime performance and memory efficiency. Python's internal implementation, primarily in C/C++ (as in the CPython interpreter), abstracts away low-level memory management to provide ease of use and portability. While this design choice enhances developer productivity, it also introduces significant overheads in memory allocation and garbage collection, making Python unsuitable for low-resource or memory-constrained systems such as embedded devices or edge computing environments. Internally, Python relies on a layered allocation strategy involving object-level memory pools**,** reference counting, and a cyclic garbage collector, all of which contribute to nontrivial memory fragmentation and allocation latency. These mechanisms, though robust and general-purpose, are not optimized for specific workloads or architectures. Furthermore, the lack of fine-grained control over memory allocation and deallocation limits opportunities for customized optimizations, such as adaptive allocation policies, cache-aware data structures, or lightweight garbage collection schemes.

This thesis aims to optimize Python's core memory management subsystem, focusing on redesigning or tuning the internal data structures and algorithms responsible for object allocation, reference tracking, and memory reclamation. The goal is to improve memory efficiency, allocation performance, and scalability without compromising Python's high-level semantics or portability. Through detailed profiling, kernel-level instrumentation, and prototype modifications of the CPython memory allocator, this work seeks to provide a foundation for a more resource-efficient Python runtime, capable of operating effectively even in low-memory or heterogeneous system environments.

TOPICS AVAILABLE: 1

PREREQUISITES:

- C/C++, Python, Bash/Shell Scripting
- Assembly, Computer Architecture
- Linux OS, Data Structures, Dynamic Memory Allocation

CONTACT INFORMATION:

Dr. Manolis Katsaragakis, Post-Doc Researcher, NTUA: mkatsaragakis@microlab.ntua.gr

Prof. Dimitrios Soudris, NTUA: dsoudris@microlab.ntua.gr