Distributed Parallel Computing with Python

The aim of this project is to use the different frameworks there exists under Python to expand the task farming exercise from week 3 of the course to work on task farming with and without dependencies.

The project consists of a background part, an implementation part, and an investigation part

Background: make a brief exposition of the principles of distributed parallel computing as expressed in python. What exists, and what is the conceptual ideas behind them.

Implementation: Use a range of multi-processing approaches to

- Re-implement the task farming exercise with the high energy particle physics data [start with re-implementing the code in python without any parallel execution and validate you get it right].
- You have access to a larger range of data packages. For the sake of demonstration, construct a dependency tree for the different data files, and show how this can be processed, with scheduling taking in to account the different dependencies.

Investigation: execute the code(s) on the ERDA MODI cluster, benchmark how well they perform, visualise the workflows, and discuss strategies for bunching processing to improve scalability.

References:

- Material from week 3
- Material from week 6
- Concurrency in Python: https://realpython.com/python-concurrency/
- Dask: https://dask.org
- MPI for python documentation: https://mpi4py.readthedocs.io/en/stable
- More data files: https://sid.erda.dk/sharelink/hLuophqdMK

Prerequisite: Running extra python packages inside a MODI job is a bit involved since the nodes do not see your ERDA mount. Please read the ERDA MODI usersguide chapter 8 for how to set this up.

Running mpi4py or other packages in an interactive session on MODI+ERDA can be done using the HPC notebook as long as you disable the high-performance network (see point 6 below). Here are instructions on how to get it working:

1) Login to MODI on ERDA using the HPC notebook to get access to MPI.

2) Open a new tab with a terminal. You can run interactive `mpi4py` programs by using `mpiexec`.

3) To run on the cluster nodes, you need your program to be stored on `~/modi_mount`. Another important detail is that on the cluster nodes, when using `mpiexec`, the conda environment will not be picked up. To get the correct environment (in this example job script I use the default '`python3`' environment) you must give an explicit path to the python binary. The following job script should work:

```
#!/usr/bin/env bash
```

```
#SBATCH --job-name=mpi4py
#SBATCH --partition=modi_HPPC
#SBATCH --nodes=1
#SBATCH --ntasks=2
mpirun apptainer exec \
   ~/modi_images/ucphhpc/hpc-notebook:latest \
   /opt/conda/envs/python3/bin/python test_mpi4py.py
```

4) To get started, you can click on the “Python File” icon in the launcher and rename it appropriately or create an empty python file, which can be opened in the editor by double-clicking it

```
touch test_mpi4py.py
```

5) Copy-paste a simple mpi4py program into the empty file. It is adapted from https://mpi4py.readthedocs.io/en/stable/tutorial.html#point-to-point-communication. It sends data from rank 0 to rank 1 and print it out. To make it possible to use mpi4py we add the path to it using the sys package). Remember to save the file after copy-pasting!

```
import sys
sys.path.append("/home/jovyan/erda_mount/__dag_config__/python3")

from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = {'a': 7, 'b': 3.14}
    comm.send(data, dest=1, tag=11)
elif rank == 1:
    data = comm.recv(source=0, tag=11)
    print(data)
```

6) Execute the python file on the frontend under MPI by going to the terminal tab. Make sure you are in the correct folder and write the command:

```
mpiexec --mca oob_tcp_if_include lo -np 2 python3 test_mpi4py.py
```

This is a basic test of python running under MPI. The correct output is:

```
{'a': 7, 'b': 3.14}
```