Technical Report TR-2013-4

A High Performance Computing (HPC) approach to the Smoothed Particle Hydrodynamics (SPH) method

Arman Pazouki, Dan Negrut

November 8, 2013

Abstract

This document describes the fundamentals of Smoothed Particle Hydrodynamics (SPH) for fluid flow simulations. A High Performance Computing (HPC) relying on GPU implementation is described and tested. Further descriptions and tests can be found in our peer-reviewed publications.

The Smoothed Particle Hydrodynamics method

An in-depth discussion of the SPH method and recent developments can be found in [1-3]. Here we only highlight the fundamentals of the methodology required for fluid flow simulation. Smoothed Particle Hydrodynamics is a Lagrangian method that probes the fluid domain at a set of moving markers, each of which has a specified domain of influence. In this document, the term "marker" denotes an SPH discretization point that has an associated kernel function with a compact support, see Figure 1. The choice of kernel function W is not unique. In this work,

$$W(r,h)=\frac{1}{4\pi h^3}\times\begin{cases}(2-q)^3-4(1-q)^3, & 0\le q<1\\ (2-q)^3, & 1\le q<2,\\ 0, & q\ge 2\end{cases} \qquad (1)$$

where h is the kernel function's characteristic length and $q\equiv|r|/h$. The radius of the kernel function, κh, is proportional to the characteristic length, where $\kappa=2$ for the kernel function defined by eq. (1).

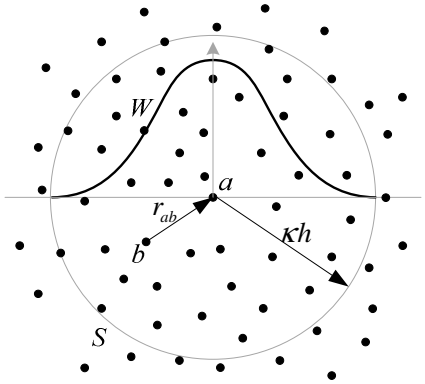

Figure 1. Illustration of the kernel, W, and support domain, S. SPH markers are shown as black dots. For 2D problems the support domain is a circle, while for 3D problems it is a sphere.

In terms of notation, in what follows, ρ and μ are the fluid density and viscosity, respectively, $\mathbf{v}$ and p are flow velocity and pressure, respectively, and m is the mass associated with an SPH marker. The continuity and momentum equations, given respectively by [4]

$$\frac{\mathrm{d}\rho}{\mathrm{d}t} = -\rho\nabla.\mathbf{v} \tag{2}$$

and

$$\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = -\frac{1}{\rho}\nabla p + \frac{\mu}{\rho}\nabla^2\mathbf{v} + \mathbf{f}\,, \tag{3}$$

are discretized within the framework of SPH as

$$\frac{\mathrm{d}\rho_a}{\mathrm{d}t} = \rho_a \sum_b \frac{m_b}{\rho_b}\left(\mathbf{v}_a - \mathbf{v}_b\right).\nabla_a W_{ab} \tag{4}$$

and

$$\frac{\mathrm{d}\mathbf{v}_a}{\mathrm{d}t} = -\sum_b m_b \left(\left(\frac{p_a}{{\rho_a}^2} + \frac{p_b}{{\rho_b}^2}\right)\nabla_a W_{ab} + \Pi_{ab} \right) + \mathbf{f}_a\,, \tag{5}$$

where

$$\Pi_{ab} = -\frac{(\mu_a + \mu_b)\mathbf{r}_{ab}.\nabla_a W_{ab}}{\bar{\rho}_{ab}{}^2 (r_{ab}^2 + \varepsilon \bar{h}_{ab}^2)}\,\mathbf{v}_{ab} \tag{6}$$

imposes the viscose force based on the discretization of the '∇^2' operator; the subscripts a and b denote the indices of two interacting markers, and the summation is over all markers within the support domain of marker a (Figure 1). We evaluated several definitions [1, 4] for the artificial viscosity and discretization of '∇^2' in conjunction with the simulation of transient

Poiseuille flow and concluded that Π_{ab} of Eq. (6) led to the most accurate results in the widest range of Reynolds numbers. Moreover, Eq. (6) replaces the tuning parameters used in artificial viscosity [1] with physics-based fluid parameters.

The pressure p is evaluated using an equation of state [1]

$$p = \frac{c_s^2 \rho_0}{\gamma} \left\{ \left(\frac{\rho}{\rho_0} \right)^{\gamma} - 1 \right\}, \tag{7}$$

where ρ_0 is the reference density of the fluid, γ tunes the stiffness of the pressure-density relationship, and c_s is the speed of sound. To increase the simulation efficiency, c_s can be adjusted depending on the maximum speed of the flow, $V_{\max}$, to keep the flow compressibility below any arbitrary value. In the current work, we chose $\gamma = 7$ and $c_s = 10 V_{\max}$ which allows 1% flow compressibility [1].

Finally, note that the fluid flow equations (2) and (3) are solved in conjunction with a third equation that links the time derivative of the generalized positions to the velocity of the SPH markers:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{v}. \tag{8}$$

The SPH discretizations proposed in Eqs. (4) and (5) as well as the form of the kernel function W are not unique. The choices reported led to the most robust solutions and accurate results.

Compared to Eq. (4), which evaluates the time derivative of the density, the original SPH summation formula calculated the density according to

$$\rho_a = \sum_b m_b W_{ab}. \tag{9}$$

Equation (4) was preferred to Eq. (9) since it produced a smooth density field and worked well for markers close to the boundaries, i.e., the free surface, solid, and wall. However, Eq. (4) does not guarantee consistency between a marker's density and associated mass and volume [4-6]. Using Eq. (9) has problems of its own: the density field can experience large variations, particularly close to the boundary. The approach adopted here is to combine the two methods in

a so-called "density re-initialization technique," [7] in which Eq. (4) is implemented at each time step, while Eq. (9) corrects the mass-density inconsistency every n time steps. The results reported herein were obtained with $n=10$. Other techniques, such as the Moving Least Squares method or a normalized version of Eq. (9), could alternatively be used to address the aforementioned issues [7, 8].

The so-called XSPH correction was implemented to further improve the accuracy of the numerical solution; it prevents extensive marker interpenetration and enhances incompressibility of the flow [9]. This correction takes into account the velocity of neighboring markers through a mean velocity evaluated within the marker's support as

$$\left\langle \mathbf{v}_a \right\rangle = \mathbf{v}_a + \Delta\mathbf{v}_a , \tag{10}$$

where

$$\Delta\mathbf{v}_a = \varepsilon \sum_b \frac{m_b}{\overline{\rho}_{ab}} (\mathbf{v}_b - \mathbf{v}_a) W_{ab} , \tag{11}$$

and $0 \le \varepsilon \le 1$ adjusts the contribution of the neighbors' velocities. The modified velocity calculated from Eq. (10) replaces the original velocity in the density and position update equations, but not in the momentum equation [7].

Simulation algorithm

The time evolution of the dynamic system is determined using a second order explicit Runge-Kutta method [10]. At the beginning of each time step, a neighbors list is assembled to indicate the set of markers that fall within the kernel support of a marker; if N markers are used in the simulation, N lists are generated. The force components appearing on the right hand side of Eqs. (4) and (5) are subsequently computed based on these neighbors lists. Two different functions are called to capture the interaction between markers according to their types, i.e., fluid or solid[1] (Fluid-solid and solid-solid interactions are not the subject of the current document. We refer the interested readers to our peer-reviewed publications), via SPH or lubrication force models. In the second stage, the state of the fluid markers, including position, velocity, and density, is updated

[1] Fluid-solid is not the subject of the current document. We refer the interested readers to our peer-reviewed publications

based on Eqs. (4), (5), and (8). Rigid wall boundaries are implemented by attaching 3 layers of markers with zero or pre-defined velocities to the wall. The overall simulation algorithm is summarized in Figure 2.

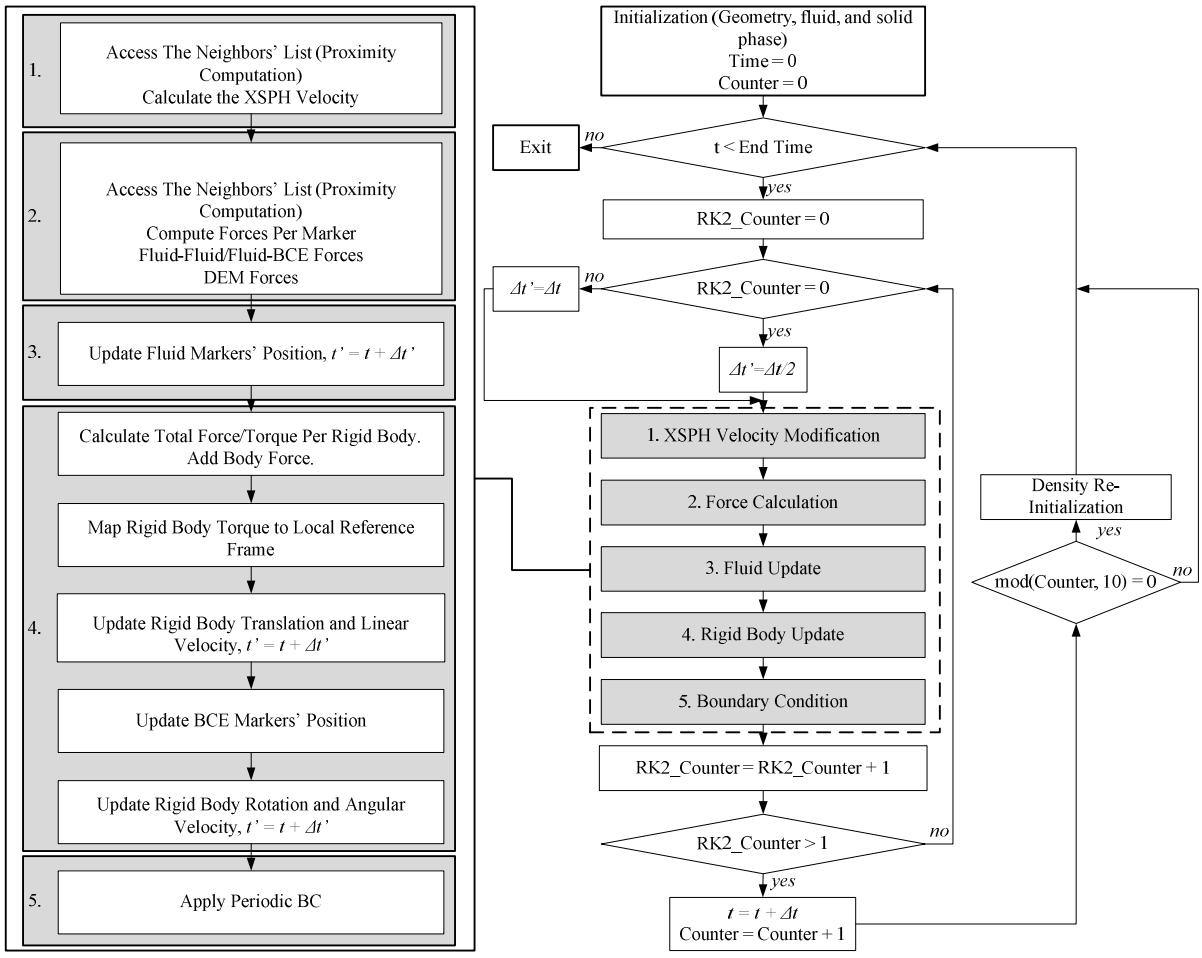

Figure 2. Flow diagram of the simulation algorithm.

The algorithm above was implemented to execute in parallel on Graphics Processing Unit (GPU) cards using the CUDA library [11] and a C-with-extensions language. The hardware used to run the simulations that produced the results reported in this contribution, NVIDIA Kepler K20, has close to 3000 parallel scalar processors. At each time step, four different functions are launched on the GPU to calculate the inter-marker forces, carry out fluid markers time integration, carry out rigid body time integration, and enforce boundary conditions. The lists of neighbors needed to evaluate the inter-marker forces are generated via a proximity computation algorithm which is based on a decomposition of the computation domain into smaller cubes

called bins (Figure 3). The side length of each bin is roughly equal to the radius of the support domain of an SPH marker. Two arrays are used as a hash table to sort the markers according to their location in the domain. Based on the sorted hash table, each marker accesses the list of markers intersecting the self and neighboring bins to calculate the forcing terms. The proximity computation algorithm takes advantage of the fast GPU sorting and scan algorithms provided by the Thrust library [12]. After finding the forcing terms, the fluid markers and rigid bodies are updated based on their specific GPU-based parallel algorithms.

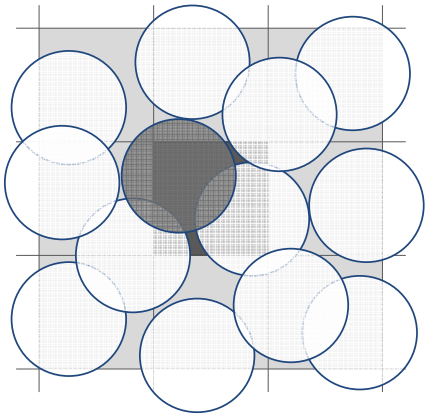

Figure 3. 2D representation of spatial subdivision used for proximity computation. The support domain of the main and neighbor markers, which are accessed through the neighbor bins, are shown with gray and white, respectively.

Several Thrust library [12] functions including radix sort, scan, and reduction are invoked at each time step to implement the time integration algorithm. The algorithm in Figure 2 was implemented as a sequence of GPU functions calls for improved use of fast memory (cache, shared, and registers) and code vectorization through coalesced memory accesses. A similar coding style was maintained for the density re-initialization, boundary condition implementation, and mapping of the markers' data on an Eulerian grid for post processing. Overall, the simulation tool scales linearly and is therefore capable of modeling large systems. Figure 4 and demonstrates the linear scalability in terms of the domain size (number of simulation markers).

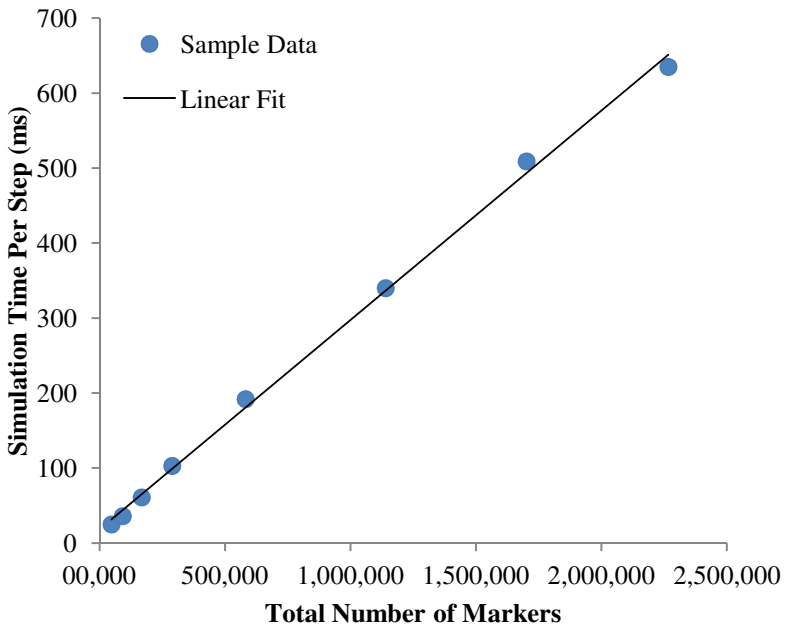

Figure 4. Scaling analysis: simulation time vs. total number of markers. The simulation time scales linearly as the size of the computation domain increases.

References

1. Monaghan, J., *Smoothed particle hydrodynamics.* Reports on Progress in Physics, 2005. **68**(1): p. 1703-1759.
2. Monaghan, J.J., *An introduction to SPH.* Computer Physics Communications, 1988. **48**(1): p. 89-96.
3. Liu, M. and G. Liu, *Smoothed Particle Hydrodynamics (SPH): An overview and recent developments.* Archives of Computational Methods in Engineering, 2010. **17**(1): p. 25-76.
4. Morris, J., P. Fox, and Y. Zhu, *Modeling low Reynolds number incompressible flows using SPH.* Journal of Computational Physics, 1997. **136**(1): p. 214-226.
5. Benz, W., *Smoothed particle hydrodynamics: A review in the numerical modelling of nonlinear stellar pulsations: Problems and prospects, JR Butchler ed.* 1990, Kluwer Acad. Publ.
6. Monaghan, J.J., *Smoothed particle hydrodynamics (to simulate nonaxisymmetric phenomena in astrophysics).* Annual Review of Astronomy and Astrophysics., 1992. **30**.
7. Colagrossi, A. and M. Landrini, *Numerical simulation of interfacial flows by smoothed particle hydrodynamics.* Journal of Computational Physics, 2003. **191**(2): p. 448-475.
8. Dilts, G.A., *Moving-least-squares-particle hydrodynamics-I. Consistency and stability.* International Journal for Numerical Methods in Engineering, 1999. **44**(8): p. 1115-1155.

9. Monaghan, J., *On the problem of penetration in particle methods.* Journal of Computational Physics, 1989. **82**(1): p. 1-15.
10. Atkinson, K.E., *An introduction to numerical analysis*. 1989, John Wiley & Sons, USA.
11. NVIDIA Corporation. *NVIDIA CUDA Developer Zone*. 2012 [cited 2012 October 30, 2012]; Available from: https://developer.nvidia.com/cuda-downloads.
12. Hoberock, J. and N. Bell. *Thrust: C++ Template Library for CUDA*. December 23, 2012]; Available from: http://thrust.github.com/.