Interface, outputs, post-processing

Frédéric Pérez

SMILEI training workshop
November 6-7, 2017
Maison de la Simulation

The minimum knowledge to use Smilei

if you want to keep away from the code

1. Compile
2. Write an input file (a.k.a. *namelist*)
3. Run the program
4. Read & post-process the outputs

Smilei already does much of the work for you

The minimum knowledge to use Smilei

if you want to keep away from the code

1. Compile
2. Write an input file (a.k.a. *namelist*)
3. Run the program
4. Read & post-process the outputs

Requirements to compile & run

- C++11 compiler (with openMP)
- MPI, supporting `MPI_THREAD_MULTIPLE` if openMP
- Compatible HDF5
- Python 2.7+

Help is given on our website

Compiling Smilei

```
$ make
```

```
$ make -j 8       # compile with 8 processors
```

```
$ make clean      # reset compilation
```

```
$ make doc        # compile documentation
```

The minimum knowledge to use Smilei

if you want to keep away from the code

1. Compile
2. Write an input file (a.k.a. *namelist*)
3. Run the program
4. Read & post-process the outputs

The input file is written in *python*

- While the code is *C++*, the input is *python*
- Units are all normalized ($c, m_e, q_e, m_e c, m_e c^2, n_c$, etc.)
- Instructions interpreted by Smilei only inside *blocks*

```
Main(
  timestep = 0.01,
  grid_length = [10., 20.],
  ...
)

LoadBalancing(
  every = 200
)
```

```
Species(
  name = "electrons1",
  particles_per_cell = 1000,
  ...
)

DiagFields(
  every = 1000,
)
```

The *python* input provides flexibility

- Calculate simulation parameters at runtime

```
omega0_SI    = 2. * math.pi * 3e8 / 1.06e-6
duration_SI = 300.e-15
Main( simulation_time = duration_SI * omega0_SI, ... )
```

Change units

```
for s in [“ion1”, “ion2”, “ion3”, “ion4”]:
    DiagParticleBinning( species = [s], ... )
```

Loops

- Plasma & laser profiles are given as *functions*

```
def f(x,y): return x*math.exp(-(y-y0)**2)
Species( number_density = f, ... )
```

User’s profile

```
Species( number_density = gaussian(fwhm=3), ... )
```

Built-in profile

- Any python code accepted

Import modules, read external files, run other scripts, etc.

The minimum knowledge to use Smilei

if you want to keep away from the code

1. Compile
2. Write an input file (a.k.a. *namelist*)
3. Run the program
4. Read & post-process the outputs

A *test mode* to check input consistency

```
$ smilei_test myinput.py
```

Runs through code initialization only.

Only loads one patch.

Fast & convenient

A basic example to run a simulation

```
$ mkdir mysimulation
$ cp myinput.py mysimulation
$ cp smilei mysimulation
$ cd mysimulation
$ mpirun -n 4 smilei myinput.py
```

The minimum knowledge to use Smilei

if you want to keep away from the code

1. Compile
2. Write an input file (a.k.a. *namelist*)
3. Run the program
4. Read & post-process the outputs

Outputs

- Standard output: *lots of info, warnings, errors, etc.*
- Diagnostics

`Scalar`	TXT	global simulation quantities
`Fields`	HDF5	direct output of the fields arrays
`Probe`	HDF5	fields interpolated on regular grids
`ParticleBinning`	HDF5	versatile averaged particle data
`Screen`	HDF5	time-integrated particles passing through surface
`TrackParticles`	HDF5	particle trajectories

- Migration towards openPMD (standard for particle/mesh data)

  Operational for `Fields` & `TrackParticles`

- Checkpoints (≡ dumps and restarts)

  HDF5 files in the `checkpoints` folder for restarting the simulation

Included *python* post-processing

- The repository includes a *python* module happi

```
$ make happi
$ ipython
In [1]: import happi
```

Limitations: no parallel processing; limited in memory

- Get simulation parameters

```
In [2]: S = happi.Open(“path/to/my/simulation/”)
In [3]: timestep = S.namelist.Main.timestep
```

- Data manipulation

basic operations, change units, slicing, etc.

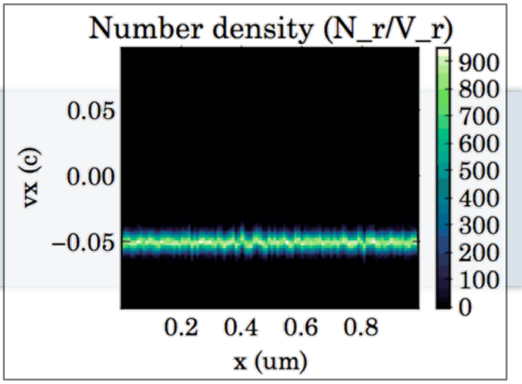
- Obtain raw or processed data

```
In [7]: S.ParticleBinning(0).getData()
```

Easy visualization

- Plot simulation results

```
In [4]: S.ParticleBinning(0).plot()
In [5]: S.ParticleBinning(0).streak()
In [6]: S.ParticleBinning(0).animate()
```



- Convert to VTK format (for VisIt or Paraview)

```
In [7]: S.ParticleBinning(0, timesteps=1000).toVTK()
```

- The openPMD-compliant diagnostics can be opened in VisIt

The minimum knowledge to use Smilei

if you want to keep away from the code

1. Compile
2. Write an input file (a.k.a. *namelist*)
3. Run the program
4. Read & post-process the outputs

Questions ?