Nested grids treatment

García Pérez, Jorge Alberto[*]
Solar Corona - Spacecraft Interaction (SCSI)

When conceiving the idea of implementing a Nested Grid (NG) for the SCSI code, I thought of two possibilities that can benefit frommy already constructed classes. I think most of the code from the function `poissonSolver_2D_rm_SORCA`, as well as the entire classes of `Species`, `Mesh_2D_rm_sat`, `PIC_2D_rm1o`, `Electrostatic_2D_rm_sat` and the boundary classes can be recycled for this new step in the code without sacrificing to much the efficiency of the execution.
The two possibilities considered are:

1. Create mini simulations for each patch, with the entirety of the objects replicated for each patch. This means, for example, several types of `Species` objects for the same physical Species. That also means that when a particles leaves a finer mesh, the particle is treated as it has left through a boundary, being deleted from that `Species`, and then being injected in the correspondant `Species` for the new patch. There are several issues that I have not thought about it, and I think this solution requires more work and is less efficient than the following.
2. Create a Mesh class that englobes all the patches of the dominion. This class will handle several childern `Mesh_2D_rm_sat` or `Mesh_2D_rm` objects that will account for the different patches. The responsabilities in the code execution will be or either conducted by the wrapping class alone, or the class will organize and delegate responsabilities to its children. So, for example, there might be an enumeration of the nodes of the mesh as a whole, which will be known only by the parent class, and then several enumeration systems particular to each patch that will speed up the processes conducted in each children object. Acually, since many of the procedures are very recursive, I think I will try to create like a wrapper class for `Mesh_2D_rm_sat` and `Mesh_2D_rm` such that the objects inherit the functionalities and behaviour of these classes but also gain the functionalities of having children meshes and all the functionalities necessary for NGs.

I have decided to try the latter option, which now I will proceed to describe:
My plan for working with the idea of Nested Grids is that, since a particular grid will have a way of carrying out the different procedures linked with a mesh, like calculating electromagnetic fields, doing the scatter and gather processes or implementing mesh functions like `arrayToIndex` and `getIndex`, the intention is to separate the feature of the mesh having another mesh inside with its characteristics that are due to the nature of the mesh itself. This will be done by creating a new class for NGs that will inherit, both the type of mesh it is, as well as the functionalities it needs to implement due to having nested meshes within it. The latter concept will be carried out by creating an **abstract class** that will have the attributes and functions every class with NGs needs.
The same methodology will be applied to the different types classes that need a transformation to handle NGs. The type of classes that will be affected are: Field, PIC and Mesh. Since some instances of these classes are attributes of other objects of these same classes, there might be a bit of confussion, so the following rules will be used for organization:

- **Classes that are attributes of other classes:** Field has a PIC object as attribute, Motion_Solver has a PIC object as attribute, PIC has a Mesh object as attribute, and Mesh has several Boundary objects as attributes. In these cases, the classes will have as attributes the objects that refer to the same Nested Grid that the class is handling. So far Boundary objects will not be part of the recursive process, but rather, each Mesh object of a NGs will contain its borders specified.
- **Parent-child relationships between objects of the same class:** Since all the classes mentioned before will have NG-related functionalities and will function recursively, all of them will hold relationships analogue to the ones for Mesh classes. For meshes, a parent mesh will contain a list of children, which are the subgrids contained in-and-only-in the current mesh. Thus, for example, the PIC object that contains a Mesh object will have as children the PIC objects that are related with the children of the Mesh object.

Since `Particles_In_Mesh` class simply accumulates the information of the Species in the domain as big numpy arrays, the information between the index of the node and its correspondent position in the domain must be provided by the mesh. Since the mesh as a whole is not a structured mesh, the functions `getIndex`, `arrayToIndex` and `indexToArray` do not have a clear functionality. What I am considering now is to leave these functions the way they work in the class `Mesh_2D_rm` and add aditional functions that treat the addition of Nested Grids. For organizing the nodes in a 1D array, I propose the following:
Indexing: Since the coarsest mesh contains all the other meshes inside, and recursively that happens for every new

[*] ja.garciap13@gmail.com

inner layer of meshes, I can start by indexing as a normal 2D rectangular mesh the coarsest mesh. The indexing is horizontal first, starting from the bottom-left corner and then going right and up until the top-right corner. After finishing the indexing of this mesh the next node is the bottom-left corner of the first patch found when moving according to the indexing procedure. Then, the patch is indexed in the same methodology as the coarses mesh, and it will go in the big 1D array from $nPoints_0$ to $nPoints_0 + nPoints_1$, for an 0-based indexation with the coarsest mesh having $nPoints_0$ nodes and the second mesh having $nPoints_1$ nodes. Then, it is checked whether there are finer meshes inside the first patch, and the process is started again, recursively, until the first patch and all its children patches are mapped to the 1D array. Then, the second patch found in the indexation of the coarsest mesh is processed. This procedure is repeated until all the nodes in the NGs are indexed.

Scatter: For scatter, the idea is that I can implement part or all the methods in the abstract function. So scatter definition on the current implemented class is `scatter(self, positions, values, field)`. Since is so broad, I can use it directly for grids without NGs and for grids with NGs. For grids with NGs, the only thing is that it is necessary to build positions and values not only from the particles that belong to the mesh, but also from the subgrids' values. Then, for example, for `scatterDensity`:

Implementation on current class (PIC_2D_rm1o(PIC)): `scatterDensity(self, species)`

1. If the mesh is the root, make density = 0.
2. Go through the list of children PIC objects of this object and execute `scatterDensity`.
3. Execute `scatter` with the following parameters: `positions`: The positions of the nodes of the children meshes that are not zero, plus the position of the particles assigned to the current mesh. `values`: The values of the aforementioned nodes times their respective volumes, plus an array of the size of the particles assigned to this mesh filled with the SPWT value for the species being treated. `field`: The correspondant array to be treated, but only the section of the array that corresponds to the nodes assigned to the current mesh.
4. Execute the division by volume in the nodes assigned to this mesh.
5. I have to think about the accDensity part.

Note: Actually, I think is worthy to compare the previous methodology for doing a Scatter function, with the simpler idea of running through the the whole set of particles over an over and create with the same particles different scatters for different meshes. To see what would be the difference, let's consider one mesh and a child finer mesh. On one side, when doing scatter on the parent function, the former methodology would prevent us from going over the whole set of particles that are inside the children mesh again. However, some resources will be spend on the creation of the arrays that will act as fictitious particles for the parent mesh. Then, the comparison between number of nodes in the children mesh vs. the number of particles inside that mesh is important. Nevertheless, since there should be $20 \approx 50$ particles per cell, the reduction is expected to be significant. Other topic about this is the increment in inaccuracy of the scattered values in the parent mesh.

Handling of particle-to-mesh assignment:

Inject: The particles will be injected to the dominion through the boundaris of the root mesh. Since the root mesh has the information of all the NGs, any recursive methodology can be applied to carry out any part of the process.

Wall-related values: As there are phenomena that connect all the wall