PySPH: A Python framework for SPH

Kunal Puri*, Prabhu Ramachandran*, Pankaj Pandey, Chandrashekhar Kaushik, Pushkar Godbole*
* Department of Aerospace Engineering, Indian Institute of Technology Bombay
Mumbai, India

kunal.r.puri@gmail.com

***Abstract*—We present an open source, object oriented framework for Smoothed Particle Hydrodynamics called PySPH. The framework is written in the high level, Python programming language and is designed to be user friendly, flexible and application agnostic. PySPH supports distributed memory computing using the message passing paradigm and (limited) shared memory like parallel processing on hybrid (CPU + GPU) machines using OpenCL. In this work, we discuss the abstractions for an SPH implementation and the resulting design choices that resulted in the development of PySPH.**

I. Introduction

Smoothed Particle Hydrodynamics (SPH) is a meshless method employing the use of points/particles as discretization entities for the numerical simulation of continuum mechanics problems. The governing PDEs are transformed into ordinary differential equations for particle field variables, constraining individual particles to interact with near neighbors. The subsequent motion of particles with the local material velocity renders the method Lagrangian. This model has been successfully applied to large deformation continuum problems in astrophysics, solid-mechanics and incompressible fluid flow. We refer the interested reader to [1, 2, 3, 4] and the references cited therein for a more detailed exposition on SPH.

The development of SPH algorithms for different problems leads us to the realization that while the applications may be diverse, there exist a fundamental group of operations for any SPH implementation. These include a mechanism to access particle properties, fast neighbor queries to get the interaction lists, discretizations to compute the forces and a time integrator to update the solution. Of these, the force computation represents the "physics", or the particular application area of SPH. Supplementary operations like I/O and post-processing of results are common to numerical simulations by any technique. Fundamental operations notwithstanding, application nuances introduce subtle changes to the algorithm that necessitate a careful design of a general SPH implementation. Further challenges arise when the algorithm is to be implemented in parallel across distributed memory machines and possibly on modern heterogeneous architectures involving hybrid CPUs and Graphics Processing Units (GPUs).

In this work, we describe an open-source, object oriented framework for SPH simulations called PySPH, which is designed to be flexible, user friendly and application agnostic. PySPH arose out of the need for a general purpose SPH tool that hides the implementation details like parallelism from the casual user, yet enables quick prototyping and extension to new problems. With PySPH, the user can choose between solvers for gas-dynamics, solid mechanics and incompressible free surface flows. The solver supports variable smoothing lengths commonly encountered in astrophysical simulations, dynamic and repulsive boundary conditions for free surface flows and multi-stage time integrators. The framework supports distributed memory parallelism using the message passing paradigm (MPI [5]) and limited shared memory like parallelism on GPUs and/or CPUs using OpenCL [6]. While open source, parallel implementations of SPH and particle methods are not new ([7, 8, 9]), users should find PySPH an attractive option to use and possibly extend to their SPH applications. PySPH 0.9beta is released under the BSD license and is publically available at http://pysph.googlecode.com Hereafter, we will drop the PySPH version specification in favor of brevity.

This paper is outlined as follows. Section II explains our motivation behind the development of PySPH by considering an abstract formulation for a continuum problem and the corresponding abstractions required for an SPH implementation. In Section III, we discuss the philosophy adopted in the design of PySPH in light of these abstractions. In Section IV, we provide example results to demonstrate the current capability of PySPH. We conclude this work in Section V with a summary and an outlook to future versions of PySPH.

II. Motivation

Consider the general continuum equations expressing conservation of mass, momentum and energy as

$$\frac{d\rho}{dt} = -\rho\nabla\cdot(\boldsymbol{v}) \tag{1a}$$

$$\frac{dv^i}{dt} = \frac{1}{\rho}\frac{\partial\sigma^{ij}}{\partial x^i} + g^i \tag{1b}$$

$$\frac{de}{dt} = -\frac{P}{\rho}\nabla\cdot(\boldsymbol{v}) \tag{1c}$$

where, $\frac{d}{dt} = \frac{\partial}{\partial t} + \boldsymbol{v}\cdot\nabla(*)$ is the material derivative, g^i is the component of body force per unit mass and $\sigma^{ij} = -P\delta^{ij} + S^{ij}$ is the stress tensor composed of the deviatoric component S^{ij} and the isotropic pressure P. For solid mechanics, we require a stress model like the Hooke's law, coupling the deviatoric stress to velocity gradients. For compressible problems, an equation of state $P(\rho, e)$ is needed to couple the pressure P to the density ρ and thermal energy e. The equations

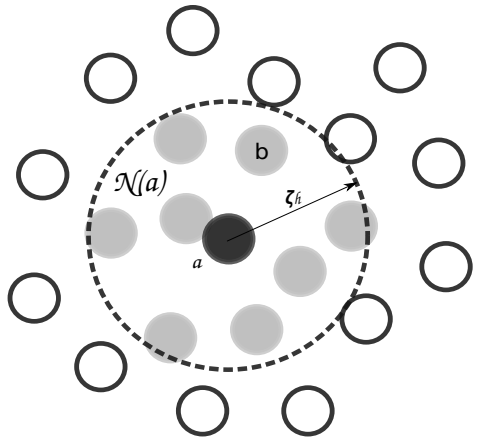

Fig. 1. Set of neighbors for a given particle.

(1) could describe for example, high Mach number flows in compressible gas dynamics [4], high strain solid mechanics [10, 11] or viscous incompressible fluid flow in open and closed channels [2]. A general SPH discretization for (1) can be written as

$$\frac{d\rho_a}{dt} = \sum_{b\in\mathcal{N}(a)} m_b \mathcal{D}_{ab}(\boldsymbol{v}) \cdot \nabla_i W_{ab}(h) \tag{2a}$$

$$\frac{dv^i}{dt} = \sum_{b\in\mathcal{N}(a)} m_b \mathcal{F}_{ab}(\rho, P, \boldsymbol{v}) \frac{\partial}{\partial x^i} W_{ab}(h) \tag{2b}$$

$$\frac{de_a}{dt} = \sum_{b\in\mathcal{N}(a)} m_b \mathcal{E}_{ab}(\rho, P, \boldsymbol{v}) \cdot W_{ab}(h) \tag{2c}$$

where the evolution of particle properties is expressed through interactions over it's nearest neighbors. We denote by $\mathcal{N}(a)$ as the set of neighbors for a given particle as shown in Fig.1. The interaction terms $\mathcal{D}$, $\mathcal{F}$ and $\mathcal{E}$ can be thought to represent "fluxes" of mass, momentum and energy that contribute to local changes in particle properties. While not being exhaustive, we limit the discussions in this work to discretizations represented by (2). Note that we permit the possibility of a "summation" operation wherein a certain field quantity is calculated as

$$f_a = \sum_{j\in\mathcal{N}(a)} \mathcal{R}_{ab} W_{ab}(h) \tag{3}$$

Important examples of this type of operation are "summation density" and kernel normalization used in astrophysical and fluid flow problems respectively. The SPH discretization process has transformed the system of PDEs (1) into a system of ordinary differential equations (2) for the field variables density, velocity and thermal energy.
With this discretization as a reference, we proceed to enlist the necessary abstractions required for a numerical implementation.

A. Particle container

The principle discretization entity in SPH being points/particles, the implementation must have data structures to represent arbitrary collections of particles and perform operations on them. A material (species) can be represented as a collection of arrays (C++ *vectors*) describing the required properties. For example, a kinematic description of a fluid requires the position coordinates, mass and velocity components, making a total of 7 arrays. For multi-material problems (fluid, solid), we have the choice of using a collated array along with an additional array of markers identifying the material, or create separate container objects for the different species. For distributed computations using MPI, the container must enable add/remove operations for local and remote particles. Finally, for GPU based simulations, we require that the container implement functions to transfer particle data back and forth from the GPU compute device.

B. Nearest neighbor queries

Each particle needs to know a list of it's nearest neighbors to efficiently carry out the summations in (2). As shown in Fig. 1, the set of neighbors is calculated as a ball of radius ζh about a target particle. The smoothing length h may, in general be variable across particles. It is typically adaptively tuned to the local particle density for SPH simulations of compressible gas dynamics [4, 12], while kept constant for simulations involving incompressible fluids. Spatial indexing algorithms like the linked list [13], radix sort [14] and oct-trees [4] can be used to generate the neighbor lists. While the tree based method is hierarchical in nature, variable smoothing lengths pose implementation challenges to the linked list and radix sort algorithms, particularly on the GPU. The large dynamic range of the smoothing lengths across a domain can lead to unnecessarily large neighbor lists in regions of high density.

C. Interaction physics

The summations in (2) define the evolution of particle field properties as a discrete analogue to (1). The interaction terms $\mathcal{D}$, $\mathcal{F}$ and $\mathcal{E}$, come about from the particular discretization of the continuum equations and may include viscous stress or artificial viscosity needed to stabilize the scheme. We focus our discussion in an application agnostic manner and hence refer the reader to [1, 2, 3, 4] for a lucid description on SPH for specific application areas. Once the neighbor lists are generated for a given particle distribution, computation of the accelerations is essentially a local process. This computation may be off-loaded to an available GPU for efficiency.

D. Integrator

After computation of the particle interactions, an integrator is used to update the solution in time. For stability, a global time step must be chosen from across all particles in the domain. Multi-stage integrators require multiple force evaluations and additional storage. Movement of particles in a sub time step must be handled carefully in parallel as particles may have crossed processor boundaries. Commonly used integrators in SPH are the second order symplectic integrators [1] and leap-frog two stage integrator [2].

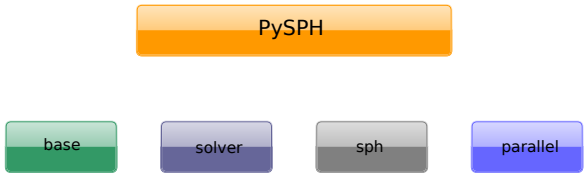

Fig. 2. PySPH subpackages.

E. Miscellaneous

Given the aforementioned abstractions that comprise the core SPH operations, we require additional tools and routines to easily construct solvers, save/restart solution data and visualize the results. Other desirable features are the capability of remotely monitoring a solution, pausing a solution to query field data and changing parameters/functions during the course of a simulation. Such tools are discussed in the context of PySPH in the following section.

III. The design of PySPH

To the casual user, PySPH is written in Python but it is really blend of different languages and libraries. Cython [15] is used to generate C/C++ extensions for the performance critical parts like neighbor searching and pair-wise force computations. Optionally, we use Python's string processing capabilities to generate OpenCL code on the fly and dynamically compile it using PyOpenCL [6]. We would like to highlight at this point that this capability is in it's infancy and only a subset of the functions have OpenCL portability. MPI is used for distributed memory computations through the Python binding mpi4py [5]. Currently under development, the Zoltan data management library [16], using a custom wrapper for Python is used for dynamic load balancing. This feature will be available in the next release and an example demonstrating this feature is shown in Section IV-C. The use of Python is favored to facilitate readability and user friendliness. In the following, we describe the components of PySPH in moderate detail. We use bold type to denote Python classes and italics to denote Python constructs like modules, methods and functions.

A. PySPH

PySPH is broken up into four subpackages as shown in Fig. 2 The *base* subpackage defines the data structures for the particle arrays and nearest neighbor queries on particle sets. The *sph* subpackage defines all aspects related to the "physics" of a particular problem. For instance, it is here that the interaction functions for the myriad problems are defined. The *solver* subpackage is the entry point of PySPH for the user. This package provides an intuitive way to construct a solver from the available functions, choose a time stepping scheme, set solver parameters like I/O frequency and begin the simulation. Finally, the *parallel* subpackage is responsible for the domain decomposition, load balancing and computation of remote neighbors using MPI. The OpenCL support is integrated as an option into the packages in an "on demand"

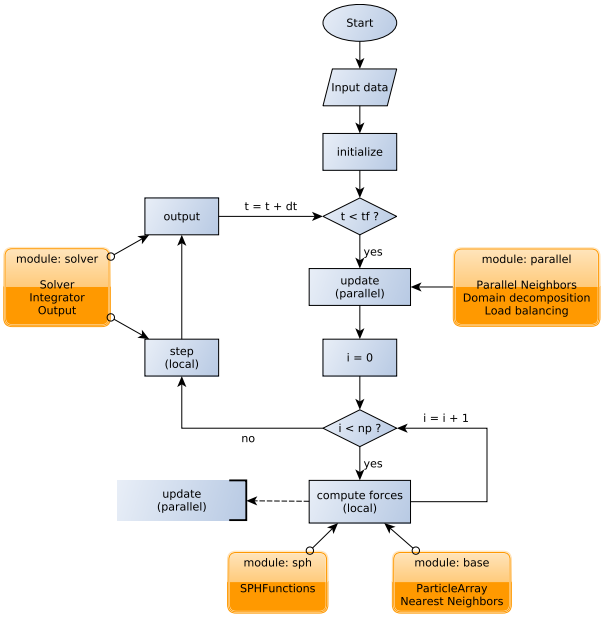

Fig. 3. Flowchart for the SPH algorithm.

fashion. For example, the user may specify that the neighbor lists and force computations be carried out on an OpenCL compute device or only the neighbor lists. These options are set upon construction from the *solver*. We emphasize again that this support is only partial and a future release will include full OpenCL support. The flowchart for the SPH algorithm highlighting the roles of the different subpackages is shown in Fig.3. In the following section, we briefly describe the different subpackages in the context of the abstractions introduced in Section II.

B. Subpackage: base

In PySPH, the basic container for a homogeneous set of particles is a **ParticleArray**. Each ParticleArray is a collection of statically typed property arrays called a **Carray**. The number and type of the Carrays are user defined but cannot be changed during the course of a simulation. A Carray (implemented in Cython) as the name suggests, is a contiguous block of memory to hold an individual particle property like position, velocity, smoothing length etc. Using this underlying data representation, MPI communication of individual arrays and allocation on an OpenCL compute device is trivial. A collection of ParticleArrays in PySPH is called **Particles** and this in turn is managed by a **DomainManager** to handle various types of nearest neighbor queries. The DomainManager returns appropriate **NeighborParticleLocators**

for pairs of ParticleArrays. In a given pair, the ParticleArray for which the neighbors are sought is called "destination" and the ParticleArray from where the neighbors are found is called "source". The DomainManager implements a method *update* which seamlessly updates the particle data and re-computes the neighbor lists either in serial or parallel. While seeming to obfuscate, these layers of abstractions are necessary to implement the algorithms in a hybrid (CPU + GPU) manner. The currently implemented hybrid DomainManagers in PySPH are the **RadixSortManager** and **LinkedListManager** for their respective indexing mechanisms. The DomainManager interfaces with the *parallel* subpackage to distribute particle data across processors and compute remote neighbors.

C. *Subpackage:* sph

The *sph* subpackage is where the SPH functionality is implemented. This includes the interaction functions **SPHFunction**, which operate on a fixed source-destination pair and the **SPHCalc** which groups all such functions for a given destination ParticleArray. The force computation is essentially a local process and does not require information from remote processors. For OpenCL support, each function returns a Python string containing valid OpenCL code computing the interaction between a source and destination particle. This string is used for run time compilation of the OpenCL kernel and executed on the compute device.
PySPH defines functions for compressible gas dynamics using the MPM [1], ADKE [12] and GSPH [17] schemes, incompressible fluid flows using the weakly compressible approach [18] with dynamic and Monaghan type repulsive boundary particles [8, 2] and elastic solid mechanics [11]. The prospective user will have to add a function to this package to extend the capability of PySPH. A new function can be added for prototyping in pure Python albeit with a performance penalty.

D. *Subpackage:* solver

The *solver* subpackage defines the **Application** class which is the main entry point for PySPH. Through the Application, the user may change any feature of the SPH solution. For example, one may opt for neighbor searching using a radix sort algorithm or choose between the different load balancing functions. Another important feature is the construction of a **Solver** object that marshals the simulation. The *solver* subpackage abstracts out a general SPH expression as an **SPHOperation**. The SPHOperation assumes some property ("lhs") is being updated through the action ("rhs") of an assignment (as in equation of state), summation (summation density) or integration (accelerations). This operation acts on a set of "destination" **ParticleTypes** (the "lhs") and computes the action from an optional list of "source" **ParticleTypes**. These SPHOperations are added to a Solver to define a simulation. This mechanism provides an intuitive and powerful way to specify operations between different entities and define a simulation. We use an example to elucidate this. Consider the driven cavity problem with the schematic as shown in

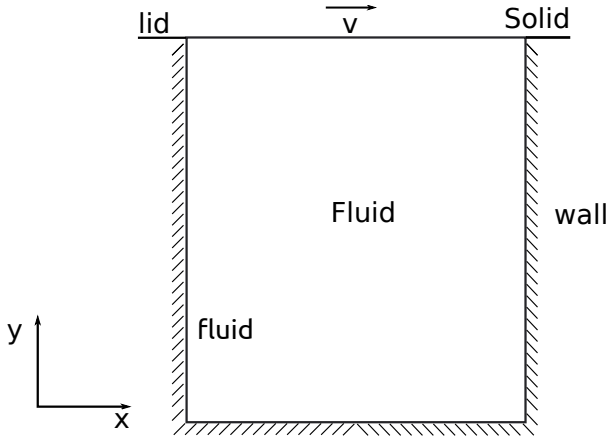

Fig. 4. Schematic for the lid driven cavity problem.

Fig.4 For the discretization we have 3 named ParticleArrays: "fluid" of type **Fluid**, "wall" of type **Solid** and "lid" of type **Solid**. The no-penetration boundary condition is enforced by using repulsive particles. The specification of this operation in PySPH is

```
solver.add_operation(SPHIntegration(
    MonaghanBoundaryForce(srcs=["fluid"], dsts=["
        wall","lid"]),
    updates=["u", "v", "w"],
    id="bc")
  )
```

which adds the **MonaghanBoundaryForce** as an **SPHIntegration** operation, integrating velocity components by considering the "fluid" as a destination and "wall" and "lid" as sources. New solvers are constructed in PySPH using this procedure. PySPH defines "canned" solvers for gas-dynamics, elasticity and free surface flows which the user may modify by adding, deleting or inserting operations as desired.

E. *Subpackage:* parallel

The *parallel* subpackage is responsible for domain decomposition and dynamic load balancing required in SPH simulations on distributed memory machines. In a parallel implementation, each processor is assigned a set of particles ("local" particles) to evolve. Moreover, to carry out pairwise interactions across processor boundaries, ghost particle data ("remote" particles) must be shared as a buffer or halo region around a neighboring processor as shown in Fig.5. The assignment of particles to processors must ensure a balanced workload and a minimal surface area to reduce the time spent communicating ghost/buffer particles. We use a custom Python wrapper for Zoltan [16] to achieve the desired particle distribution and optimal load balance across processors through the **ParallelManager**. The **DomainManager** from the *base* subpackage interfaces with the ParallelManager to distribute particles and compute remote neighbors. The *parallel* subpackage is entirely optional and users can use the same script to run the examples in either serial or parallel.

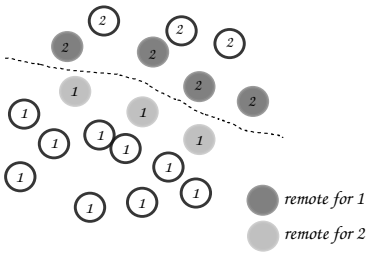

Fig. 5. Local and remote particles for an imaginary particle distribution. Particles assigned to a processor are "local" while those shared for neighbor requirements are "remote"

F. Miscellaneous: viewer

Post processing of CFD simulations is equally important to assess the quality and gain insights into the obtained results. We use the VTK based viewer, Mayavi [19] as an interactive viewer for PySPH. A screen-shot of the viewer is shown in Fig. 6. Apart from the general visualization capabilities of

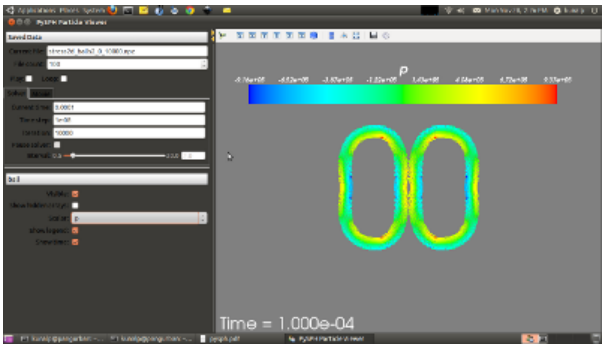

Fig. 6. Interactive visualization of the collision of two elastic balls using the Mayavi viewer.

Mayavi, the viewer supports an XML-RPC interface through which we can remotely monitor and interrupt simulations. For example the user might wish to change the print frequency or the numerical value of a viscosity used in a simulation. Alternatively, the particle data being stored as Python-accessible arrays, can be visualized and operated upon using the standard scientific computing tool-chain (numpy, scipy, matplotlib) for Python.

IV. Examples

In this section, we present results from some of the available solvers for the compressible Euler equations, solid mechanics and incompressible free surface flows to demonstrate some of the capabilities of PySPH. The examples are not exhaustive and the prospective user should look in the *examples* directory for a more through suite.

A. Compressible Euler equations

Simulations for the Euler equations of gas dynamics are important from the general CFD perspective in that SPH is a genuinely multidimensional technique and as such, should provide an attractive alternative to finite volume schemes. For the example, we consider the case of a cold gas brought to rest against a wall, usually referred to as the Noh's cylindrical implosion problem. A shock front forms at the wall that travels upstream, bringing the cold gas to rest. The post shock state is a hot, dense gas with zero velocity. The simulation is setup in 2D and we use the ADKE [12] scheme with adaptive

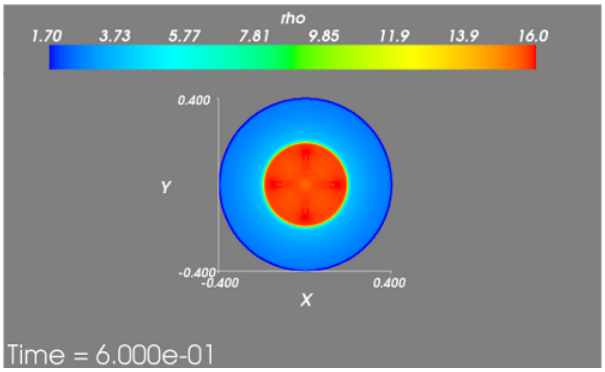

Fig. 7. Density map for the Noh's cylindrical implosion problem using the ADKE scheme.

smoothing lengths. Fig.7 shows the particles colored with density. We can see a cylindrical shock front across which the density changes sharply. The analytical maximum for the density at the shock front is 16 which is reproduced by the SPH simulation.

B. Solid mechanics

The next eaxmple is a high strain solid mechanics problem in which a metallic projectile impacts at high velocity onto a plate. Figure.8 shows a snapshot when the projectile has partly

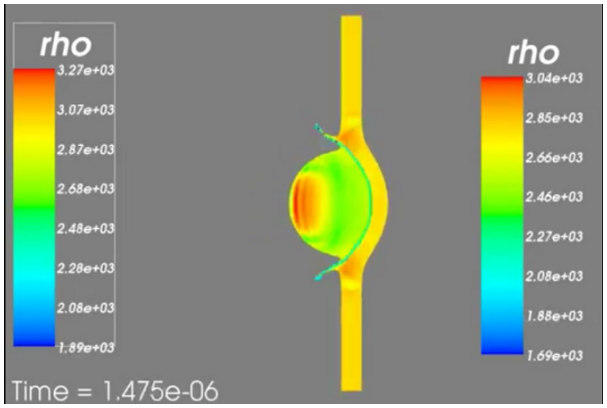

Fig. 8. 2D simulation of a metallic projectile impacting a plate.

penetrated the plate. The simulation uses a stress model with the Mie Gruneisen equation of state and ≈ 15000 particles.

C. Free surface flows

Free surface flows are the principle area of application of SPH. In such problems, an incompressible fluid flows

under the action of gravity in an open vessel. The fluid may encounter obstacles in it's path as for example, a wave striking a lighthouse tower. Figure.9 shows the results of a $2D$ dam

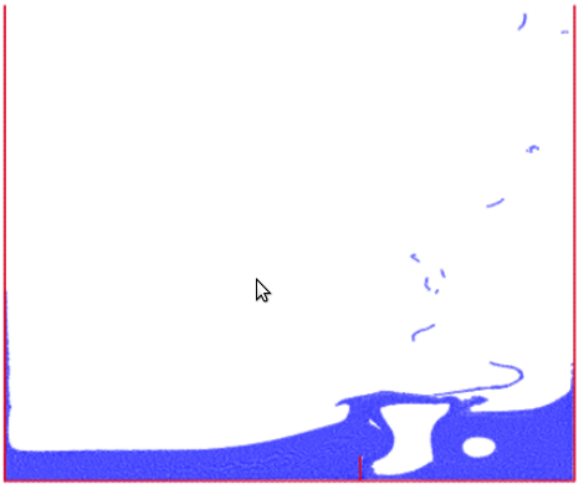

Fig. 9. 2D dam break in the presence of an obstacle. The simulation uses 70000 particles with RIB load balancing on 4 processors.

break problem as it encounters a vertical obstacle. We use the weakly compressible SPH approach with dynamic boundary particles [8]. We use the Monaghan style artificial viscosity and a 2^{nd} order, modified predictor corrector integrator for the time-stepping. The simulation uses ≈ 70000 particles with Recursive Inertial Bisection (RIB) based load balancing on 4 processors. We would like to point out that the parallel run was made with the latest version of PySPH using the Zoltan library for dynamic load balancing.

V. Summary and an outlook to the future

We discussed the development and design philosophy of PySPH, an open source, parallel framework for Smoothed Particle Hydrodynamics that respects the diversity in SPH by being application agnostic. PySPH implements abstractions for the SPH algorithm that make it easy for the user to construct new solvers and functions. We demonstrated the capability of PySPH as applied to the compressible Euler equations, incompressible fluid flows and solid mechanics. The package is released under the BSD license and is available from http://pysph.googlecode.com.

A. Issues to be addressed in a future release

We would like to point out that PySPH is a work in progress and the discussion in this work was related to PySPH version 0.9beta. The design paradigms introduced serve as a proof of concept, validating the choice of the numerical tools with respect to the implementation. There are however, important issues yet to be tackled in a future version which are enlisted below.

- OpenCL: Currently, this feature is in it's infancy with support for a subset of the functions and limited to problems with a single particle array. A future version of PySPH would have full OpenCL compatibility on single and multiple GPU platforms.

- Parallel performance: The discerning reader would have noticed the absence of any benchmarks for the parallel implementation. The reason for this is because we found that the current implementation does not scale satisfactorily to large problems. To this end, we have re-written the parallel subpackage to interface with a custom wrapper for the Zoltan data management library [16]. Integration of the Zoltan wrapper within PySPH is under progress and is planned for the next release.

References

[1] D. Price, "Smoothed particle hydrodynamics and magnetohydrodynamics," *Journal of Computational Physics*, vol. 231, pp. 759–794, 2012.

[2] J. Monaghan, "Smoothed Particle Hydrodynamics and it's Diverse Applications," *Annual Review of Fluid Mechanics*, vol. 44, pp. 323–346, 2012.

[3] J. Monaghan, "Smoothed particle hydrodynamics," *Reports on Progress in Physics*, vol. 68, pp. 1703–1759, 2005.

[4] Volker Springel, "Smoothed Particle Hydrodynamics in Astrophysics," *Annual reviews of Astronomy and Astrophysics*, vol. 48, pp. 391–430, 2010.

[5] Lisandro Dalcin, "MPI for Python," 2010.

[6] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih, "PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation," *Parallel Computing*, vol. 38, no. 3, pp. 157–174, 2012.

[7] V. Springel, "The Cosmologhical simulation code GADGET-2," *Monthly Notices of the Royal Astronomical Scociety*, vol. 364, pp. 1105–1134, 2005.

[8] M. Gomez-Gesteira and B.D. Rogers and A.J.C. Crespo and R.A. Dalrymple and M. Narayanaswamy and J.M. Dominguez, "SPHysics development of a free-surface fluid solver Part 1: Theory and formulations," *Computers and Geosciences*, vol. 48, no. 0, pp. 289 – 299, 2012.

[9] D. Valdez-Balderas, J. M. Domnguez, B. D. Rogers, and A. J. Crespo, "Towards accelerating smoothed particle hydrodynamics simulations for free-surface flows on multi-GPU clusters," *Journal of Parallel and Distributed Computing*, no. 0, pp. –, 2012.

[10] P. Randles and L. Libersky, "Smoothed particle hydrodynamics: Some recent improvements and applications," *Computer Methods in Applied Mechanical Engineering*, vol. 139, pp. 375–408, 1996.

[11] J.P. Gray and J. J. Monaghan and R.P. Swift, "SPH elastic dynamics," *Computer Methods in Applied Mechanics and Engineering*, vol. 190, pp. 6641–6662, 2001.

[12] L. D. G. Sigalotti, H. Lopex, A. Donoso, E. Siraa, and J. Klapp, "A shock capturing sph scheme based on

adaptive kernel estimation.," *Journal of Computational Physics*, vol. 212, pp. 124–149, 2006.

[13] Allen, M. P. and Tildesley D.J., *Computer Simulation of Liquids* . Clarendon Press, 1989.

[14] Juraj Onderik and Roman Ďurikovič, "Efficient Neighbor Search for Particle Based Fluids," *Journal of the Applied Mathematics, Statistics and Informatics*, vol. 2, no. 3, 2007.

[15] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. Seljebotn, and K. Smith, "Cython: The best of both worlds," *Computing in Science Engineering*, vol. 13, no. 2, pp. 31 –39, 2011.

[16] Karen Devine and Erik Boman and Robert Heaphy and Bruce Hendrickson and Courtenay Vaughan, "Zoltan Data Management Services for Parallel Dynamic Applications," *Computing in Science and Engineering*, vol. 4, no. 0, pp. 90–97, 2002.

[17] S.-I. Inutsuka, "Reformulation of Smoothed Particle Hydrodynamics with Riemann Solver," *Journal of Computational Physics*, vol. 179, pp. 238–267, 2002.

[18] Moncho Gomez-Gesteria and Benidict D. Rogers and Robert. A Dalrymple and Alex J. Crespo, "State-of-the-art classical SPH for free-surface flows," *Journal of Hydraulic Research*, vol. 84, pp. 6–27, 2010.

[19] Ramachandran, P. and Varoquaux, G, "Mayavi: 3D Visualization of Scientific Data," *IEEE Computing in Science Engineering*, vol. 13, no. 2, pp. 40 –51, 2011.