PySPH: Smoothed Particles Hydrodynamics in Python

Pankaj Pandey Kunal Puri Prabhu Ramachandran

Department of Aerospace Engineering
IIT Bombay

SciPy.in 2010

Outline

Outline

Smoothed Particle Hydrodynamics
SPH

- Mesh-free Lagrangian particle method
- Started with applications to astrophysics
- Fluid mechanics, free surface flows, fracture, porous media, explosions
- Games and videos animations
- Major advantage lies in simulation of complex problems with moving geometries

SPH Basics

Interpolating Integrals

$$f(\mathbf{r}) = \int f(\mathbf{r}')\delta(\mathbf{r} - \mathbf{r}')d\mathbf{r}'$$

Kernel Approximation

$$f(\mathbf{r}) \approx \int f(\mathbf{r}')w(\mathbf{r} - \mathbf{r}', h)d\mathbf{r}'$$

Particle Approximation

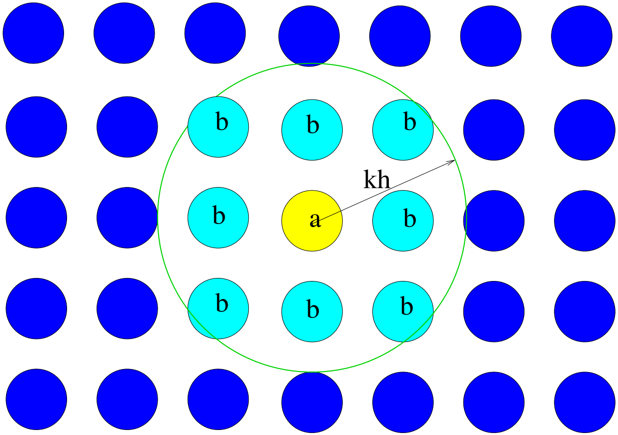
$$\langle f(\mathbf{r}_i)\rangle = \sum_j f(\mathbf{r}_j)\frac{m_j}{\rho_j}w(\mathbf{r}_i - \mathbf{r}_j, h_j)$$

SPH Basics

- Smoothed particles diffused in space by kernel function
- Material properties are weighted sum of particle properties
- Particle properties updated using Lagrangian form of governing equations
- Derivatives transferred to the kernel function
- PDE $\rightarrow$ ODE

SPH Basics

SPH Approximation



PySPH

- Parallel extensible SPH framework written in Python/Cython
- `http://code.google.com/p/pysph`
- Open source (BSD licensed)

Why PySPH?

- Many SPH codes exist, including few open source
- Most in Fortran/C/C++
  – Learning and extension difficult, slow development, memory management complicated
- Needed tool for experimentation in SPH methods, trying out new methods and ideas quickly
- Choice of Python as programming language for PySPH

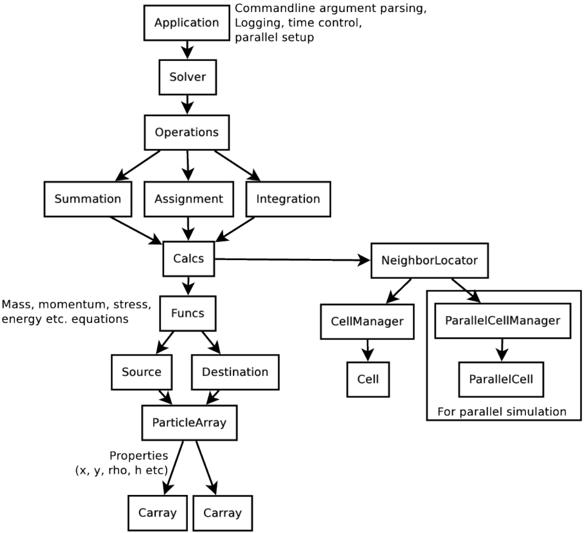
Why Python?

Python

- easier and faster to learn and code
- easier for new comtributors to start contributing
- self-documenting (doc-strings)
- builtin high level data structures
- lots of available scientific/other libraries
- performance critical sections written in Cython (compiles python-like code to C extension)

Outline

PySPH Architecture Overview



Particle Storage

- Particle properties (mass, x, y etc) stored in carrays (resizable typed c arrays like numpy 1d arrays)
- Different arrays stored in a ParticleArray (Python allows adding/removing arrays as named attributes)
- Different ParticleArrays for different entities
- All SPH operations on ParticleArrays

```
pa =
pysph.base.get_particle_array(name='fluid',
type=pysph.base.Fluid, x=x, m=m, ...)
```

Solver

Solver is has all the operations to be performed in a simulation. Steps for creating a new solver are:

- Define relevant operations (subclasses of **SPHFunctionParticle**) in the *sph* module (e.g. Hooke's law in case of solid mechanics problems)
- Add relevant operations to instance of the *Solver* class (or a subclass to make the solver reusable) in appropriate order (e.g. add Hooke's law before the stress-momentum equation)
- Add relevant properties to the particles (e.g. stress σ_{ij}) while creating the particles for the an problem to solve

Solver

Solver is has all the operations to be performed in a simulation.
Steps for creating a new solver are:

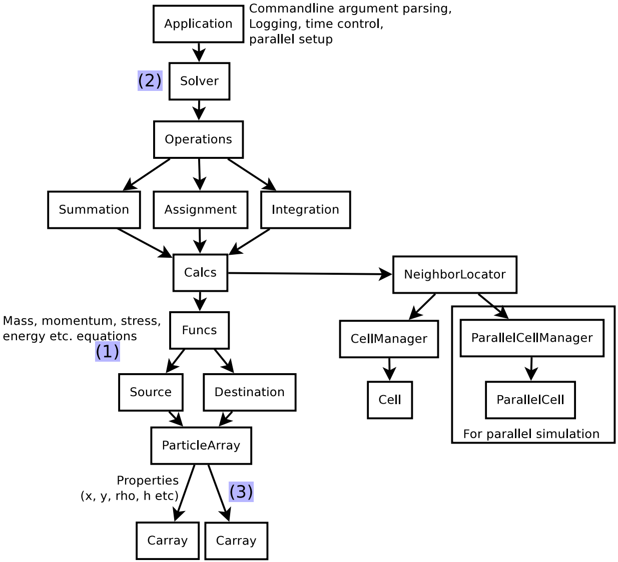
- Define relevant operations (subclasses of **SPHFunctionParticle**) in the *sph* module (e.g. Hooke's law in case of solid mechanics problems)
- Add relevant operations to instance of the *Solver* class (or a subclass to make the solver reusable) in appropriate order (e.g. add Hooke's law before the stress-momentum equation)
- Add relevant properties to the particles (e.g. stress σ_{ij}) while creating the particles for the an problem to solve

Solver

Solver is has all the operations to be performed in a simulation.
Steps for creating a new solver are:

- Define relevant operations (subclasses of **SPHFunctionParticle**) in the *sph* module (e.g. Hooke's law in case of solid mechanics problems)
- Add relevant operations to instance of the *Solver* class (or a subclass to make the solver reusable) in appropriate order (e.g. add Hooke's law before the stress-momentum equation)
- Add relevant properties to the particles (e.g. stress σ_{ij}) while creating the particles for the an problem to solve

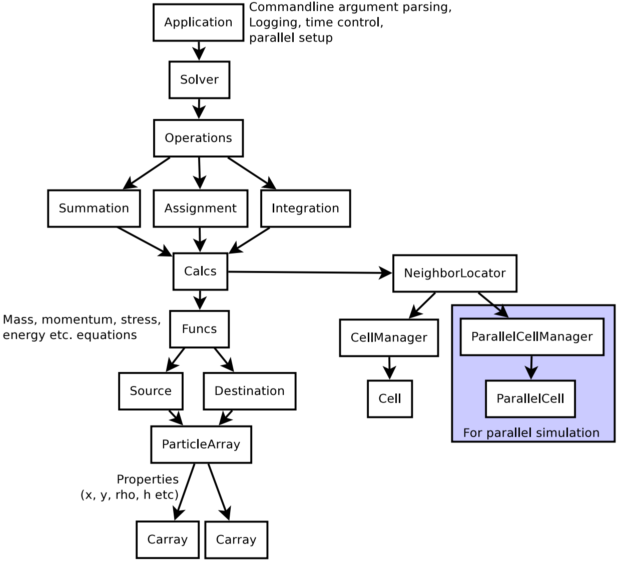
Extension


Application
Commandline argument parsing,
Logging, time control,
parallel setup
(2)
Solver
Operations
Summation
Assignment
Integration
Calcs
NeighborLocator
Mass, momentum, stress,
energy etc. equations
(1)
Funcs
CellManager
ParallelCellManager
Source
Destination
Cell
ParallelCell
For parallel simulation
ParticleArray
Properties
(x, y, rho, h etc)
(3)
Carray
Carray

Parallelization

- Parallelization done using mpi4py bindings
- User requires no knowledge of parallelization
- Automatic load balancing among different processes
- *Application* class implements various switches to handle runs for parallel/serial cases
- command-line option parsing, load distribution, dumping output files

RunSnakeRun

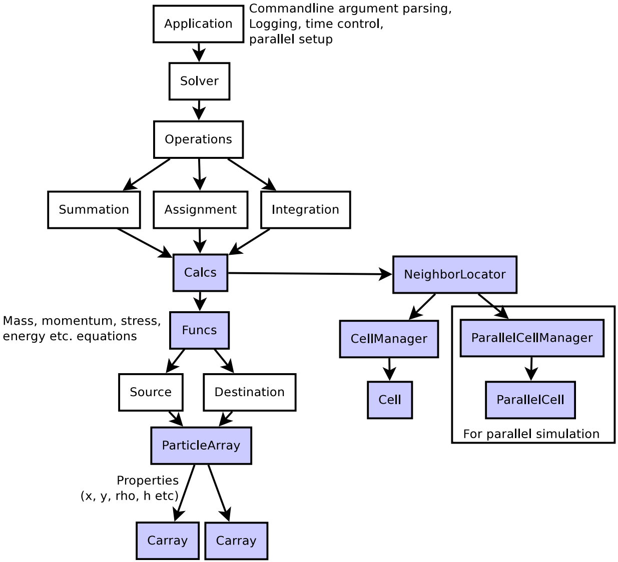


Outline

Cython

- Statically typed C-like code in python-like language
- Compiled to fast native-code
- Mix-n-match C and Python data types
- Performance critical code written in Cython

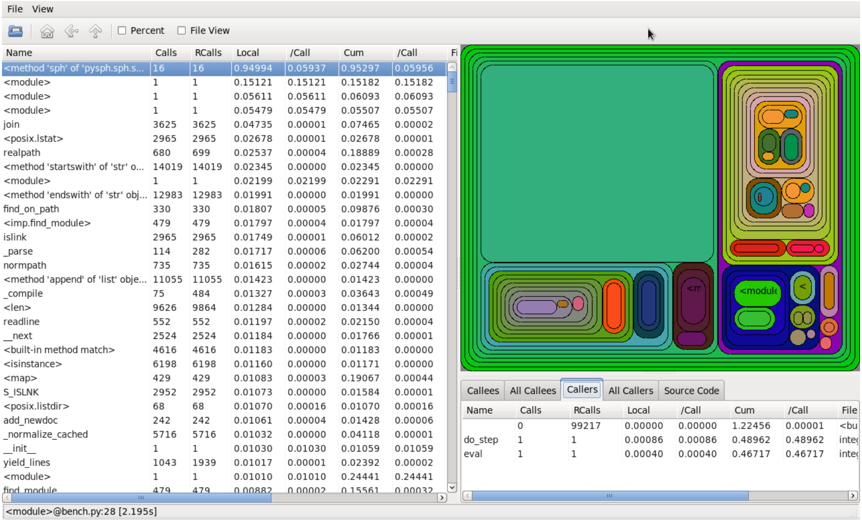
RunSnakeRun



Profiling

- Premature optimization root of all evil
- Never optimize w/o knowing what to optimize
- Works with cython if compiled using profile=True directive
- Visualize profiling data using runsnakerun/KCacheGrind
- Valgrind/SystemTap(DTrace) for tracing python/cython functions
- benchmark critical code sections to keep track of progress

RunSnakeRun



Name	Calls	RCalls	Local	/Call	Cum	/Call
<method 'sph' of 'pysph.sph.s...	16	16	0.94994	0.05937	0.95297	0.05956
<module>	1	1	0.15121	0.15121	0.15182	0.15182
<module>	1	1	0.05611	0.05611	0.06093	0.06093
<module>	1	1	0.05479	0.05479	0.05507	0.05507
join	3625	3625	0.04735	0.00001	0.07465	0.00002
<posix.lstat>	2965	2965	0.02678	0.00001	0.02678	0.00001
realpath	680	699	0.02537	0.00004	0.18889	0.00028
<method 'startswith' of 'str' o...	14019	14019	0.02345	0.00000	0.02345	0.00000
<module>	1	1	0.02199	0.02199	0.02291	0.02291
<method 'endswith' of 'str' obj...	12983	12983	0.01991	0.00000	0.01991	0.00000
find_on_path	330	330	0.01807	0.00005	0.09876	0.00030
<imp.find_module>	479	479	0.01797	0.00004	0.01797	0.00004
islink	2965	2965	0.01749	0.00001	0.06012	0.00002
_parse	114	282	0.01717	0.00006	0.06200	0.00054
normpath	735	735	0.01615	0.00002	0.02744	0.00004
<method 'append' of 'list' obje...	11055	11055	0.01423	0.00000	0.01423	0.00000
_compile	75	484	0.01327	0.00003	0.03643	0.00049
<len>	9626	9864	0.01284	0.00000	0.01344	0.00000
readline	552	552	0.01197	0.00002	0.02150	0.00004
__next	2524	2524	0.01184	0.00000	0.01766	0.00001
<built-in method match>	4616	4616	0.01183	0.00000	0.01183	0.00000
<isinstance>	6198	6198	0.01160	0.00000	0.01171	0.00000
<map>	429	429	0.01083	0.00003	0.19067	0.00044
S_ISLNK	2952	2952	0.01073	0.00000	0.01584	0.00001
<posix.listdir>	68	68	0.01070	0.00016	0.01070	0.00016
add_newdoc	242	242	0.01061	0.00004	0.01428	0.00006
_normalize_cached	5716	5716	0.01032	0.00000	0.04118	0.00001
__init__	1	1	0.01030	0.01030	0.01059	0.01059
yield_lines	1043	1939	0.01017	0.00001	0.02392	0.00002
<module>	1	1	0.01010	0.01010	0.24441	0.24441
find_module	479	479	0.00882	0.00002	0.15561	0.00032

Name	Calls	RCalls	Local	/Call	Cum	/Call	File
	0	99217	0.00000	0.00000	1.22456	0.00001	<bu
do_step	1	1	0.00086	0.00086	0.48962	0.48962	inte
eval	1	1	0.00040	0.00040	0.46717	0.46717	inte

Testing

- Verify correct functioning, refactoring, regressions
- Python unittest module for writing tests
- Nose test collector and runner
- Ned Batchelder's coverage: custom monkey patch for cython function coverage

Coverage report: 45%

Module	*statements*	*missing*	*excluded*	*coverage*
/mnt/data/CourseWare/ddp/pysph/pyx_coverage	92	69	0	25%
main	20	2	0	90%
test	13	1	0	92%
test2	7	1	0	86%
Total	**132**	**73**	**0**	**45%**

coverage.py v3.5a1

Coverage for **main** : 90%

20 statements | 18 run | 2 missing | 0 excluded

```python
import test
import test2

def uncovered():
    print 'yo'
    a = 1+2


def covered1():

    a = test.A()
    a.func_b()
    a.func_c()

def covered2():
    test.func_f()
    test.func_e()

def covered3():
    test2.main()

def main():
    covered1()
    covered2()
    covered3()

if __name__ == '__main__':
    main()

```

« index coverage.py v3.5a1

Coverage for **test** : 92%

13 statements | 12 run | 1 missing | 0 excluded

```python
#cython:profile=True
cdef class A:
    cdef public int a
    cdef public str s
    def __cinit__(self):
        print 'A.__cinit__'

    cdef str func_a(self, int a=1):
        print 'A.func_a'
        self.a += a
        return self.s

    cpdef str func_b(self, int a=2):
        print 'A.func_b'
        self.a += a
        return self.s

    def func_c(self, a=3):
        print 'A.func_c'
        self.a += a
        return self.s, self.a

    cdef uncovered_a(self):
        print 'A.uncovered_a'

cdef class B(object):
    cdef public int a
    cdef public str s
    def __cinit__(self):
        print 'B.__cinit__'

    cdef str func_a(self, int a=1):
        print 'B.func_a'
```

Python for scientific computing

- Fast prototyping, interactive interpreter
- Simplicity, less code, easier contribution
- Many scientific libraries available
- Full-fledged programming language, no restrictions on future ideas
- Great plotting and visualization packages available

Thank You !!!