Homework 5
ME 5773 High Performance Computing Due: 02/27/2024 11:59pm

This is a team assignment. The teams are assigned by the instructors. The team members are posted on Canvas. The deliverable for this assignment is a zip file that should be uploaded to canvas by the due date. **No submissions will be considered after the due date.**
The .zip file should contain: A pdf document (2-page max) with the results summary, the files generate.py and loadFiles.py, and the excel file with the results. DO NOT include the generated files (CSV, NPY or HDF5).

1. **Exporting information to files.**
   Consider 5 matrices named A, B, C, D and E. The matrices should have the characteristics indicated below:

Matrix	Min. value	Max. value	Filling	Order	Shape	Type
A	2	9	Arbitrary	Fortran	(5000 × 5000)	64-bit integer
B	100	127	Arbitrary	C	(5000 × 5000)	8-bit integer
C	0.33333	0.33333	Exact	C	(5000 × 5000)	8-byte float
D	1001	1100	Sequential	Fortran	(10,10)	2-byte integer
E	350.0	350.3	Sequential	C	(2,2)	4-byte float

The minimum and maximum values indicate the limits for the actual values of the elements in the corresponding matrix. For instance, any element in **A** should have values between 2 and 9, i.e. $2 \leq A_{ij} \leq 9$. The filling indicates the way the elements should be defined within each matrix.
Arbitrary means each team is free to assign the values such that it complies with the limits.
Exact means use exactly the value indicated for all elements.
Sequential means that the array should be created with the first element as the minimum value, and the last element as the maximum value. The filling sequence must follow the specified order (row major, column major).

Note: When executing the following python scripts, use the 'hpc' conda environment in Arc, on a compute node.

a. Create a python script file named 'generate.py'
b. Import the libraries time, numpy and h5py.
c. Create each of the matrices A, B, C, D, and E. You can fill them any way you find appropriate, as long as they follow the indications in the table above.
d. Export each matrix into a corresponding CSV file. Name each file after the name of the array, e.g. the file for the array A should be named "`A.csv`".
   - Use numpy's savetxt function (https://numpy.org/doc/stable/reference/generated/numpy.savetxt.html).
   - The CSV file should follow the conventions for CSV file format.
   - For integer-typed matrices, use the export format `fmt='%d'`.

- For 64-bit floats, use the default format (`fmt='%.18e'`). This exports each element in scientific format with 18 significant digits.
- For 32-bit floats, export in scientific format with 7 significant digits.
- For each matrix, measure the time, in seconds, spent in generating the file and print it to console.

e. Export each matrix into a corresponding npy file format.
- Use the function `save` from numpy.
- Each matrix file should be named with the name of the matrix and extension ".npy".
- For each matrix, measure the time each file takes to save.

f. Export some matrices into a single hdf5 file named "`matrix_db.hdf5`". This should be done following the next indications:
- Create a group named "integer_group".
- Add to "integer_group" an attribute named "description", with its value being a string with a one phrase description of the elements in this group.
- Add to the integer group, the matrix A as a database with compression and with chunks of 500x500.
- Add the matrix B to the integer group, as a database with compression and with chunks of 1000x1000.
- Add the matrix D as a database into the integer group.
- Create a group named "float_group".
- Add matrices C and E as individual datasets. C should be compressed.
- Measure the time to create each database into the hdf5 file and print it to console.

g. Execute the python file and generate the different files.
- Measure the size of each one of the files using the terminal command `du -h <filename>`
- (du is a command to estimate the disk size of a file. The -h flag outputs a human readable format)
- Use the cpu times and file sizes of each of the elements to create a table comparative table of each of the matrices.

2. Reading information from files.

a. Create a python script file named 'loadFiles.py'
b. Import the libraries time, numpy and h5py.
c. Load each one of the matrices with the corresponding commands.
- CSV files: use numpy's loadtxt
- NPY files: use numpy's load.
- HDF5 files: use h5py's File and the specific instructions below(*).
- Measure the time required to load each one of the matrix files and print it to console
    i. (*) *Loading HDF5 file databases*: As seen in class, h5py does not load the data into memory once a file is first opened. The dataset is loaded when the database object is created and the data is requested with "[…]". This creates a numpy array with the data, and this is the moment you shouldmeasure its cpu time. For example:

```
db  = f['<group>/<database>']
ts  = time.time() #start time measure here
Mat = db[...]     #This is what loads data.
te  = time.time() #end time measure here
```

d. Execute the file and add the CPU times into an excel file to compare the load times for each matrix.
e. Write a short pdf document (2 pages max) indicating which file was the fastest to write, fastest to read and what format provided the smallest file size, for each of the matrices.