Design Plan Hypre Sub-Cycling

1. Functionality

To equip hypre with the ability of sub-cycling itself. Used for solving diffusive processes. The general idea is determining a preferred time step within hypre and advancing with this time step till the required hydro time step length is reached.

2. Current implementation

Currently, the sub-cycling is done in the Elliptic routine in bearez.f90. It determines a preferred time step by calling routine HypreTimeStep(Info, dt_hypre), then repeatedly call hypre till the accumulated time is equal to the time step from hydro part. To determine the preferred dt, we can use the formula:

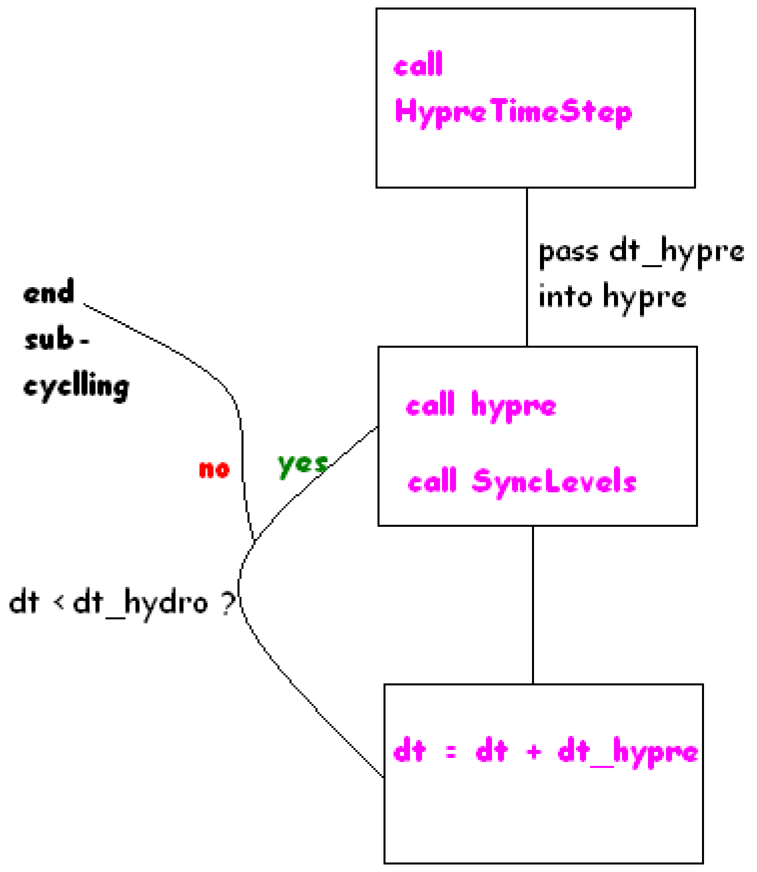
$$\delta t = f \frac{\delta l^2}{\kappa_e} \qquad (1)$$

where δl is the smallest physical length scale determined by the problem. For instance, to observe the change of an ablation front, this length can be chosen as the width of the ablation front. f is called "leap factor" which indicates how aggressive we are when using the implicit solver. A reasonable choice will be f = 1, but sometimes a problem can run OK with a larger f. κ_e is the effective diffusivity determined by the following relation:

$$\kappa_e = \kappa T^n / \rho$$

with κ being the Spitzer diffusivity, n being the diffusion index (n = 2.5 for electron diffusion), T being temperature, ρ being density. Naturally, in determining a preferred time step, one would choose δl to be four or five grid width (in the ablation problem, the ablation front is about 5 grids in width), the largest κ_e across the domain, and a reasonable leap factor (usually by try and error). In routine HypreTimeStep, the largest κ_e is determined by a subroutine called findTmax(Info, Tmax), which simply loop through the entire domain and returns the maximum κ_e stored in Tmax. HypreTimeStep then use equation (1) to compute a time step stored in dt_hypre and return it.

In sub-cycling hypre, there are two options. The first one is determining a dt_hypre per sub-cycle, the other is determining a dt_hypre only at the first sub-cycle, then use this dt_hypre throughout the rest sub-cycling. Although dt_hypre tends to grow b/c of the peak temperature is in general decreasing, the benefit from generating a dt_hypre each sub-cycle cannot justify its cost. So we are currently using method 2, only determining dt_hypre at the initial cycle. The whole process can be illustrated by the following flow chart.



3. Current Problems

Although the fixed grid sub-cycling is working, it has several flaws. Here I list the problems that needs to be solved by this design plan.

(1) Since the sub-cycling is done in the Elliptic routine, hypre has to do the assembling and destruction for each cycle, which is not needed. Because the grid configuration should be exactly the same for all the sub-cycles, we should be able to only cycling the following three routines:

MatrixSetValues, VectorSetValues, VectorGetValues
So the cyclings should rather be done within the routine StructHypre and SemistructHypre, but not routine Elliptic.

(2) If we cycle the above three routines within StructHypre and SemistructHypre, we need to update the boundaries inside hypre for each sub-cycle, which is tricky. I used to write a routine trying to copy the boundary values, but it seems it cannot work well inside hypre (it generates kinks on the boundaries). That is partly why currently sub-cycling is done in Elliptic routine, since there I can call SyncLevel to do the updating. Within hypre, we cannot call the SyncLevel routine, so we need to come up with a function that can update the boundaries properly within hypre.

4. Solution

My current thought is to write hypre versions for Synclevels and all the SetBoundaries routines in hypreBEAR so that we can call them while sub-cycling.