Chapter 3: Python on the GPU

Compiled vs. Interpreted Computation

Python is one of the most popular programming languages for science, engineering, data analytics, and deep learning applications. However, as an interpreted language, it's been considered too slow for high-performance computing.

Programs compiled into machine code have a speed advantage over interpreted languages, as there is no intermediary step required before instructions execute on the processor.

Additionally, Python threads do not behave like threads in the operating system due to the face that only one thread can execute in the interpreter at a time. The Global Interpreter Lock (GIL), in essence, ensures that threads cannot operate in parallel and causes threads in Python to perform like a single-core CPU. Often, Python programmers have to rely on concurrent programs to take advantage of parallelism, as opposed to multithreading.

Many external libraries in Python have been implemented with other languages like C or C++ to take advantage of multithreading. One of these libraries is NumPy, a library that serves as a building block for numerical computation in Python.

Python Scientific Library Landscape

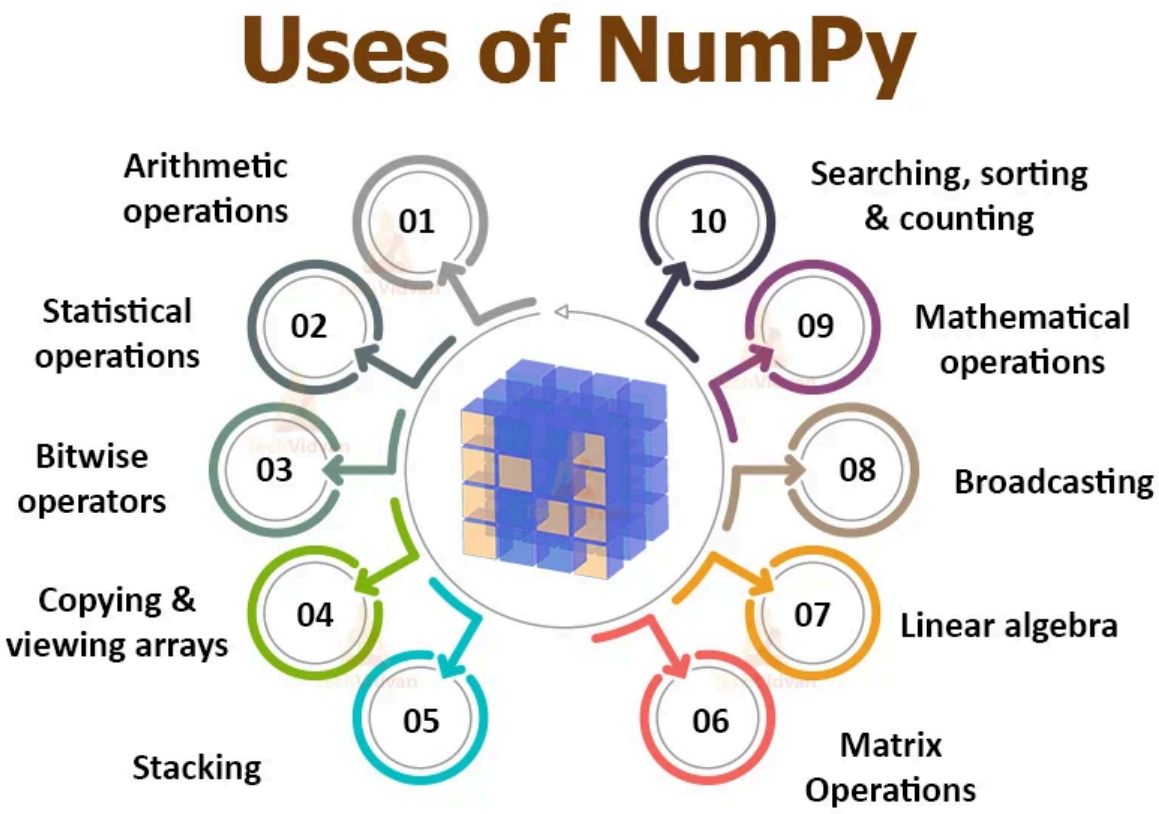
NumPy in the Python Scientific Computing Ecosystem

NumPy (short for Numerical Python) was created in 2005 by merging Numarray into Numeric. Since then, the open source NumPy library has evolved into an essential library for scientific computing in Python. It has become a building block of many other scientific libraries, such as SciPy, Scikit-learn, Pandas, and others. What makes NumPy so incredibly attractive to the scientific community is that it provides a convenient Python interface for working with multi-dimensional array data structures efficiently; the NumPy array data structure is also called `ndarray`, which is short for n-dimensional array.

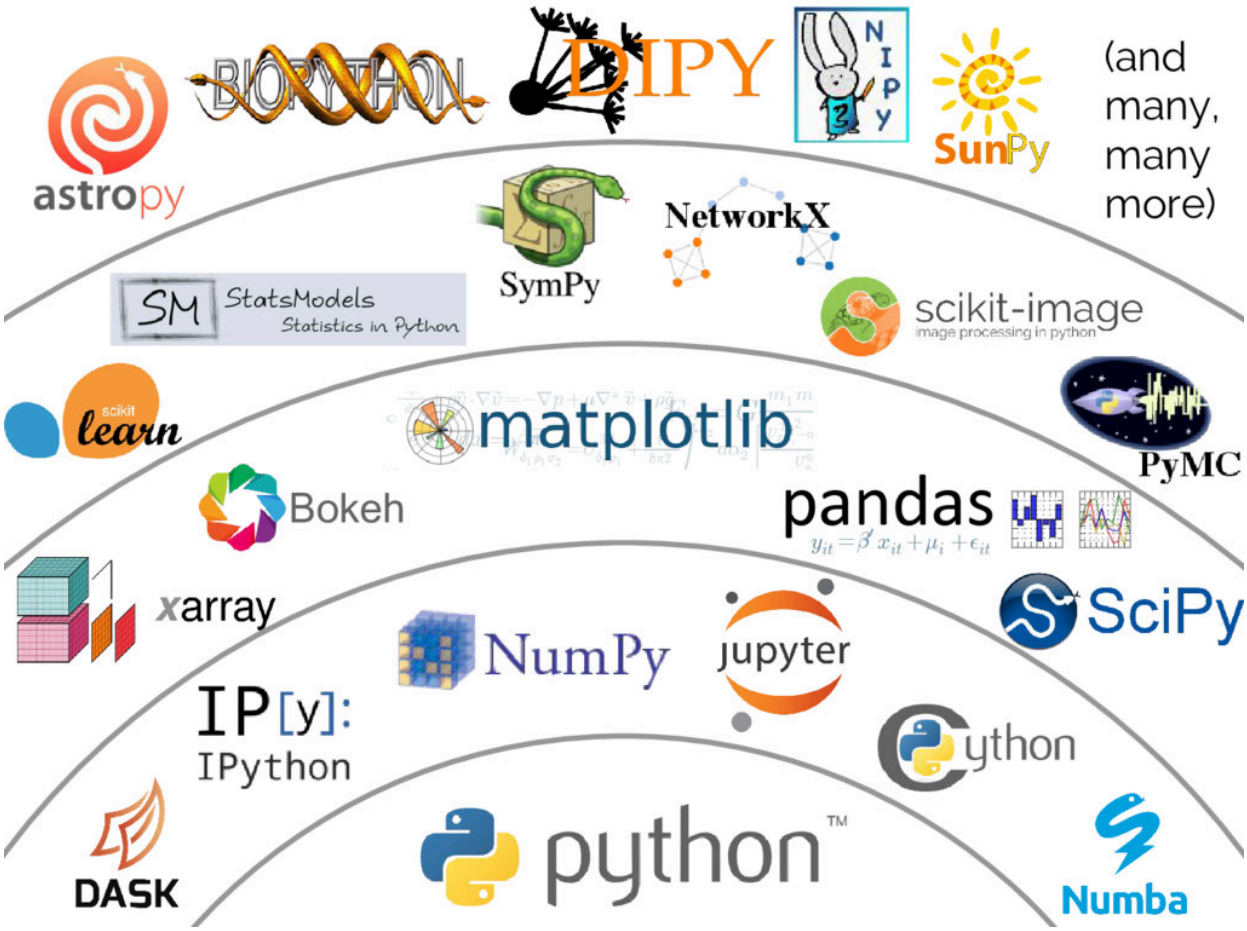
In addition to being mostly implemented in C and using Python as a "glue language," the main reason why NumPy is so efficient for numerical computations is that NumPy arrays use contiguous blocks of memory that can be efficiently cached by the CPU. In contrast, Python lists are arrays of pointers to objects in random locations in memory, which cannot be easily cached and come with a more expensive memory-look-up. However, the computational efficiency and low-memory footprint come at a cost: NumPy arrays have a fixed size and are homogeneous, which means that all elements must have the same type. Homogenous `ndarray` objects have the advantage that NumPy can carry out operations using efficient C code and avoid expensive type checks and other overheads of the Python API. While adding and removing elements from

the end of a Python list is very efficient, altering the size of a NumPy array is very expensive since it requires to create a new array and carry over the contents of the old array that we want to expand or shrink.

Besides being more efficient for numerical computations than native Python code, NumPy can also be more elegant and readable due to vectorized operations and broadcasting, which are features that we will explore in this article.



Today, NumPy helps forms the basis of the scientific Python computing ecosystem. (see numpy.org) There are already GPU accelerated versions of these fundamental libraries being developed, but there is an opportunity for all scientific libraries to take advantage of these optimizations.



Using CuPy in Python Scientific Computing Libraries

The `cupy.ndarray` class is at the core of CuPy and is a replacement class for NumPy's `numpy.ndarray`, making it an excellent entry point into the Python CUDA ecosystem, especially for scientific library developers who have built their software with this dependency on NumPy.

The goal of the CuPy project is to provide Python users GPU acceleration capabilities, without the in-depth knowledge of underlying GPU technologies. The CuPy team focuses on providing:

- A complete NumPy and SciPy API coverage to become a full drop-in replacement, as well as advanced CUDA features to maximize the performance.
- Mature and quality library as a fundamental package for all projects needing acceleration, from a lab environment to a large-scale cluster.

CuPy is built on top of CUDA and provides a higher level of abstraction, making it easier to port code between different GPU architectures and versions of CUDA. This means that CuPy code

can potentially run on different CUDA-compatible systems without requiring significant modifications.

Under the hood, CuPy utilizes CUDA Toolkit libraries including cuBLAS, cuRAND, cuSOLVER, cuSPARSE, cuFFT, cuDNN and NCCL to make full use of the GPU architecture.

Resources

NumPy https://numpy.org/
CuPy https://cupy.dev/
Differences between NumPy and CuPy
https://docs.cupy.dev/en/stable/user_guide/difference.html