# High Performance Python

January 31, 2023

## Contents



# 1 Performant Python

## 1.1 Computer System

1. **Computational Uinit**
   - Key properties are **IPC** (instruction per cycle) and **clock speed** (number of cycles per second).
   - **GIL (Global Interpreter Lock)** makes sure that a python process can run only one instruction at a time regardless of number of cores it is currently using. Can be avoided by multiprocessing, numpy, Cython or distributed models of computing.
2. **Memory Unit**
   In terms of read/write speeds and latency, Spinning hard drive < Solid-state hard drive < RAM < L1/L2 cache
3. **Communication Layers**
   - Bus can only move contiguous chuncks of memory

## 1.2 Python Performance Dragger

1. **Not compiled** hence missing compiler tricks from source code to machine code
2. **Dynamic data type** more overhead before computation
3. **Garbage Collected Laguange > memory fragmentation**: Python objects are not laid out in the most optimal way in memory.
4. **GIL**: Only one instruction at a time

# 2 Python Data Type

## 2.1 List and & Tuple

- Array data structure. Fast in retriving item by location (O(1)). Reasonable speed in look up with bisect search (O(log(n)))
- **Dynamic data type**: Elements can be mixed data types. Library module **array** can reduce dynamic data type overheads for non-numerical situations. Library module **numpy** can help for numerical situations.
- **List** are dynamic arrays that are **mutable** and **allow for resizing**. List will reserve additional memory space for potential append hence less memory efficient. A new list will be created once memory reserved for the list is not sufficent for further append.
- **Tuples** are static arrays that are **immutable** and can not be changed after creation. Tuples are cached by Python runtime hence much faster.
- Library module **array** can be reduce dynamic data type overheads for nonnumerica situations

## 2.2 Dictionary & Set

- Dictionaries are data types efficient in lookup (O(1)) due to hashable functions. Sets are dictionaries with only keys
- Sets are dictionaries with only keys. Sets are efficient in keep unique values (appending O(1) with set.add())
- **Hash Function** transform keys to index for efficient lookup
- **Name Space**: python use dictionaries to hold all objects. Local > Enclosing > Global > Built-in

## 2.3 Iterator & Generator

- Generator can be used to process data larger than RAM

# 3 Numerical calculation with python & pandas

## 3.1 Native Python List

- **Memory segmentation**: Python lists store pointers to actual data. Actual data are spreaded in the memory (might not be continuous). Fragmentation increases cost to transfer data from memory to CPU.
- Vectorization can occur only when CPU cache with all the relevant data. Python bytecode is not optimized for vectorization so for loops can not predict when using vectorization would be beneficial.
  1. All relevant data in CPU cache
  2. Invoke SIMU instruction for calculation

## 3.2 Numpy

- **Numpy** store data in contiguous chunks of memory and supports vectorized operation on its data.
- Static data types so less overhead
- **In place Operation** can help to reduce number of allocations we make in code. In place operations such as +=, *= can help to skip the step of allocating memory to new objects
- Numpy's optimization of vector operation occur only one operation at a time. For operation A * B + C, numpy first evaluate and store A * B in a temporary vector then add this new vector to C.

- **numexpr** is a module that can take an entire vector expression and compile it into efficient code. **numexpr** is using multile core for calculation. When matrix is small it is does not compensate enough for overhead on multiple cores. While when the matrix is very large, it can help to speed up.

### 3.3 Pandas

- Columns opf the same dtype are grouped together hence row-wise operations on columns of the same datatype faster.
- Numeric columns directly reference their Numpy data; String columns reference a list of python strings, which are scattered in memory
- Avoid concate as it creates a new section of memory
- avoid inplace=True operator
- use del keyword and drop function to delete un-used references
- include numexpr and bottleneck module
- For large series of strings with low cardinality, convert it to category type

## 4 multiprocessing

The multiprocessing module provides process and thread based parallel processing, share work over queues and share data among process