DD2358 – Introduction to the Course

Stefano Markidis

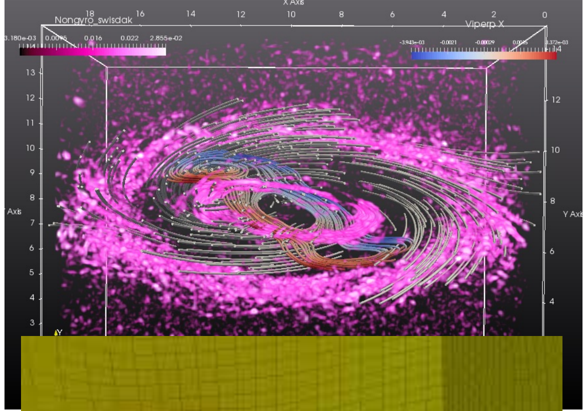
KTH Royal Institute of Technology

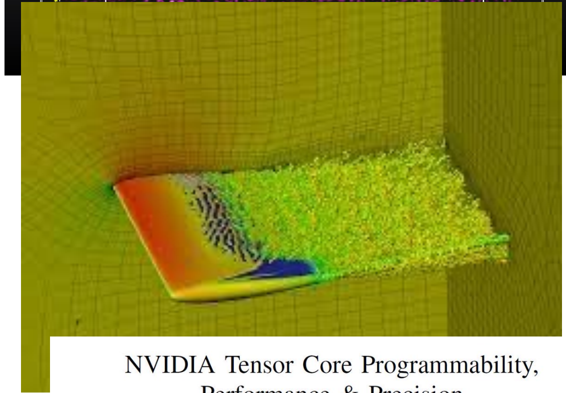
DD2358 Course Responsible, Teacher & Examiner

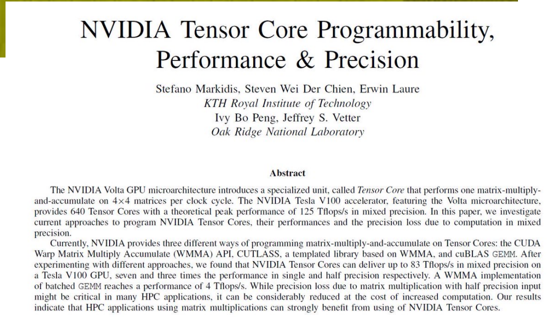
- Stefano Markidis
- Associate Professor in High-Performance Computing
  - markidis@kth.se
  - Best if you post questions related to the course and topics on the **Canvas discussion pages**

HPC Research

- Designing algorithms and methods to develop **scientific applications for high-performance computing systems**, such as supercomputers
  > *Computer Architecture for HPC: GPUs, FPGAs, …*
  > *Programming Models: MPI, OpenMP*
  > *Scientific Computing: Computational Fluid Dynamics, Computational Plasma Physics, Deep-Learning*







NVIDIA Tensor Core Programmability, Performance & Precision

Stefano Markidis, Steven Wei Der Chien, Erwin Laure
KTH Royal Institute of Technology
Ivy Bo Peng, Jeffrey S. Vetter
Oak Ridge National Laboratory

Abstract

The NVIDIA Volta GPU microarchitecture introduces a specialized unit, called *Tensor Core* that performs one matrix-multiply-and-accumulate on 4×4 matrices per clock cycle. The NVIDIA Tesla V100 accelerator, featuring the Volta microarchitecture, provides 640 Tensor Cores with a theoretical peak performance of 125 Tflops/s in mixed precision. In this paper, we investigate current approaches to program NVIDIA Tensor Cores, their performances and the precision loss due to computation in mixed precision.

Currently, NVIDIA provides three different ways of programming matrix-multiply-and-accumulate on Tensor Cores: the CUDA Warp Matrix Multiply Accumulate (WMMA) API, CUTLASS, a templated library based on WMMA, and cuBLAS GEMM. After experimenting with different approaches, we found that NVIDIA Tensor Cores can deliver up to 83 Tflops/s in mixed precision on a Tesla V100 GPU, seven and three times the performance in single and half precision respectively. A WMMA implementation of batched GEMM reaches a performance of 4 Tflops/s. While precision loss due to matrix multiplication with half precision input might be critical in many HPC applications, it can be considerably reduced at the cost of increased computation. Our results indicate that HPC applications using matrix multiplications can strongly benefit from using of NVIDIA Tensor Cores.

MS Courses at KTH

HPG Group is responsible for the HPC MS courses and **HPC MS sub-Track**:

1. ***DD2358* – *Introduction to High-Performance Computing***
2. ***DD2356*** – *Methods in High-Performance Computing – Continuation Course*
   - Focus on performance modeling & parallel computing – C and Fortran
3. ***DD2360*** – *Applied GPU Programming - Focus on programming accelerators*
4. ***DD2375*** – *Project course in High-Performance Computing*
5. ***DD2367*** – *Quantum Computing for Computer Scientists*
6. ***DD2370*** – *Computational Methods for Electromagnetics*
7. ***MS Theses*** *in the area of HPC and scientific computing*

DD2358 – First Course in the HPC MS Track

- The goal of this course is to introduce you the methodologies, approaches and tools that are necessary for professionals in HPC and software development for Scientific Computing
  - **Examples:** profilers, BLAS, perf, Visit, Paraview, ...
- This part is often missed by regular courses in HPC and assumed by the following courses, such as DD2356 and DD2360.
  - The goal is to fill this gap with DD2358

DD2358 – Python as Main Language of the Course

- Python emerged as one of the most prominent languages for developing scientific applications.
  - We will take Python to explain many HPC concepts without losing generality
  - I will try to point out tools also used in other languages, such as profilers, tools, and libraries…
  - Coding assignments are to be done in Python.
- We will provide a brief overview of the language basics, but we assume that you have been exposed before

Prerequisites

- Basic knowledge of Python
- Basic knowledge of C or Fortran (we will couple Python to C/Fortran in third module)
  - We will post a few points about these languages
- Knowledge of Linux command line for running, compiling, and using online editors (emacs, vi, …)

Needed Resources for the Course

- Your laptop will be enough to complete the coding assignments
- The best would be to have a Linux machine, but windows and mac also work
  - On Windows systems, you will need to install the **Ubuntu command line**.
- Possible to use Google’s Colab.
- We will use one tool, called *perf*, available only on Linux (not on Mac, Windows, and Colab)
  - For Linux machines:
    - > *Computer labs at KTH (Grön, Röd, …)*
    - > *Connect to Linux machines provided by KTH*
    - > *Virtual machines*

Intended Learning Outcomes

At the end of the course, you will be able to:

1. Describe and list the **different computer components** that are vital
   to achieve high performance for serial applications
2. **Develop** and **program**, and **optimize** the performance of a scientific application
3. Use efficiently several **software engineering tools**, **modules** and **libraries** for the **development** and **optimization** of scientific applications
4. Develop and optimize a scientific application from scratch following the **best practices** and more **convenient HPC tools**

What is HPC?

- How can we design a code that is as fast as possible on a given computer system?
  – Know the hardware and map the algorithm to the hardware
    1. *We need to know the hardware*
    2. *We need to know the parameters that characterize the performance*
    3. *We need to know what it is maximum performance achievable on a system*
    4. *We need to be able to measure how efficiently the hardware is used!*

HPC & Scientific Computing Today: Working in Teams

- Scientific applications used to be the result of heroic effort of one or maximum two scientists writing the all code
- Today, we have several people – often with different competences – in a team working together on different parts of the code at the same time
- It is critical that all the members know how to collaborate virtually and write code together
  - > *Version control / Git*
  - > *Unit test*
  - > *Documentation*
  - > *Deployment*
  - > …
- Tutorial on tools

Course Organization – 4 Modules – Roughly 2 weeks each

1. Fundamentals of Computer Systems & Profiling Codes

2. Data structures and methods for HPC

3. Compilation Techniques for Performance Optimization

4. Concurrency and Parallelism

Course Activities

1. Pre-recorded video lectures for four modules
2. 4 Individual Assignments
    1. *You can work in groups of 2-3, but the report text should be different, and the submission should state who you worked with.*
3. Individual peer review of the assignments
4. In-person class to introduce the modules and discuss assignments
    > *Not compulsory*
5. Suggested readings
6. Individual Final Project

Individual Assignments & Peer-Review

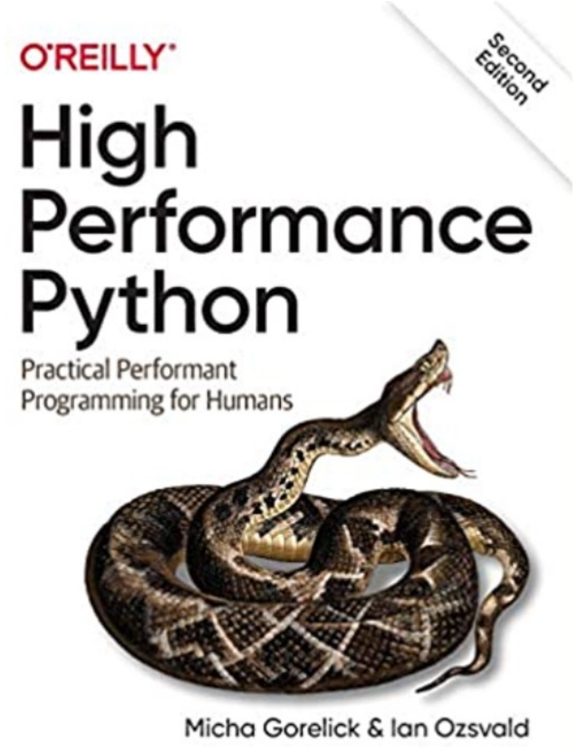
- 4 P/F assignments with bonus exercises to increase the final grade
  - You can work in groups of 2-3, but the report text should be different, and the submission should state who you worked with.
- Individual peer-review of assignment
  - Peer-review of assignment: 2 strengths + 2 weaknesses + checking submitted code
  - To be submitted as a comment in Canvas.

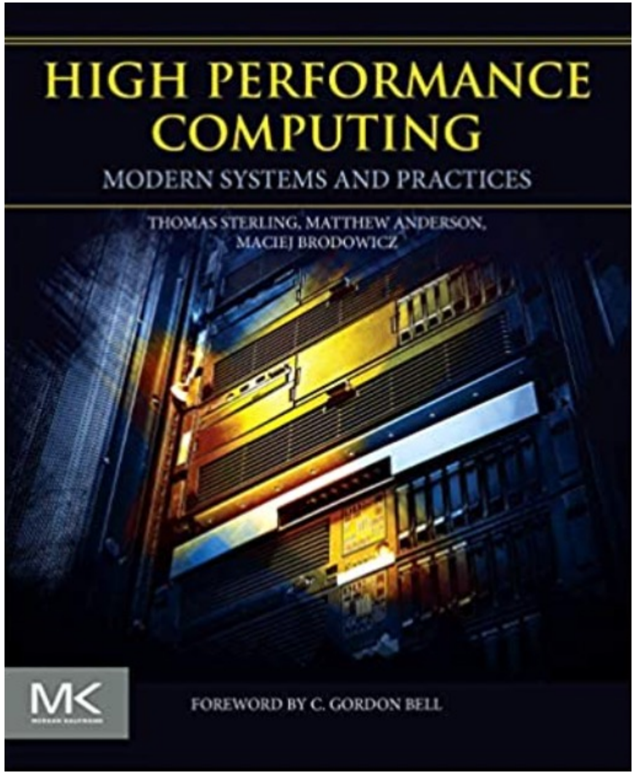
To Pass this Course

- Submit and complete 4 P/F Individual Assignments successfully.
  - If you complete the bonus exercise, you will receive "+ 1” toward the final grade.
  - If you submit and successfully complete the 4 assignments without bonus exercises, you pass the course with a grade of E
  - With all bonus exercises, your grade is B
- Prepare 4 **peer-review**
- To obtain A, you need to work individually on a final project and **submit a report** (maximum 8 pages)
- The final project is <u>individual.</u>
  - The topic of the project is fixed and published two weeks before the end of the course.
  - The submission and presentation of the report project are scheduled for the last day of the exam period.

Course Material & Reference Textbooks

- We will provide all the slides on Canvas.
- We will also provide links to book chapters we are using for preparing the lectures and/or to go deeper on the topic.
- All the books we use in the course are available through the KTH library online.

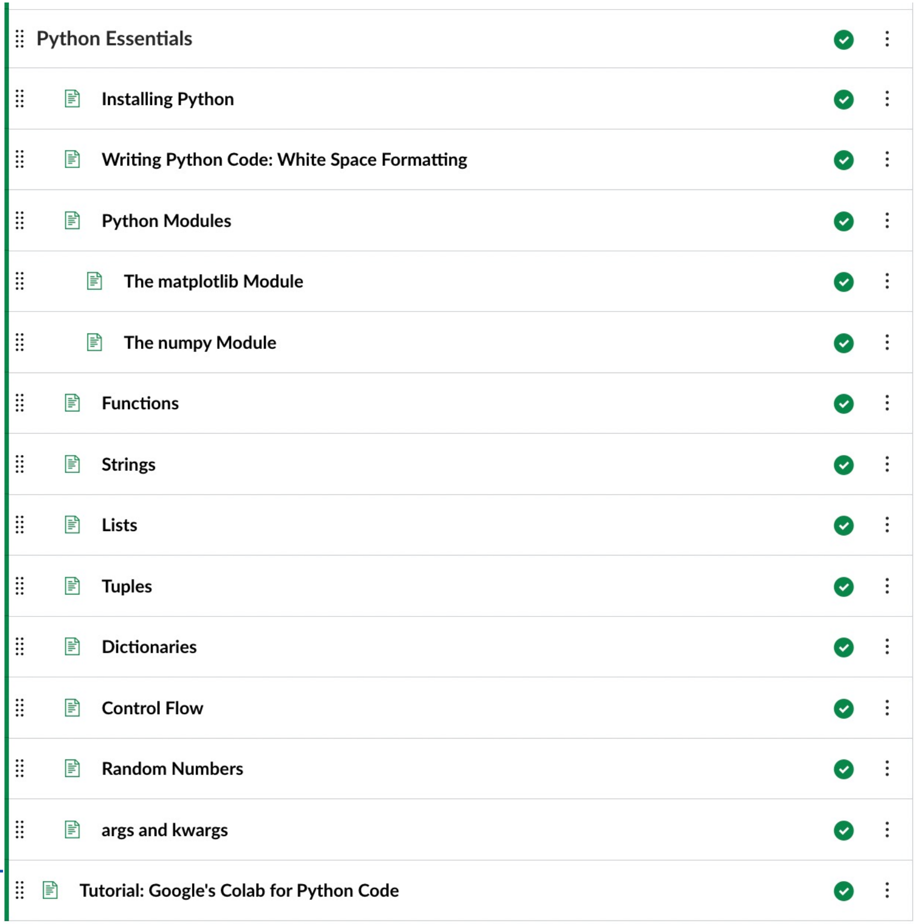

O'REILLY
Second Edition
High Performance Python
Practical Performant Programming for Humans
Micha Gorelick & Ian Ozsvald


HIGH PERFORMANCE COMPUTING
MODERN SYSTEMS AND PRACTICES
THOMAS STERLING, MATTHEW ANDERSON, MACIEJ BRODOWICZ
MK
FOREWORD BY C. GORDON BELL

Getting Help in the Course

- Use the Canvas discussion section to post questions and issues about the topic of the course
- Each module has a discussion page
  - Better Canvas than email so all the instructors and students can see the issue and reply quickly.

Step 0: Refresh / Get the Basics of Python

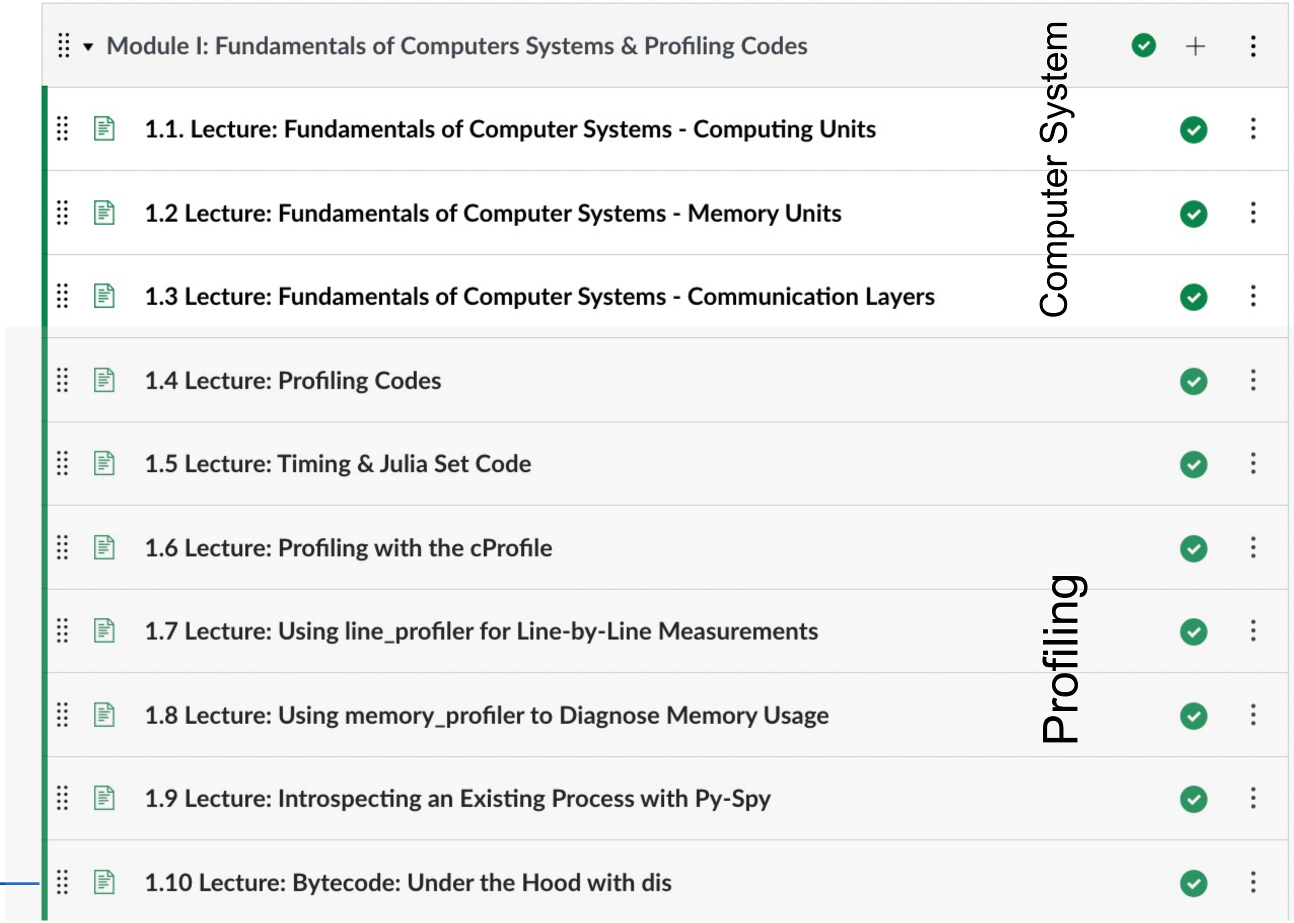


Python Essentials ✓

- Installing Python ✓
- Writing Python Code: White Space Formatting ✓
- Python Modules ✓
  - The matplotlib Module ✓
  - The numpy Module ✓
- Functions ✓
- Strings ✓
- Lists ✓
- Tuples ✓
- Dictionaries ✓
- Control Flow ✓
- Random Numbers ✓
- args and kwargs ✓

Tutorial: Google's Colab for Python Code ✓

Module I: Fundamentals of Computers Systems & Profiling Codes



Module I: Fundamentals of Computers Systems & Profiling Codes

1.1. Lecture: Fundamentals of Computer Systems - Computing Units

1.2 Lecture: Fundamentals of Computer Systems - Memory Units

1.3 Lecture: Fundamentals of Computer Systems - Communication Layers

1.4 Lecture: Profiling Codes

1.5 Lecture: Timing & Julia Set Code

1.6 Lecture: Profiling with the cProfile

1.7 Lecture: Using line_profiler for Line-by-Line Measurements

1.8 Lecture: Using memory_profiler to Diagnose Memory Usage

1.9 Lecture: Introspecting an Existing Process with Py-Spy

1.10 Lecture: Bytecode: Under the Hood with dis

Fundamentals of Computer Systems

- To be able to write efficient code, we need to know the target hardware we want to run.
  - CPU, kind memory, size of the memory, memory caches, … impact on the performance
- Understand the current state and the future of computer systems
  - HPC laws: Moore’s, Dennard’s, Amhdal’s laws …
- Understand metrics to express performance
  - FLOPS/s, IPC, bandwidth, latency, …

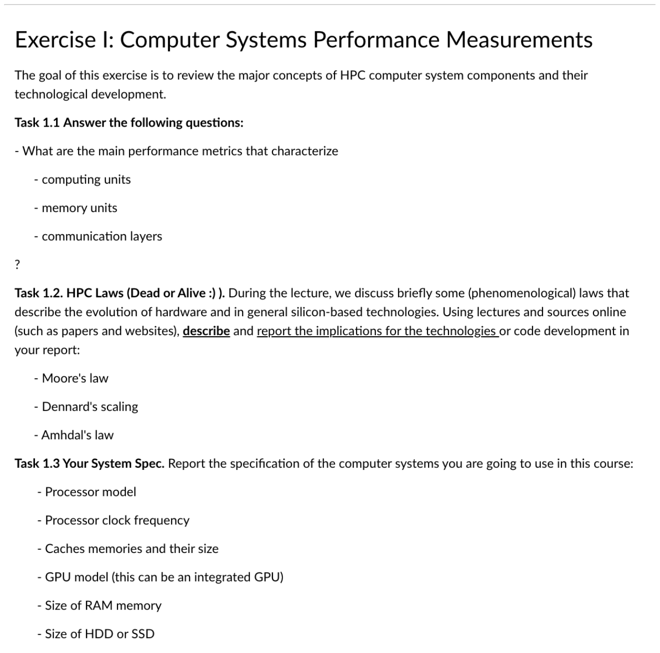
Profiling a Code = Understand the Performance of a Code using Tools

- Simple Approach: Timing
  - What is the precision of our timers?
  - How do I report timing in my report? Average and error (standard deviation
- Where is the performance bottleneck of my code?
  - How do I find this information and how I visualize it?
- How well is my hardware used?
  - Memory usage, …
- How I report about performance in my reports, read https://canvas.kth.se/courses/37558/modules/items/607430

HPC Software Engineering

- GIT and Version Control

Assignment I: Due Jan 27, 2023



Exercise I: Computer Systems Performance Measurements

The goal of this exercise is to review the major concepts of HPC computer system components and their technological development.

Task 1.1 Answer the following questions:

- What are the main performance metrics that characterize

  - computing units

  - memory units

  - communication layers

?

Task 1.2. HPC Laws (Dead or Alive :)). During the lecture, we discuss briefly some (phenomenological) laws that describe the evolution of hardware and in general silicon-based technologies. Using lectures and sources online (such as papers and websites), **describe** and report the implications for the technologies or code development in your report:

  - Moore's law

  - Dennard's scaling

  - Amhdal's law

Task 1.3 Your System Spec. Report the specification of the computer systems you are going to use in this course:

  - Processor model

  - Processor clock frequency

  - Caches memories and their size

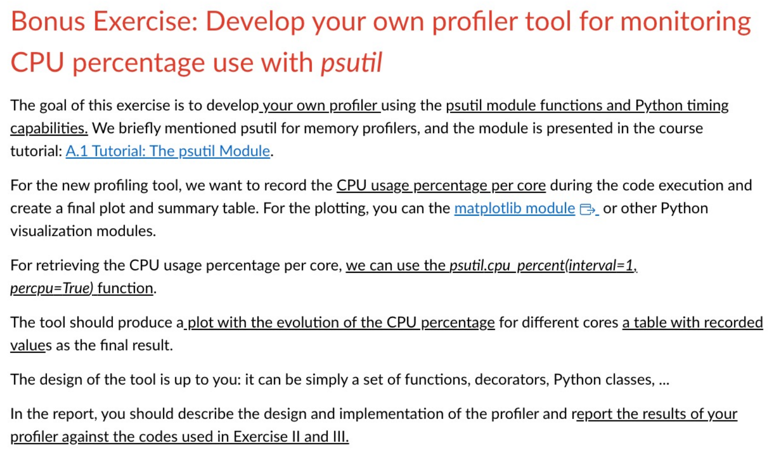
  - GPU model (this can be an integrated GPU)

  - Size of RAM memory

  - Size of HDD or SSD

...

+1



Bonus Exercise: Develop your own profiler tool for monitoring CPU percentage use with *psutil*

The goal of this exercise is to develop your own profiler using the psutil module functions and Python timing capabilities. We briefly mentioned psutil for memory profilers, and the module is presented in the course tutorial: A.1 Tutorial: The psutil Module.

For the new profiling tool, we want to record the CPU usage percentage per core during the code execution and create a final plot and summary table. For the plotting, you can the matplotlib module or other Python visualization modules.

For retrieving the CPU usage percentage per core, we can use the *psutil.cpu_percent(interval=1, percpu=True)* function.

The tool should produce a plot with the evolution of the CPU percentage for different cores a table with recorded values as the final result.

The design of the tool is up to you: it can be simply a set of functions, decorators, Python classes, ...

In the report, you should describe the design and implementation of the profiler and report the results of your profiler against the codes used in Exercise II and III.