PySPH: A Python framework for SPH

Prabhu Ramachandran
Chandrashekhkar P. Kaushik

Department of Aerospace Engineering and
Department of Computer Science and Engineering
IIT Bombay

SciPy 2010
Austin, TX, June 30 – July 1, 2010
Outline

Introduction

PySPH

Architecture
Outline

Introduction

PySPH

Architecture
SPH examples
SPH examples
3D
Parallel?
Smoothed Particle Hydrodynamics

- Particle based, Lagrangian
- Gingold and Monaghan (1977), Lucy (1977)

- PDEs
- Complex problems
- Moving geometries
- Free surface problems
- ...
Smoothed Particle Hydrodynamics

- Particle based, Lagrangian
- Gingold and Monaghan (1977), Lucy (1977)
- PDEs
- Complex problems
- Moving geometries
- Free surface problems
- ...
SPH basics

- The SPH approximation

\[ f(r) = \int f(r') W(r - r', h) \, dr' \]

- \( W(r - r', h) \): kernel, compact support
- \( h \): size of the kernel
SPH basics
SPH basics

Nearest neighbors

kh
SPH basics . . .

- Derivatives: transferred to the kernel
- Lagrangian

- PDE $\rightarrow$ ODE
Scale-up for larger problems requires parallelization
Motivation

Reproducible and open research
Outline

Introduction

PySPH

Architecture
PySPH

- SPH + Parallel + Python
- pysph.googlecode.com
- Open Source (BSD)
Requirements and installation

- Python, setuptools
- numpy, Cython-0.12, mpi4py-1.2
- Mayavi-3.x (optional)
- Standard Python package install
High-level solver outline

1. Create and Setup a Solver
2. Create particles which represent physical entities involved in the simulation
3. Perform any additional operations (auxilliary to the main solver)
4. Start the solver
Serial Dam break

```python
solver = FSFSolver(time_step=0.0001, 
total_simulation_time=10., 
kernel=CubicSpline2D())

# create the two entities.
dam_wall = Solid(name='dam_wall')
dam_fluid = Fluid(name='dam_fluid')

# The particles for the wall.
rg = RectangleGenerator(...)
dam_wall.add_particles(rg.get_particles())
solver.add_entity(dam_wall)
```
Serial Dam break ... 

# Particles for the left column of fluid.
rg = RectangleGenerator(...)
dam_fluid.add_particles(rg.get_particles())
solver.add_entity(dam_fluid)

# start the solver.
solver.solve()
Parallel Dam break

```python
solver = ParallelFSFSolver(
    time_step=0.0001,
    total_simulation_time=10.,
    kernel=CubicSpline2D())

# Load particles in proc with rank 0.
```
Outline

Introduction

PySPH

Architecture
Software architecture

- Particle kernel
- SPH kernel
- Solver framework
- Serial and Parallel solvers
Software architecture

Solver Framework
(Python & Cython)

Solvers

SPH (Cython)

NNPS (Cython)

Cell Manager (Cython)

Particle Arrays (Cython)

C–Arrays (Cython)
Particle kernel

- C-arrays: numpy-like but faster and resizable
- ParticleArray: arrays of properties
- NNPS (Nearest Neighbor Particle Search)
- Cell, CellManager
- Caching
SPH kernel

- SPH particle interaction: interaction between 2 SPH particles
- SPH summation: interaction of all particles
Solver framework

- Entities (made of particles)
- Solver components (do various things)
- Component manager (checks property requirements)
- Integrator
- Basic Solver
Parallel solver

- Parallel cell manager
- Parallel solver components
Parallel solver

Serial case

Solver components

CellManager

Particles

Parallel case

Processor 1

Solver components

ParallelCellManager

Particles

Processor 2

Solver components

ParallelCellManager

Particles
Challenges

- Particles move all over
- Fixed partitioning will not work
- Scale up
- Automatic load balancing?
Terminology: Region

Region 1

Region 2

P1

P2
Terminology: Domain decomposition
Approach

- Domain decomposition: Cells
- Cells: dynamically created/destroyed
- Processors manage Regions
- Cells moved to balance load
Load Balancing

Donate Cells

- Boundary cells
- Cells with least number of local neighbors
Load balancing
Efficiency

The graph shows the efficiency of a system as the number of processors increases. Two curves are plotted, one for 1e06 particles and another for 500000 particles. As the number of processors increases, the efficiency decreases for both datasets.
Efficiency

![Efficiency graph](image)
Current capabilities

- Fully automatic, load balanced, parallel framework
- Relatively easy to script
- Good performance
- Relatively easy to extend

- Free surface flows
Immediate improvements

- Solver framework redesign
- More documentation
- Reduce dependency on TVTK for easier installation
- Improved testing on various platforms
- Gas dynamics (coming soon)
- Solid mechanics (next year)
Thank you!