CoGrammar

Welcome to this session:

Skills Bootcamp - Tutorial

The session will start shortly...

Questions? Drop them in the chat.
We'll have dedicated moderators answering questions.

Skills Bootcamp Data Science Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. We will be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Skills Bootcamp Data Science Housekeeping

- For all **non-academic questions**, please submit a query: ***www.hyperiondev.com/support***

- **Report a safeguarding incident: *www.hyperiondev.com/safeguardreporting***

- We would love your feedback on lectures: ***Feedback on Lectures.***

- Find all the lecture **content** in your **Lecture Backpack** on GitHub.

- If you are hearing impaired, kindly use your computer's function through Google chrome to enable captions.

CoGrammar

Safeguarding & Welfare

We are committed to all our students and staff feeling safe and happy; we want to make sure there is always someone you can turn to if you are worried about anything.

If you are feeling upset or unsafe, are worried about a friend, student or family member, or you feel like something isn't right, speak to our safeguarding team:

Ian Wyles
Designated Safeguarding Lead

Simone Botes

Nurhaan Snyman

Rafiq Manan

Ronald Munodawafa

Tevin Pitts

Scan to report a safeguarding concern

or email the Designated Safeguarding Lead:
Ian Wyles
safeguarding@hyperiondev.com

CoGrammar HyperionDev

Skills Bootcamp Progression Overview

✅ Criterion 1 - Initial Requirements

Specific achievements within the first two weeks of the program.

To meet this criterion, students need to, by no later than **01 December 2024 (C11)** or **22 December 2024 (C12):**

- **Guided Learning Hours** (GLH): Attend a minimum of 7-8 GLH per week (lectures, workshops, or mentor calls) for a total minimum of **15 GLH**.
- **Task Completion:** Successfully complete the **first 4 of the assigned tasks**.

✅ Criterion 2 - Mid-Course Progress

Progress through the successful completion of tasks within the first half of the program.

To meet this criterion, students should, by no later than **12 January 2025 (C11)** or **02 February 2025 (C12):**

- **Guided Learning Hours** (GLH): Complete at least **60 GLH**.
- **Task Completion :** Successfully complete the **first 13 of the assigned tasks**.

CoGrammar

Skills Bootcamp Progression Overview

✅ Criterion 3 – End-Course Progress

Showcasing students' progress nearing the completion of the course.

To meet this criterion, students should:

- **Guided Learning Hours** (GLH)**:** Complete the **total minimum required GLH,** by the **support end date.**
- **Task Completion : Complete all mandatory tasks**, including any necessary resubmissions, by the end of the bootcamp, **09 March 2025 (C11)** or **30 March 2025 (C12)**.

✅ Criterion 4 - Employability

Demonstrating progress to find employment.

To meet this criterion, students should:

- **Record an Interview Invite:** Students are required to record proof of invitation to an interview by **30 March 2025 (C11)** or **04 May 2025 (C12)**.
  - **South Holland Students** are required to proof and interview by **17 March 2025.**
- **Record a Final Job Outcome :** Within 12 weeks post-graduation, students are required to record a job outcome.

CoGrammar

Learning Outcomes

- **Implement parallel computing in Python** using multiprocessing, joblib, and Dask to accelerate data science workflows.
- **Use GPU acceleration for deep learning and matrix computations** with CUDA and TensorFlow/PyTorch.
- **Utilize distributed computing frameworks** (e.g., Apache Spark) for large-scale data processing.
- **Optimize data pipelines for high-performance execution using profiling tools** such as cProfile and line_profiler.
- **Apply HPC solutions to real-world data science problems.**

Which of the following is a key advantage of using Dask over Pandas?

A. Dask has more built-in statistical functions
B. Dask is optimized for deep learning
C. Dask automatically fixes memory leaks
D. Dask can handle computations across multiple CPUs and machine

CoGrammar

Which of the following is a key advantage of using Dask over Pandas?

A. Dask has more built-in statistical functions
B. Dask is optimized for deep learning
C. Dask automatically fixes memory leaks
D. Dask can handle computations across multiple CPUs and machine

CoGrammar

What is the primary purpose of CUDA in high-performance computing?

A. To store large datasets

B. To run computations on GPUs instead of CPUs

C. To visualize high-dimensional data

D. To replace the need for cloud computing

What is the primary purpose of CUDA in high-performance computing?

A. To store large datasets

B. To run computations on GPUs instead of CPUs

C. To visualize high-dimensional data

D. To replace the need for cloud computing

CoGrammar

When should you use Spark instead of Dask?

A. When dealing with real-time streaming data
B. When handling small datasets
C. When running only CPU-based computations
D. When using only a single computer

CoGrammar

When should you use Spark instead of Dask?

A. **When dealing with real-time streaming data**
B. When handling small datasets
C. When running only CPU-based computations
D. When using only a single computer

HPC in Python

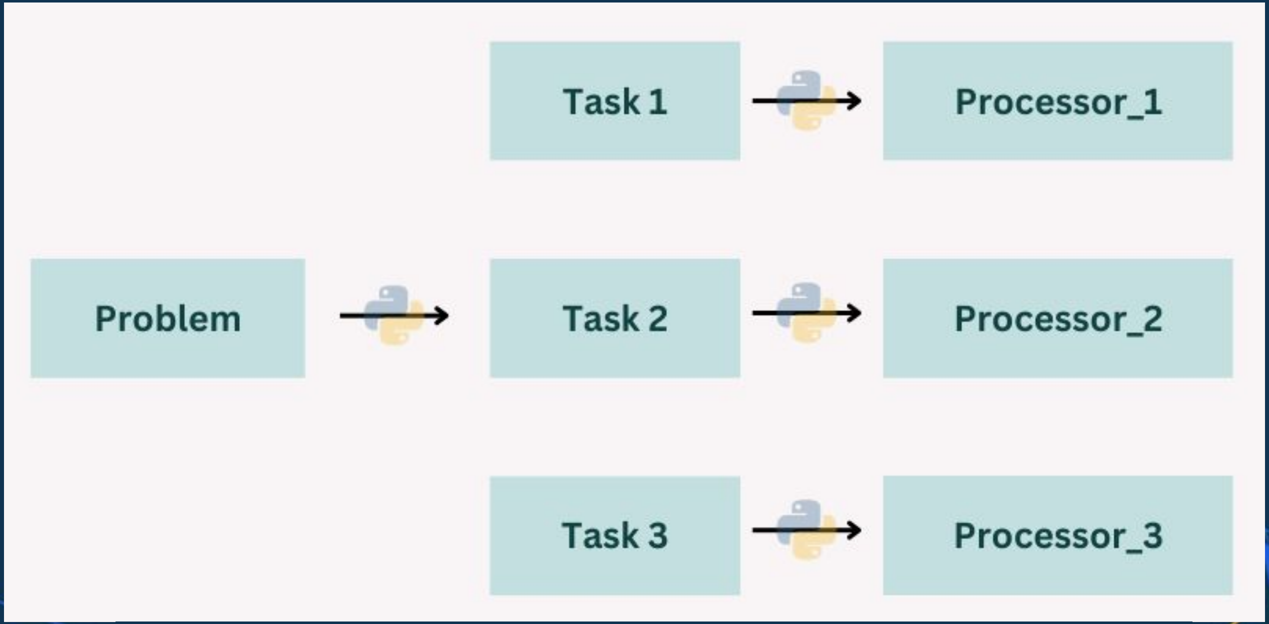
HPC enables faster and more scalable solutions for AI model training, big data analysis, and complex simulations. Understanding parallel computing, GPU acceleration, and distributed processing can help optimize performance in real-world applications.

- What are some Python tasks you've encountered that **took too long** to run? How did you handle them?
- How can we **integrate HPC techniques** into **Python workflows** to handle **large-scale computations efficiently**?

Parallel Computing in Python

- **Parallel computing** runs multiple tasks simultaneously on multiple, closely-connected processors, for example on different CPU cores.
- Best for computationally complex problems.
- Ideal for CPU-bound tasks like data transformations.
- For parallelism, it is important to **divide the problem** into sub-units that **do not depend** on other sub-units (or less dependent).
- If we can divide our problem into sub-units that are **completely independent** of each other, we call it **embarrassingly parallel.**

Parallel Computing in Python



Source: [Guide to Parallel Processing in Python]

Joblib for Parallel Processing

- Joblib is a module in Python especially used to **execute tasks in parallel** using **pipelines**.
- **Joblib** makes it easy to **parallelise computations** across **multiple cores.**

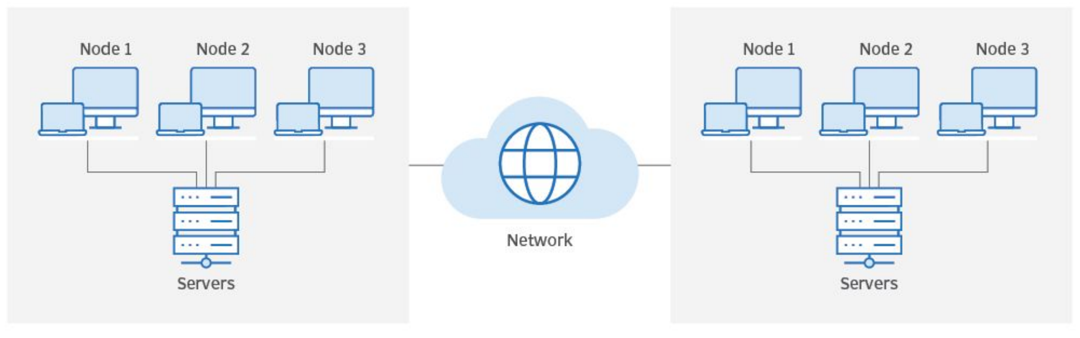
```python
from joblib import Parallel, delayed
import time

def slow_function(x):
    time.sleep(1)
    return x * x

# Run the function in parallel using 4 CPU cores
results = Parallel(verbose=100, n_jobs=4)(delayed(slow_function)(i) for i in range(10))
print(results)
```

Joblib for Parallel Processing

```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done   1 tasks      | elapsed:    1.9s
[Parallel(n_jobs=4)]: Done   2 tasks      | elapsed:    1.9s
[Parallel(n_jobs=4)]: Done   3 tasks      | elapsed:    1.9s
[Parallel(n_jobs=4)]: Done   4 out of  10 | elapsed:    1.9s remaining:    2.9s
[Parallel(n_jobs=4)]: Done   5 out of  10 | elapsed:    2.9s remaining:    2.9s
[Parallel(n_jobs=4)]: Done   6 out of  10 | elapsed:    2.9s remaining:    1.9s
[Parallel(n_jobs=4)]: Done   7 out of  10 | elapsed:    2.9s remaining:    1.2s
[Parallel(n_jobs=4)]: Done   8 out of  10 | elapsed:    2.9s remaining:    0.7s
[Parallel(n_jobs=4)]: Done  10 out of  10 | elapsed:    3.9s finished
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Distributed Computing in Python

- **Distributed computing** connects computers via a network so that they can act as one powerful machine.
- In a distributed system, each device has its own processing capabilities and may also store and manage its own data.
- Suites problems with massive datasets.



Source: [Distributed Computing]

CoGrammar

Distributed Computing in Python

❖ Two of the most popular distributed computing frameworks are:

  - **Dask:** Parallelises **Pandas and Numpy operations** across multiple CPU cores.
  - **Apache Spark:** Optimised for **big data processing** using SQL and **real-time data streaming**. It's written in Scala and offers a Python API called **PySpark**.

```python
import dask.dataframe as dd

# Read a large CSV file using Dask
df = dd.read_csv("file.csv")

# Perform a groupby operation in parallel
result = df.groupby("Age").mean().compute()
print(result)
```

Distributed Computing in Python

	Dask	Apache Spark	pandas
APIs	Reuses pandas APIs	Own APIs	Own APIs
Use Case	Data Science/General	General	Data Science/Analysis
Scalability	High	High	Limited
Nodes	Multiple Nodes	Multiple Nodes	Single Node
Mutability	Immutable	Immutable	Mutable
Work	Complex Work	Lazy Execution	Quick Execution

Source: Consensius

- When we are working with datasets that are are **larger than your RAM**, use **dask**, otherwise **pandas** is the best option. See more here...

Let’s Breathe!

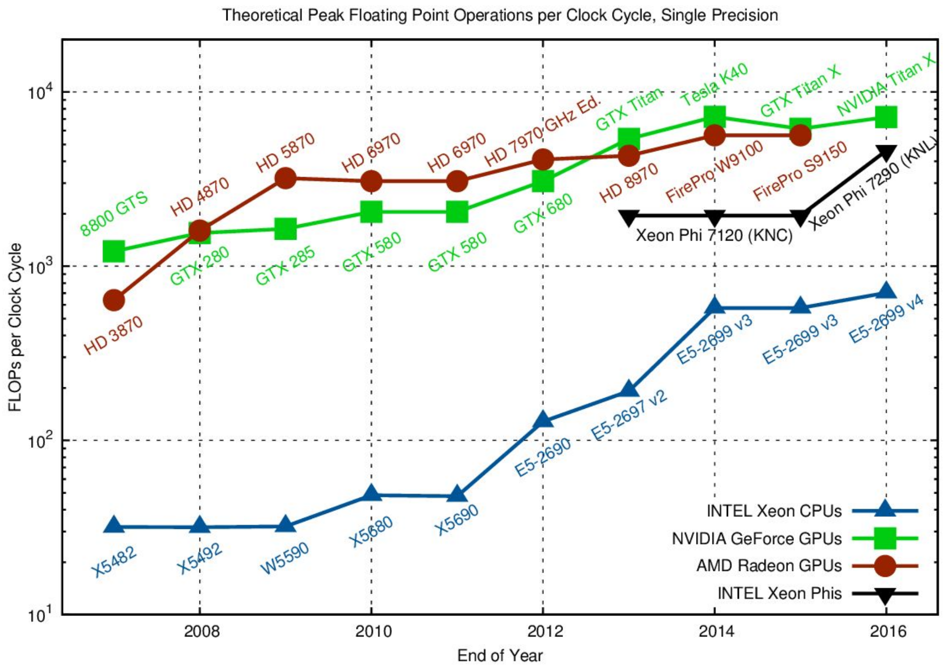
Let’s take a small break before moving on to the next topic.

CoGrammar

GPU Acceleration for Machine Learning

- **GPUs** execute **thousands** of operations **in parallel**, making them ideal for the parallel processing requirements of machine learning algorithms.
- They excel in **matrix operations** and **parallel computations**, which are prevalent in algorithms like **deep learning**.
- GPUs are less suited for tasks where a **wide variety of different operations** are needed simultaneously, as their architecture prioritizes uniformity in operation across many data elements.

CPU vs GPU for ML



Source: [Comparing CPUs and GPUs for Machine Learning]

GPU Acceleration for Machine Learning

- Several Python libraries have been optimized for GPU usage.
- These libraries abstract much of the complexity involved in writing GPU-accelerated code, which is usually done using a platform called **CUDA**.
- **TensorFlow** and **PyTorch** offer the best GPU integration.
- **CuPy** provides an N-dimensional array and mimics NumPy's API but executes operations on a GPU.

CoGrammar

GPU Acceleration for Machine Learning

- To use GPU acceleration, ensure that your environment is setup correctly first and you have installed all the necessary drivers.

```python
import torch

# Check if GPU is available and use it
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

# Create tensors and perform matrix multiplication
a = torch.tensor([[1.0, 2.0], [3.0, 4.0]], device=device)
b = torch.tensor([[5.0, 6.0], [7.0, 8.0]], device=device)
c = torch.matmul(a, b)

print(c)
```

Profiling and Optimising

- Sometimes, the return on investment in performance optimizations just isn't worth the effort.
- **Profiling** helps pinpoint **slow sections** of code, so you can determine whether optimizing the code is necessary.
- Use **cProfile** to analyse **execution time**.
- Use **line_profiler** for **detailed line-by-line profiling.**

CoGrammar

Profiling and Optimising

```python
import cProfile

def example_function():
    total = 0
    for i in range(1000000):
        total += i
    return total

# Profile the execution of the function
cProfile.run('example_function()')
```

```
Ordered by: standard name

ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.000    0.000    0.030    0.030 <string>:1(<module>)
     1    0.030    0.030    0.030    0.030 profiling_c.py:3(example_function)
     1    0.000    0.000    0.030    0.030 {built-in method builtins.exec}
     1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
```

Profiling and Optimising

```python
from line_profiler import LineProfiler

def slow_function():
    total = 0
    for i in range(1000000):
        total += i
    return total

# Create a profiler and add the function
lp = LineProfiler()
lp.add_function(slow_function)

# Run the profiler
lp.enable()
slow_function()
lp.disable()

# Print the profiling results
lp.print_stats()
```

```
Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     3                                           def slow_function():
     4         1       1000.0   1000.0      0.0      total = 0
     5   1000001  117521000.0    117.5     54.1      for i in range(1000000):
     6   1000000   99880000.0     99.9     45.9          total += i
     7         1       2000.0   2000.0      0.0      return total
```

CoGrammar

Which library is best suited for running distributed Pandas-like operations in Python?

A. Dask
B. NumPy
C. Scikit-learn
D. Matplotlib

Which library is best suited for running distributed Pandas-like operations in Python?

A. **Dask**
B. NumPy
C. Scikit-learn
D. Matplotlib

CoGrammar

What is the advantage of using GPUs over CPUs for deep learning?

A. GPUs have more storage

B. GPUs consume less power

C. GPUs are better at handling missing data

D. GPUs can process multiple computations in parallel, reducing training time

CoGrammar

What is the advantage of using GPUs over CPUs for deep learning?

A. GPUs have more storage

B. GPUs consume less power

C. GPUs are better at handling missing data

D. GPUs can process multiple computations in parallel, reducing training time

How can you identify performance bottlenecks in Python?

A. Running code on a single core

B. Increasing dataset size

C. Using cProfile or line_profiler

D. Using print statements

CoGrammar

How can you identify performance bottlenecks in Python?

A. Running code on a single core

B. Increasing dataset size

C. Using cProfile or line_profiler

D. Using print statements

Summary

- ★ Parallel computing speeds up CPU-bound computations.
- ★ Distributed computing (Dask/Spark) enables scalable big data processing.
- ★ GPU acceleration significantly improves deep learning performance.
- ★ Profiling and optimization help identify bottlenecks in data science workflows.

CoGrammar

Q & A SECTION

Please use this time to ask any questions relating to the topic, should you have any.

Thank you for attending

CoGrammar

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education