Lagrangian Point Particle code Readme Documentation

Ahmed Basil KOTTILINGAL
Sorbonne University

2021
August

1 Introduction

This code works with basilisk (http://basilisk.fr/). To install basilisk, refer to http://basilisk.fr/src/INSTALL

Use the line

```
#include "lpp.h"
```

for including tracer particles in the NS simulation. **(As of now LPP is not implemented. Only tracer particle option is available)** Initialisation of tracer particle has to be carried out by the user.

2 Links

To access the code visit the gitlab project https://gitlab.com/ahmedbasil/lpp/-/tree/master/src.

3 Drag Model

Simple model

$$\frac{d}{dt}\mathbf{U}_p = -\frac{1}{\tau}\left(\mathbf{U}_p - \mathbf{u}\right) + (\rho_p - \rho)\frac{4\pi {r_p}^3}{3}\mathbf{g}$$

Explicit discretisation

$$\frac{\mathbf{U}_p^{n+1} - \mathbf{U}_p^n}{\mathtt{dt}} = \frac{1}{\tau}\left(\mathbf{u} - \mathbf{U}_p^{n+1}\right) + \mathbf{g}',$$

4 Two way coupling

...**NOT YET DONE**

5 Struct used

The struct named Particle stores the information of the coordintae of a point particle. It also stores velocity, density and radius of the point particle. Each particle has a unique tag of type long.

```
typedef struct{
  double x[dimension];
#if _MPI
  int pid;
#endif
#if _PINERTIAL
  double u[dimension];
  double rho, r;
  long tag;
#endif
}Particle;

```

The struct Particles maintain list of particles. The default list of Particles is _particles.

```
typedef struct {
  Particle * p;
  unsigned int len, max;
}Particles;

```

To add a new particle to a list of particles, use add_particle(). The function add_particle() return the pointer of newly added particle (Particle *) which can be used to set the properties of the particle. For example

```
Particle * p = add_particle(P); //Adding a new particle to P
p->x[0] = 3.5;                  //setting properties of particle, p

```

6 Particle Cache

Say "P" is a set of particles (Particles *), to iterate through the particles in P, you can use the iterator foreach_particle(P). The iteration order is random as the particles are randomly ordered. But in many case you might need to access the particles in each AMR cell, its neighbors etc. So to do that, we maintain another cache of particles.

```
typedef struct{
  Particle ** cache;
  unsigned int len, max;
}Particles_cache;

```

The default cache of particles is Pcache. To set Pcache, call the function

```
particles_cache_update(P);    \\maintaing a cache of particles, P in Pcache
                              \\Pcache is the _default cache;

```

When the particles_cache_update(P), the scalars nparticles[] and zparticles[] are updated. nparticles[] store the number of particles in each AMR cell and zparticles[] store the starting integer index to the cache of the particle in each AMR cell.

So inorder to iterate through all the particles in the cell you can use the iterator foreach_particle_in_cell(zp, np). Example for the usage is

```
foreach_particle_in_cell((int) zparticles[], (int) nparticles[]){
    foreach_dimension()
      assert(xp >= (x-0.5*Delta) && xp <  (x+0.5*Delta));
      //xp is the coordinate of particle.
}

```

To iterate through all the particles in the cache,

```
foreach_particle_cache(){
    //..
}

```

Make sure that Pcache is updated by calling particles_cache_update(P) before using any particle cache iterators.

7 Foreach Iterators

- foreach_particle(P)
  You can iterate through all the particles in the list P.
- foreach_particle_in_cell(zp, np)
  You can iterate through all the particles in the cache Pcache whose integer indices lies in [zp,zp+np) .
- Inorder to iterate through all the particles in the cell, use

```
    foreach_particle_in_cell((int) zparticles[], (int) nparticles[]){
    }
```

- In order to iterate through all the particles in Pcache, use

```
    foreach_particle_in_cell(0, Pcache->len){
    }
```

- You can also use foreach_particle_cache() to iterate through all the particles in Pcache

Make sure that Pcache is updated by calling particles_cache_update(P) before using any particle cache iterators.

8 lpp-tree.h

If Octree (or quadtree) is being used in basilisk, you (might) have to use grid adapt function adapt_wavelet(). If you are using lpp.h, **make sure you use adapt_wavelet_particles() instead of adapt_wavelet()**. Use arguments in the same way.

Will try to fix in the next update of lpp-tree.h

9 Additional Notes/Readings

9.1 Warning

Some testcases (eg:random.c) doesn't work with latest verions of basilisk

Current version doesn't work for periodic bounday.

Need to fix locate_rank (struct _locate p) in lpp-common.h

9.2 lpp-view.h

The function lpp_view (struct _lpp_view p) is not well written.

Make sure the correct GL libraries are included when including lpp-view.h

9.3 Advection Particles

To advect all the particles in time with a local velocity vector **u**, use the function particle_advance(Particle * particle, vector u). Refer lpp.h

9.4 Adding a particle to Particles list

Suppose you wan't to add 100 points to the Particles list P, **you have to make sure enough memory is created to accomodate them**. You can do this by

```
particles_append(P, 100);

```

Then you can add 100 points like this

```
for (int i=0; i<100; ++i){
    Particle * p = add_particle(P); //Adding a new particle to P
    //Don't forget to set the coordinates and other props of the particle
    p->x[0] = 3.5;                  //setting properties of particle, p
    p->x[1] = 2.1;
}

```

9.5 Particle Boundary Condition

To add boundary condition to each particle, use the default function pointer particle_boundary.

```
int (* particle_boundary)(Particle * p) = NULL;

```

Example of a reflective boundary condition is given here

```
#define reflection(x) {\
x[0] = x[0] < X0 ? (2*X0 - x[0]) : x[0] > (X0+L0) ? (2*(X0+L0) - x[0]) : x[0]; \
x[1] = x[1] < Y0 ? (2*Y0 - x[1]) : x[1] > (Y0+L0) ? (2*(Y0+L0) - x[1]) : x[1]; \
}
int reflection_boundary(Particle * particle){
        particle_var();
        reflection(particle->x);
        return 1;
}
event defaults(i=0){
        particle_boundary = reflection_boundary;
}

```

Note that boundary condition is used to update coordinate of particles even though you can update other particle variables also inside the function. **Make sure reassigning new coordinates to particles doesn't violate the CFL condition. There is only one function pointer as of now. So put all boundary condition in one function**

9.6 Debug option

Add the compiler flag **-D_PDEBUG** to add debug option

9.7 Other compiler options

Add the compiler flag **-D_PINERTIAL** to simulate inertial particles. Add the compiler flag **-D_PTAG** to add tag variable to each particle. Add the compiler flag **-DTWO_WAY** to add two way force coupling.

9.8 Examples