Homework 6
ME 5773 High Performance Computing Due: 03/24/2024 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 a summary of the results, the python files created for this analysis, and the excel file with the timing results.

1. Finite Element problem assembly.

Consider the 1D Finite Element problem of linear elasticity. The goal is to solve the differential equation:

$$\frac{d}{dx}\left(AE\frac{du}{dx}\right) + b = 0$$

where u is the displacement field along the x direction, A is the cross-sectional area, E is the Young's Modulus and b is a body force. This equation relates the forces due to the deformation of a 1D solid and the external forces acting on it. The following reading is recommended [1]

One can simplify the analysis, by considering the solid subdivided into multiple spring elements. Each spring element deforms linearly according to its elastic constant $k^{(e)} = AE$. Consider an element (e), as shown below, formed by nodes i and j; with corresponding unknown nodal displacements u_i and u_j and known nodal forces $f_i^{(e)}$ and $f_j^{(e)}$.

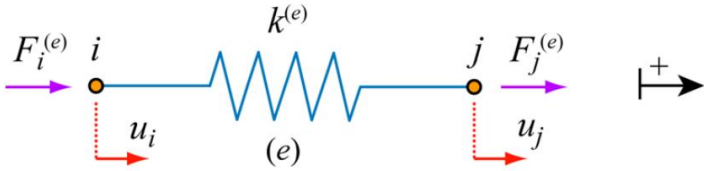

Figure 1. Schematic of a spring element. Image taken from [1].

The elastic behavior can be therefore summarized as follows:

$$k^{(e)}u_i - k^{(e)}u_j = f_i^{(e)}$$
$$-k^{(e)}u_i + k^{(e)}u_j = f_j^{(e)}$$

In matrix form, it is as follows:

$$\mathbf{K}^e\mathbf{u}^e = \mathbf{f}^e$$

These are called the element stiffness matrix ($\mathbf{K}^e$), the elemental nodal displacement vector ($\mathbf{u}^e$) and the elemental force vector ($\mathbf{f}^e$).

$$\mathbf{K}^e = \begin{bmatrix} k^{(e)} & -k^{(e)} \\ -k^{(e)} & k^{(e)} \end{bmatrix}, \qquad \mathbf{u}^e = \begin{Bmatrix} u_i \\ u_j \end{Bmatrix}, \qquad \mathbf{f}^e = \begin{Bmatrix} f_i^e \\ f_j^e \end{Bmatrix}$$

This elemental matrix system only corresponds to the effects of a single element in the solid. If there are more subdivisions, then the global behavior has a contribution from this element.

The assembly operation takes the elemental components and defines them in the corresponding global locations for the global system. Consider the two-element system below

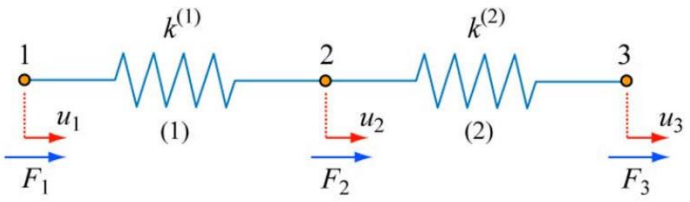

Figure 2. Schematic of two spring elements. Image taken from [1].

The global system will have a shape of 3x3 because there are 3 displacements (unknowns, degrees of freedom). The system is as follows:

$$\underbrace{\begin{bmatrix} k^{(1)} & -k^{(1)} & 0 \\ -k^{(1)} & k^{(1)}+k^{(2)} & -k^{(2)} \\ 0 & -k^{(2)} & k^{(2)} \end{bmatrix}}_{\mathbf{K}} \underbrace{\begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix}}_{\mathbf{u}} = \underbrace{\begin{Bmatrix} f_1 \\ f_2 \\ f_3 \end{Bmatrix}}_{\mathbf{f}}$$

For the case of consecutive spring elements, we can assure that the element matrix is assembled in the global matrix region enclosed by the $e{:}\,e+1$ range. For example, for the previous example, notice that the element 1 encompasses the columns 1 and 2 of the global matrix, and element 2 the region of columns 2 and 3. Similarly, the rows are distributed in the same manner. An example file is provided in the folder `/work/ME5773/hw6/FE_system.py`

Your task is to parallelize the assembly of the elements in a system with a large number of elements. Take the file FE_system.py as a base, and use the python multiprocessing module to parallelize the calculation of the Stiffness matrix and force vectors.

a. Create a copy of the sample python script file named 'FE_system_groupXX.py', replace the name accordingly with your group number.
b. Modify the code such that it is effectively parallelized using multiprocessing pool object.
    - Use domain decomposition approach: Subdivide the assembly process into a subset of the total number of elements, and send it to each worker.
    - You can create in each worker a new global matrix, and force vector, and then return it after each evaluation.
    - After each worker has finished, each matrix can be added.
c. Measure and plot the speedup and efficiency for runs assembling systems of 50 000 degrees of freedom with 1, 2, 4, 6, 8, 16, 20, and 40 workers.
d. You may notice that the matrix is filled with a significant number of zeros. This reduces performance because of memory usage, and overhead related to copying large datasets. A way to overcome this is to use sparse matrices. Sparse matrices are a way to improve performance of systems with matrices that have a large number of zeroes. Recommended reading: https://en.wikipedia.org/wiki/Sparse_matrix
    - Install the scipy library with conda in your Arc's hpc environment,
    - Create a copy of the file 'FE_system_groupXX.py', named 'FE_system_sparse_groupXX.py'. Change the type of the matrix from np.zeros() to scipy's sparse matrix.
    - Each worker should create a matrix in list of lists format https://docs.scipy.org/doc/scipy/reference/sparse.html. The slicing operations are

the same as numpy arrays, thus no other modification is required. This type of sparse matrix provides a fast slicing and item assignment operations, so it its appropriate for the assembly process. Only the non-zero values will be stored and created.

- After each worker has assembled its portion of the procedure, convert the matrix into CSR format (provides faster addition and multiplication operations). Return this CSR matrix from the worker job.
- The force vector should be computed with a normal numpy array, as it does not contribute significantly to the process.

e. After changing the matrix system use to sparse matrix, measure and plot the speedup and efficiency for runs assembling systems of 500 000 degrees of freedom with 1, 2, 4, 6, 8, 16, 20, and 40 workers.

[1] J. Dean, Introduction to the Finite Element Method (FEM), Lecture notes. University of Cambridge, Accessed on the 3/18/2024 from https://www.ccg.msm.cam.ac.uk/system/files/documents/FEMOR_Lecture_1.pdf