Leapfrog PIC treatment

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

The thing is that, for now, I don't need to update the velocity and temperature in every step, since these values in the mesh are not used in any other subroutine. They are not used as input in any physical process that modifies the behaviour of the system. So far, they are only used as output. Therefore, I want to disgregate the update of velocity and temperature from the update of positions.
Clearly, the group of attributes of the particle that need to be updated in every step will change as new phenomena are included in the simulation. Because of that, I updated the function 'updateMeshValues' to give three possibilies, distinguished by different values of a karg: 0 if all the attributes are to be updated, 1 if only the 'necessary attributes' are to be updated, and 2 if only 'non-necessary' attributes are to be updated. Then, with every new phenomenon included, probably the attributes distributed among the three categories will change. **Note:** It would be a good idea to store every previous version of the 'updateMeshValues' , mentioning which phenomena the program was handling at the moment, and the date when the function was removed.
Now, regarding the update of temperature and velocity, thera are two points for consideration:

1. The particle $\rightarrow$ mesh procedure chosen.
2. The time synchronization between velocity-related mesh attributes and position-related mesh attributes, in order to print results and to use velocity-related attributes to calculate things further needed for computing forces at the same timestep of positions. This point, given the fact that some time integration method like Leapfrog is being used, where velocity and position are normally desyncrhonized.

Fundamentally, both items can be dettached by first creating a temporal species that has the velocity values (in Particles) already scyncrhonized with position. Then, scatter process is done for velocity and any other attribute to be updated. The opposite can also be done, first scatter, then $\frac{mesh_{i-1/2}+mesh_{i+1/2}}{2}$. About this last procedure, I am not sure if it adds more error to the final result, since at least for the former method, being the standard procedure, it wouldn't be more than the addition of the second-order error of Leapgfrog + the error of the scatter function.

There is even a third procedure where both processes are done at once. Inside of the scatter process, for every attribute related with velocity, a computation is done in a particular way to maintain the second-order accuracy of Leapfrog while reducing the computation steps. This would be the more computationally-efficient solution but it reduces the generality of the program. Why? Because in the first two methods, the scatter process is only dependant on the PIC methodology that was selected, and the 'synchronization' part is only dependant on the type of time integrator selected.

I am not sure whether the third option is the most efficient one. What is the difference between computing the values at step i for velocity, and then scattering, from doing both things at the same time. Two 'numpy for' doing half the work, or one 'numpy for' doing all the work, but then not having the velocity ready for the rest of attributes.

At the end I tried the two methods mentioned in the previous paragraph, and the reduction in time by the latter method (in comparison with the first) is almost non-existent. Given that, the fact that later it is easier to implement different PIC and time integration methods, and that for the rest of attributes, when applied scatter, the velocity is already prepared in the i step, the first method will be used. See folder 'time_test' in folder 'tests_results' for the data and graphs.

* ja.garciap13@gmail.com