Homework 4
ME 5773 High Performance Computing   Due: 02/18/2024 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 an email to the instructors notifying the completion of the assignment. The activities of this assignment should be stored in the Teams github repository, within a folder named "assignment4". This assignment will be evaluated using the most updated version of the repository **by the due date**.

1. Conda environment

Define an environment file with name 'environment.yml' in the assignment4 directory. This file should create a new conda environment named envTeamX ("X" should be replaced with your team number).

The 'environment.yml' file should instruct conda to install the following libraries:

From channel:
1.1. Python version 3.10
1.2. Numpy (https://numpy.org)
1.3. Scipy (https://scipy.org)
1.4. Numexpr (https://github.com/pydata/numexpr)

From conda-forge channel:
1.5. Tensorflow version 2.15 (https://www.tensorflow.org)

From pip
1.6. Gmsh (Meshing tool https://gmsh.info)

Each student should use this file to create the envTeamX environment on his/her Arc account. The environment installation should proceed without issues.

After installing the environment file, export all the dependencies of the envTeamX as an explicit environment file named 'explicit_env.txt' and save it into the assignment4 directory.

Commit and update your server repository.

2. Python programming – Linear algebra

Consider the following matrix **K** and vector **f**:

$$\mathbf{K} = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 & 0 \\ 0 & -1 & 2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & -1 & 0 \\ 0 & 0 & 0 & \cdots & -1 & 2 & -1 \\ 0 & 0 & 0 & \cdots & 0 & -1 & 1 \end{bmatrix}, \quad f = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 1/N \end{Bmatrix}$$

- **K** is a matrix of size $N \times N$. Each diagonal element is $K_{ii} = 2$. Its last diagonal coefficient is $K_{NN} = 1$. Each element next to a diagonal coefficient is -1. The matrix **K** is zero elsewhere.
- **f** is a vector of size N. Each element in **f** is zero except its last element $f_N = \frac{1}{N}$.

The following system of equations is formed:

$$\mathbf{Ku} = \mathbf{f}$$

Where **u** is a vector of size N.

2.1. Create a python script named linalg.py in the assignment4 folder. This script should:

- Import the libraries numpy and time.
- Create an integer variable N. For example, use while developing the file N=5.
- Print the number N to console (stdout)
- Create the matrix **K** as a numpy ndarray variable. This matrix should correspond to the matrix of size $N \times N$ defined above. You can for example, first create a matrix (array) of zeros and then use for/while loops and if/else control statements to populate the matrix.
- Create the vector **f** as a numpy ndarray and populate it according to the description above.
- Measure the time elapsed while creating matrix **K** and vector **f**. Print this time to console (stdout) in seconds with 9 decimal digits.
- Find the solution to the system of equations $\mathbf{Ku} = \mathbf{f}$**.** Hint: Use numpy functions in the linalg module https://numpy.org/doc/stable/reference/routines.linalg.html
- Measure the time elapsed while solving for **u**. Print this to console (stdout) with 9 decimal digits.
- Print the solution vector **u.** In particular, print the last item u_N.
  For verification purposes, the solution to the last element in **u** is $u_N = 1$.

- After creating the linalg.py file, commit and update your repository.

2.2. Perform a runtime analysis of the execution time with respect to the actual number of threads used by the numpy library. For this:

- Define the size N to 10000 ($N = 10^4$)
- Create a slurm batch script file named job_linalg_study.slurm. This file should be configured to run on the compute1 partition with a single node, 1 task and 80 cpus-per-task. It must also output the stdout file and define a total CPU time of 5 minutes.
- This file must load the Anaconda3 module, and activate the envTeamX environment.

- Set the environment variable `MKL_NUM_THREADS` to 1, in order to specify the number of threads to execute MKL subroutines.
  ```
  export MKL_NUM_THREADS=1
  ```
- Execute the python script after each time environment variable is set with a different number, using:
  ```
  python3 linalg.py
  ```
- Repeat the previous two steps (definition of `MKL_NUM_THREADS` and running the linalg.py file) with the environment variable values of 2, 4, 8, 16, 20 and 40.
  It is suggested all these steps are done with a for loop in the bash script.
- Execute the job using sbatch. This generates an output file with the linalg.py output corresponding to each value of `MKL_NUM_THREADS`.
- Use the timing information from all the executions of linalg.py to generate the following plots (Use excel in your PC):
  (a) CPU time – Creation of K and f vs Number of threads.
  (b) CPU time – Solution of system vs Number of threads.
  (c) Total CPU time – (Creation+Solution) vs Number of threads.
  (d) Speed up Ratio vs Number of threads. The speed up ratio is computed as the ratio between the total CPU time for `MKL_NUM_THREADS=1` and the each thread
- Include the excel file that contains the plot in your assignment4 directory. as well as the text files with the timing outputs. Make sure your server repository is updated once completed.

3. Python programming – Numerical integration

Consider the integral.

$$F = \int_{-1}^{1} \sqrt{4 - 4x^2}\, dx$$

The exact solution for this integral is π. This integral can be approximated numerically using

$$F \approx \sum_{i=0}^{N-1} f(x_i)\, \Delta x$$

Where:

- $f(x) = \sqrt{4 - 4x^2}$
- $x_i = \frac{2}{N} i - 1$
- $\Delta x = \frac{2}{N}$

Create a python script named integral.py in your assignment4 folder. In this script, you will evaluate the numerical integral in three different ways. Before starting the three different ways, initialize the script with the following

- Import libraries numpy, math, numexpr and time.
- Define a variable N, as 10^9.
- Define a variable named deltax as 2/N

3.2. Using a for loop.

- Initialize a variable F1 as zero.
- Create a for loop that loops exactly N iterations.
- For each iteration, do the following:
  (a) calculate the corresponding value of x_i.
  (b) evaluate the function $f(x_i)$ using math's library functions.
  (c) Accumulate in F1 the value of $f(x_i)$*deltax. That is: F1 = F1 + $f(x_i)$*deltax
- Measure the time elapsed while computing $F1$**.** Print this in seconds to console (stdout) with 6 decimal digits.
- Print the value of F1 to console with 16 decimal digits.

3.3. Using numpy's vectorized functions.

- Create a numpy array '`i_vec`' with the values required in the summation. You can use, for example, numpy's "`arange`" function.
- Compute an array `x_vec` using '`i_vec`' values. Do not use a for loop for this. Use instead numpy's vectorized expressions:
```
x_vec = (2 * i_vec / N) - 1
```
- Compute an array `F_vec` using `x_vec`. Do not use a for loop, use instead numpy's vectorized expressions.
- Add up all the values in `F_vec` into the variable F2. You can use numpy's functions, for example "sum".
- Measure the total time elapsed to compute the integral (All steps in 3.3). Print this time in seconds to console (stdout) with 6 decimal digits.

3.4. Using numexpr evaluations.

- Create a numpy array '`i_vec`' with the values required in the summation. You can use, for example, numpy's "`arange`" function.
- Compute the array `x_vec` using '`i_vec`' values and the numexpr evaluate function.
```
x_vec = numexpr.evaluate('(2 * i_vec / N) - 1')
```
- Compute an array `F_vec` using `x_vec` using numexpr evaluate function. See https://numexpr.readthedocs.io/en/latest/intro.html#expected-performance for more information.
- Add up all the values in `F_vec` into the variable F3. You can use numexpr sum function as follows
```
F3 = numexpr.evaluate('sum(F_vec)')
```
- Measure the total time elapsed while computing the integral with numexpr functions (all steps in 3.4)**.** Print this time in seconds to console (stdout) with 6 decimal digits.

3.5. Perform a runtime analysis of the execution time with respect to the actual number of threads used by the numpy and numexpr library. For this:

- Create a slurm batch script file named job_int_study.slurm. This file should be configured to run on the compute1 partition with a single node, 1 task and 80 cpus-per-task. It must also output the stdout file and define a total CPU time of 5 minutes.
- This file must load the Anaconda3 module, and activate the envTeamX environment.

- Set the environment variables `OMP_NUM_THREADS` and `NUMEXPR_NUM_THREADS` to 1. For example
  ```
  export OMP_NUM_THREADS=1
  ```
- Execute the python script after each time environment variable is set with a different number, using:
  ```
  python3 integral.py
  ```
- Repeat the previous two steps (definition of environment variables and running the integral.py file) with the environment variable values of 2, 4 and 8.
  It is suggested all these steps are done with a for loop in the bash script.
- Execute the job using sbatch. This generates an output file with the integral.py runtimes.
- Use the timing information from all the executions of linalg.py to generate the following plots (Use excel in your PC):
  (a) CPU time for each integral vs Number of threads. (you may need to use logarithmic y axis).
  (b) Speed up Ratio vs Number of threads. For each integration 3.1, 3.2 and 3.3, calculate the speed up ratio computed as the ratio between the CPU time for `1` thread and the CPU time with X threads. Plot these in a single plot.
- Include the excel file that contains the plot in your assignment4 directory. as well as the text files with the timing outputs. Make sure your server repository is updated once completed.