A high performance computing approach to the simulation of fully resolved coupled fluid-multibody dynamics systems with...
A high performance computing approach to the simulation of fluid-solid interaction problems with rigid and flexible components

Arman Pazouki∗, Radu Serban†, Dan Negrut‡
1 University of Wisconsin-Madison, 1513 University Avenue, Madison, WI-53706, USA

Abstract
This work outlines a unified multi-threaded, multi-scale High Performance Computing (HPC) approach for the direct numerical simulation of Fluid-Solid Interaction (FSI) problems. The simulation algorithm relies on the extended Smoothed Particle Hydrodynamics (XSPH) method, which approaches the fluid flow in a Lagrangian framework consistent with the Lagrangian tracking of the solid phase. A general 3D rigid body dynamics and an Absolute Nodal Coordinate Formulation (ANCF) are implemented to model rigid and flexible multi-body dynamics. The two-way coupling of the fluid and solid phases is supported through use of Boundary Condition Enforcing (BCE) markers that capture the fluid-solid coupling forces by enforcing a no-slip boundary condition. The solid-solid short range interaction, which has a crucial impact on the small-scale behavior of fluid-solid mixtures, is resolved via a lubrication force model. The collective system states are integrated in time using an explicit, multi-rate scheme. To alleviate the heavy computational load, the overall algorithm leverages parallel computing on Graphics Processing Unit (GPU) cards. Performance and scaling analysis are provided for simulations scenarios involving one or multiple phases with up to tens of thousands of solid objects. The software implementation of the approach, called Chrono::Fluid, is part of the Chrono project and available as an open-source software.

Keywords
fluid-solid interaction · high performance computing · smoothed particle hydrodynamics · rigid body dynamics · flexible body dynamics

1. Introduction
The last decade witnessed a paradigm shift in the microprocessor industry towards chip designs that provide strong support for parallel computing. Teraflop computing, i.e. computing at the rate of 10^{12} Floating-point Operation Per Second, until recently the privilege of a select group of large research centers, is becoming a commodity due to inexpensive GPU cards and many-core x86 processors. The steady improvements in processor speed and architecture, memory layout and capacity, and power efficiency have motivated a trend of re-evaluating simulation algorithms with an eye towards identifying solutions that map well onto these new hardware platforms. In this context, a scrutinizing of the existing Fluid-Solid Interaction (FSI) solutions reveals that they are mostly inadequate since they either rely on algorithms that do not map well on emerging computer architectures, or they are unable to capture phenomena of interest which may require the simulation of tens of thousands of

∗E-mail: pazouki@wisc.edu
†E-mail: serban@wisc.edu
‡E-mail: negrut@wisc.edu
rigid and deformable bodies that interact directly or through the fluid media.

FSI problems are usually simulated in either an Eulerian-Lagrangian framework, where the Lagrangian solid phase moves with/over the Eulerian grid used for fluid discretization, or an Eulerian-Eulerian framework, where the solid phase is considered as an average of ensembles instead of a specific state. While the former approach delivers the flexibility required by general multibody dynamics problems, it does not provide a level of performance suitable for the simulation of dense fluid-solid problems such as suspensions. This can be traced back to the costly processes that overlaps the two different representations of the fluid and solid phases.

The Lagrangian representation of the fluid flow dovetails well with the Lagrangian approach used for the motion of the solid phase. This approach has recently been employed in the context of multibody dynamics and particle suspension in [12, 25, 29–31, 33]. Specifically, the FSI methodology relying on Smoothed Particle Hydrodynamics (SPH) [16, 19, 22] and rigid body Newton-Euler equation of motion [10] was discussed in detail in [29, 30]. This or a similar framework was used for the investigation and validation of rigid body suspensions [30], as well as flexible beams interacting with fluid [12, 31, 33]. Additionally, support for the short-range solid-solid interaction was introduced in [30, 31], which was leveraged in the simulation of dense suspensions.

This contribution concentrates on the high performance computing approach required for the efficient simulation of FSI problems. Although the approach and algorithms explained herein can leverage any multi-threaded architecture, the CUDA library [26] was employed for the execution of all solution components on the GPU, with negligible data transfer between the host (CPU) and the device (GPU). The paper is organized as follows. The numerical methods adopted for the simulation of fluid and solid phases, fluid-solid coupling, and solid-solid short range interaction are explained in Sect. 2. The computational scheme, including the time integration algorithm, HPC-based implementation, and parallel neighbor search required for SPH are described in Sect. 3. Section 4 contains a performance analysis of the algorithm using Kepler-type GPU cards.

2. Numerical approach

The proposed FSI simulation framework combines SPH for the simulation of the fluid flow, the Newton-Euler formalism for rigid body dynamics, and ANCF for modeling flexible body dynamics. These algorithmic components are described in more detail in the following subsections, including a discussion on the approach adopted for resolving fluid-solid and solid-solid interactions.

2.1. Smoothed Particle Hydrodynamics

The term smoothed in SPH refers to the approximation of point properties via a smoothing kernel function \( W \), defined over a support domain \( S \). This approximation reproduces functions with up to 2nd order of accuracy, provided the kernel function: (1) approaches the Dirac delta function as the size of the support domain tends to zero, that is \( \lim_{r \to 0} W(r, h) = \delta(r) \), where \( r \) is the spatial distance and \( h \) is a characteristic length that defines the kernel smoothness; (2) is symmetric, i.e., \( W(r, h) = W(-r, h) \); and (3) is normal, i.e., \( \int_{S} W(r, h) dV = 1 \), where \( dV \) denote the differential volume. A typical spatial function \( f(x) \) is then approximated by \( \langle f(x) \rangle \) as

\[
\langle f(x) \rangle = \int_{S} f(x) W(x' - x, h) dV + O(h^2)
\]

(1)

To simplify notation, in the remainder of this document we use \( f(x) \) to represent \( \langle f(x) \rangle \). Using Eq. (1) and the divergence theorem, the spatial derivatives of a function can be mapped to the derivatives of the kernel function. For instance, the gradient of a function can be written as

\[
\nabla f(x) = \int_{\partial S} \frac{f(x') W(x - x', h) n dA}{V} - \int_{S} \frac{f(x') \nabla W(x - x', h) dV}{V}
\]

(2)

where \( \partial S \) is the boundary of the integration volume \( V \) and \( dA \) is the differential area. By imposing an additional property for the kernel function, namely that (4) it approaches zero as \( r \) increases, i.e., \( \lim_{r \to \infty} W(r, h) = 0 \), the first term on the right hand side of Eq. (2) vanishes. Note that some additional considerations, which will be addressed later, are required for the SPH approximation near boundaries. The term particle in SPH terminology indicates the discretization of the domain by a set of Lagrangian particles. To remove the ambiguity caused by the use of the term rigid particles in the context of FSI problems, we use the term marker to refer to the SPH discretization unit. Each marker has mass \( m \) associated to the representative volume \( dV \) and carries all of the
essential field properties. As a result, any field property at a certain location is shared and represented by the markers in the vicinity of that location. The value of a certain unknown at a given location is calculated according to the distance between that location and the collection of markers in its vicinity using the field values at these markers and the expression of the kernel function \( W \). This leads for the second approximation embedded in SPH, which can be expressed as

\[
f(x) = \int \frac{f(x')}{\rho(x')} W(x - x', h) \rho(x') \, dV
\]

\[
\approx \sum_b \frac{m_b}{\rho_b} f(x_b) W(x - x_b, h) ,
\]

where \( b \) is the marker index and \( \rho_b \) is the fluid density, smoothed at the marker location \( x_b \). The summation in Eq. (3) is over all markers whose support domain overlaps the location \( x \). Several other properties of the kernel functions are provided in [16]. For instance, the kernel function should be a positive and monotonically decreasing function of \( r \), which implies that the influence of distant markers on field properties at a given location is less than that of nearby markers. Moreover, for acceptable computational performance and to avoid a quadratic computational complexity, kernel functions have a compact domain of influence with a radius \( R \), defined as some finite multiple \( \kappa \) of the characteristic length \( h \), as shown in Figure 1. The methodology adopted herein relies on a cubic spline.

\[
W(q, h) = \frac{1}{4 \pi h^3} \times \begin{cases} 
(2 - q)^3 - 4(1 - q)^3, & 0 < q < 1 \\
(2 - q)^3, & 1 < q < 2 \\
0, & q \geq 2
\end{cases}
\]

where \( q \equiv |r|/h \), has a support domain with radius 2\( h \).

2.1.1. SPH for fluid dynamics

For fluid dynamics, the continuity and momentum equations are given in Lagrangian form as

\[
\frac{d \rho}{dt} = -\rho \nabla \cdot \mathbf{v}
\]

and

\[
\frac{dv}{dt} = -\frac{1}{\rho} \nabla p + \frac{\mu}{\rho} \nabla^2 \mathbf{v} + \mathbf{f} ,
\]

where \( \mu \) is the fluid viscosity, while \( \mathbf{v} \) and \( p \) are the flow velocity and pressure, respectively. Within the SPH framework described earlier, Eqs. (5) and (6) are discretized at an arbitrary location \( x = x_o \) within the fluid domain as [24]

\[
\dot{\rho}_a = \frac{d \rho_a}{dt} = \rho_a \sum_b \frac{m_b}{\rho_b} (v_{ab} - v_b) \cdot \nabla_a W_{ab} ,
\]

and

\[
\dot{v}_a = \frac{dv_a}{dt} = -\sum_b m_b \left[ \left( \frac{\rho_a}{\rho_b} + \frac{\rho_b}{\rho_a} \right) \nabla_a W_{ab} + \Pi_{ab} \right] + f_a .
\]

In the above equations, quantities with subscripts \( a \) and \( b \) are associated with markers \( a \) and \( b \) (see Figure 1), respectively. It is important to note that these quantities are different from the corresponding physical quantities at locations \( x_a \) and \( x_b \). The viscosity term \( \Pi_{ab} \) is defined as

\[
\Pi_{ab} = -\frac{(\mu_a + \mu_b)}{\rho_a \rho_b} \nabla_a W_{ab} ,
\]

where \( x_{ab} = x_a - x_b, W_{ab} = W(x_{ab}, h) \), \( \nabla_a \) is the gradient with respect to \( x_a \), i.e., \( \partial /\partial x_a \), and \( \epsilon \) is a regularization coefficient. Quantities with an over-bar are averages of the corresponding quantities for markers \( a \) and \( b \).

Alternative viscosity discretizations include:

1. the model suggested in [5]:

\[
\Pi_{ab} = -\mu_a \rho_a \rho_b \nabla_a W_{ab} + \frac{2}{\rho_a \rho_b} (\mu_a + \mu_b) (x_{ab}^2 + \epsilon h_{ab}^2) \nabla_a W_{ab} ;
\]

2. direct discretization of \( \nabla^2 \) operator [22, 24]; and

3. the class of artificial viscosity models introduced in [17, 19, 23].

However, using Eq. 9 is preferred since it has the following properties: (1) it ensures that the viscous force is along the shear direction \( \mathbf{v}_{ab} \), instead of the particles center line \( x_{ab} \); (2) it is less sensitive to local velocities; (3) it allows for better computational efficiency by removing the nested loop required for the
The fluid flow equations (7) and (8) are solved together with allows 1% flow compressibility [22].

\[ p = \frac{c_s^2 \rho_0}{\gamma} \left( \left( \frac{\rho}{\rho_0} \right)^\gamma - 1 \right), \]  
(10) where \( \rho_0 \) is the reference density of the fluid, \( \gamma \) tunes the stiffness of the pressure-density relationship, and \( c_s \) is the speed of sound. The value \( c_s \) is adjusted depending on the maximum speed of the flow, \( V_{max} \), to keep the flow compressibility below some arbitrary value. We use \( \gamma = 7 \) and \( c_s = 10V_{max} \), which allows 1% flow compressibility [22].

The weakly compressible SPH model, the pressure \( p \) is evaluated using an equation of state [22]

\[ \rho = \rho_0 \left( \frac{p}{p_{ref}} \right)^{\gamma x}, \]  
(11)

\[ \Delta v_a = \zeta \sum_b m_b (v_b - v_a) W_{ab}, \]  
(14) where \( 0 \leq \zeta \leq 1 \) adjusts the contribution of the neighbors’ velocities. All the results reported herein were obtained with \( \zeta = 0.5 \). The modified velocity calculated from Eq. (13) replaces the original velocity in the density and position update equations, but not in the momentum equation [6].

### 2.2. Rigid body dynamics

Once the fluid-solid interactions between individual markers; i.e., the right hand side of Eqs. (7) and (8), are accounted for, the total rigid body force and torque due to the interaction with the fluid can be obtained by respectively summing the individual forces and their induced torques over the entire rigid body. They are then added to any other forces, including external and contact forces. The dynamics of the rigid bodies is fully characterized by the Newton-Euler equations of motion (EOM), see for instance [10]:

\[ \frac{dV_i}{dt} = F_i, \]  
(15)\[ \frac{dX_i}{dt} = V_i, \]  
(16)\[ \frac{d\omega_i}{dt} = J_i^{-1} \left( T_i - \omega_i \times I_i \omega_i \right), \]  
(17)\[ \frac{dq_i}{dt} = -\frac{1}{2} C_i \omega_i, \]  
(18) where \( F_i, T_i, X_i, V_i, \omega_i \in \mathbb{R}^3 \) denote the force, torque, position, velocity, and angular velocity associated to body \( i \), \( i = 1, 2, \ldots, n_v \) respectively. The quantity \( q_i \) denotes the rotation quaternion, while \( M_i \) and \( J_i \) are the mass and moment of inertia, respectively. Quantities with a prime symbol are represented in the rigid body local reference frame. Given \( a = [a_x, a_y, a_z] \in \mathbb{R}^3 \) and \( q = [q_x, q_y, q_z, q_w] \in \mathbb{R}^4 \), the auxiliary matrices \( \tilde{a} \) and \( G \) are defined as [10]

\[ \tilde{a} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \quad \text{and} \quad G = \begin{bmatrix} -q_y & q_z & q_w & -q_x \\ -q_z & -q_w & q_x & q_y \\ -q_w & q_x & -q_y & q_z \end{bmatrix}. \]  
(20)
2.3. Flexible body dynamics

The ANCF formulation [36], which allows for large deformations and large rigid body rotations, is adopted herein for the simulation of flexible bodies suspended in the fluid. While extension to other elastic elements is straightforward, the current Chrono::Fluid implementation only supports gradient deficient ANCF beam elements, which are used to model slender flexible bodies composed of \( n_e \) adjacent ANCF beam elements. The flexible bodies are modeled using a number \( n_n = n_e + 1 \) of equally-spaced node beam elements, each represented by 6 coordinates, \( \mathbf{e}_j = [\mathbf{r}_j^T, \mathbf{r}_j', \mathbf{r}_j'']^T, \ j = 0, 1, \ldots, n_n; \) i.e., the three components of the global position vector of the node, and the three components of its slope. This is therefore equivalent to a model using \( n_e \) ANCF beam elements with \( 6 \times n_n \) continuity constraints, but is more efficient in that it uses a minimal set of coordinates. We note that formulations using gradient deficient ANCF beam elements display no shear locking problems [9, 34, 35] and, due to the reduced number of nodal coordinates, are more efficient than fully parametrized ANCF elements. However, gradient deficient ANCF beam elements cannot describe a rotation about its axis and therefore cannot model torsional effects.

Consider first a single ANCF beam element of length \( \ell \). The global position vector of an arbitrary point on the beam center line, specified through its element spatial coordinate \( 0 \leq x \leq \ell \), is then obtained as

\[
r(x, \mathbf{e}) = S(x)e,
\]

where \( \mathbf{e} = [\mathbf{e}_x^T, \mathbf{e}_y^T]^T \in \mathbb{R}^{12} \) is the vector of element nodal coordinates. With \( I \) being the \( 3 \times 3 \) identity matrix, the shape function matrix \( S = [S_1 \ S_2 \ S_3 \ S_4 \ S_5 \ S_6] \in \mathbb{R}^{3 \times 12} \) is defined using the shape functions [36]

\[
S_1 = 1 - 3\xi^2 + 2\xi^3,
S_2 = \xi (\xi^2 - 2\xi + 2),
S_3 = 3\xi^2 - 2\xi^3,
S_4 = -\xi^2 + \xi^3,
\]

where \( \xi = x/\ell \in [0, 1] \). The element EOM are then written as

\[
\mathbf{Me} + \mathbf{Q}^e = \mathbf{Q}^n,
\]

where \( \mathbf{Q}^e \) and \( \mathbf{Q}^n \) are the generalized element elastic and applied forces, respectively, and \( \mathbf{M} \in \mathbb{R}^{12 \times 12} \) is the symmetric consistent element mass matrix defined as

\[
\mathbf{M} = \int_0^\ell \rho\mathbf{A}\mathbf{S}^T \mathbf{S} dx.
\]

The generalized element elastic forces are obtained from the strain energy expression [36] as

\[
\mathbf{Q}^e = \int_0^\ell E\mathbf{A}c_{11} \left( \frac{\partial \mathbf{e}_{11}}{\partial \mathbf{e}} \right)^T dx + \int_0^\ell E\mathbf{A}c_{13} \left( \frac{\partial \kappa}{\partial \mathbf{e}} \right)^T dx,
\]

where \( c_{11} = (r'_x r_x - 1)/2 \) is the axial strain and \( \kappa = r_x \times r_x/r_x^3 \) is the magnitude of the curvature vector. The required derivatives of the position vector \( r \) can be easily obtained from Eq. (21) in terms of the derivatives of the shape functions as \( r_x(x, e) = S_3(x)e \) and \( r_{xx}(x, e) = S_3(x)e \).

External applied forces, in particular the forces due to the interaction with the fluid, see Sect. 2.4, are included as concentrated forces at a BCE marker. The corresponding generalized forces are obtained from the expression of the virtual work as

\[
\mathbf{Q}^n = S^T(\rho_0)\mathbf{F},
\]

where \( \mathbf{F} \) is the external point force and the shape function matrix is evaluated at the projection onto the element’s center line of the force application point. The generalized gravitational force can be computed as

\[
\mathbf{Q}^g = \int_0^\ell \rho_0 \mathbf{A}\mathbf{S}^T \mathbf{g} dx.
\]

In the above expressions, \( \rho_0 \) represents the element mass density, \( A \) is the cross section area, \( E \) is the modulus of elasticity, and \( I \) is the second moment of area.

The EOM for a slender flexible body composed of \( n_e \) ANCF beam elements are obtained by assembling the elemental EOMs of Eq. (23) and taking into consideration that adjacent beam elements share \( 6 \) nodal coordinates. Let \( \hat{\mathbf{e}} = [\mathbf{e}_e^T, \mathbf{e}_e^T, \ldots, \mathbf{e}_e^T]^T \) be the set of independent nodal coordinates; then the nodal coordinates for the \( j \)-th element can be written using the mapping

\[
\begin{bmatrix}
\mathbf{e}_j \\
\mathbf{e}_j
\end{bmatrix}
= \mathbf{B}_j \hat{\mathbf{e}}, \quad \text{with} \quad \mathbf{B}_j =
\begin{bmatrix}
0 & 0 & \cdots & 1 & 0 & \cdots & 0 \\
0 & 0 & \cdots & 0 & 1 & \cdots & 0
\end{bmatrix}
\]

and the assembled EOMs are obtained, from the principle of virtual work, as follows. Denoting by \( \mathbf{M}_j \) be the element mass matrix of Eq. (24) for the \( j \)-th ANCF beam element, it can be written in block form as

\[
\mathbf{M}_j = \begin{bmatrix}
\mathbf{M}_{j,tt} & \mathbf{M}_{j,tr} \\
\mathbf{M}_{j,rt} & \mathbf{M}_{j,rr}
\end{bmatrix},
\]

where \( \mathbf{M}_{j,tt} = \mathbf{M}_{j,rr} \) and all sub-blocks have dimension \( 6 \times 6 \). Here, \( t \) denotes the left end of the beam element, i.e., the node.
characterized by the nodal coordinates $e_{i-1}$, while $r$ corresponds to the node with coordinates $e_i$. With a similar decomposition of a generalized element force into

$$Q_i = \begin{bmatrix} Q_{i,j}^r \\ Q_{i,j}^s \end{bmatrix}$$

one obtains

$$\dot{\mathbf{M}}e = \ddot{\mathbf{Q}}^e - \dot{\mathbf{Q}}^e$$

where

$$\ddot{\mathbf{Q}}^e - \dot{\mathbf{Q}}^e = \begin{bmatrix} \sum Q_{i,j}^e \\ \sum Q_{i,j}^e + \sum Q_{i,j}^s \\ \vdots \\ \sum Q_{n_r,r}^e \\ \sum Q_{n_r,r}^e + \sum Q_{n_r,r}^s \end{bmatrix} - \begin{bmatrix} Q_{i,j}^e \\ Q_{i,j}^e + Q_{i,j}^s \\ \vdots \\ Q_{n_r,r}^e \\ Q_{n_r,r}^e + Q_{n_r,r}^s \end{bmatrix}$$

Inclusion of additional kinematic constraints, e.g., anchoring the beam at one end to obtain a flexible cantilever or fixing its position to obtain a flexible pendulum, can be done either by formulating the EOM as differential-algebraic equations or by deriving an underlying ODE after explicitly eliminating the corresponding constrained nodal coordinates. The latter approach was used in all simulations involving flexible cantilevers discussed in Sect. 4.

2.4. Fluid-solid interaction

The two-way fluid-solid coupling was implemented based on a methodology described in [29]. The state update of any SPH marker relies on the properties of its neighbors and resolves shear as well as normal inter-marker forces. For the SPH markers close to solid surfaces, the SPH summations presented in Eqs. (7), (8), (12), and (14) capture the contribution of fluid markers. The contribution of solid objects is calculated via Boundary Condition Enforcing (BCE) markers placed on and close to the solid surface as shown in Figure 2. The velocity of a BCE marker is obtained from the rigid/deformable body motion of the solid thus ensuring the no-slip condition on the solid surface. Including BCE markers in the SPH summation equations (7) and (8) results in fluid-solid interaction forces that are added to both fluid and solid markers.

![Figure 2](image)

**Figure 2.** Fluid-solid interaction using BCE markers attached to a rigid body. BCE and fluid markers are represented by black and white circles, respectively. The BCE markers positioned in the interior of the body (markers $g$ and $f$ in the figure) should be placed to a depth less than or equal to the size of the compact support associated with the kernel function $W$. A similar concept yet with different kinematics is used for flexible bodies.

2.5. Solid-solid short range interaction

Dry friction models typically used to characterize the dynamics of granular materials [1, 13, 14] do not capture accurately the impact of solid surfaces in hydrodynamics media. In practice, it is infeasible to resolve the short-range, high-intensity impact forces arising in wet media due to computational limits on space and time resolution. Ladd [15] proposed a normal lubrication force between two spheres that increases rapidly as the distance between particles approaches zero and therefore prevents the actual touching of the spheres:

$$F_{ij} = \min \left\{ -6\pi h \left( \frac{a_i a_j}{a_i + a_j} \right)^2 \left( \frac{1}{s} - \frac{1}{s_0} \right) \cdot \nabla v_{nij}, \right\}$$

where, $a_i$ and $a_j$ are the sphere radii, $v_{nij}$ is the normal component of the relative velocity, and $s$ is the distance between surfaces. For $s > s_0$, $F_{ij} = 0$, and the spheres are subject only to hydrodynamic forces.

Equation (34) provides a basic model for the estimation of the lubrication force in normal direction. The generalization of this model to non-spherical objects requires the calculation of the minimum distance and the curvature of the two contact surfaces. The calculation of the partial lubrication force between nonspherical surfaces follows the approach proposed in [8] for a lattice Boltzmann method but is amended to fit the Lagrangian formulation adopted herein. Accordingly, the force model provided in Eq. (34) is modified as

$$F_{ij}^{lb} = \sum f_{ij}^b,$$

with

$$f_{ij}^b = \min \left\{ -\frac{3}{2} \pi h^2 \left( \frac{1}{s^2} - \frac{1}{s_0^2} \right) \cdot \nabla v_{nij}, \right\}$$

Figure 2 shows the fluid-solid interaction using BCE markers attached to a rigid body. BCE and fluid markers are represented by black and white circles, respectively. The BCE markers positioned in the interior of the body (markers $g$ and $f$ in the figure) should be placed to a depth less than or equal to the size of the compact support associated with the kernel function $W$. A similar concept yet with different kinematics is used for flexible bodies.
where \( s^* \) and \( \nu^* \) denote the markers’ relative distance and velocity, respectively, and the summation is over all interacting markers of the two solid objects.

3. HPC implementation

Chrono::Fluid [4], an open-source simulation framework for fluid-solid interaction, provides an implementation that executes all phases of the solution method on the GPU. A brief overview of the GPU hardware and programming model adopted is followed below by a detailed discussion of the critical kernels that implement the proposed modeling and solution approach.

3.1. GPU hardware and programming model

To a very large extent, the performance of today’s simulation engines is dictated by the memory bandwidth of the hardware solution adopted. Recent numerical experiments conducted by the authors revealed that only about 5 to 10% of the peak flop rate is reached by the large number of cores present on today’s architectures since these cores most of the time idle waiting for data from global memory or RAM. It is this observation that motivated the selection of the GPU as the target hardware for implementing Chrono::Fluid. At roughly 300 GB/s, the GPU memory bandwidth stands four to five times higher than what one could expect on a fast CPU.

To describe the hardware organization of the GPU, we consider here the NVIDIA GeForce GTX 680 [18, 27]. This GPU is based on the first generation of Kepler architecture, code name GK104, which is also implemented in Tesla K10. The Kepler architecture relies on a Graphics Processing Cluster (GPC) as the defining high-level hardware block. There are a total of 4 GPCs on the GK104. Each GPC includes two Stream Multi-processors (SM), each of which has 192 Scalar Processors (SP), for a total of 1536 SPs, and 3.1 TFlops processing rate.

In addition to the processing cores, the second important aspect of GPU hardware is that of the memory hierarchy. The memory on the GPU is divided into several types, each with different access patterns, latencies, and bandwidths:

- **Registers** (read/write per thread): 65536, 32-bit memory units. Very low latency, high bandwidth (\(\approx 10\) TB/s cumulative) memory used to hold thread-local data.

- **Shared memory/L1 cache** (read/write per block): 64 KB. Low latency, high bandwidth (\(\approx 1.5-2\) TB/s cumulative) memory divided between shared memory and L1 cache.

- **Global memory** (read/write per grid): 2 GB. Used to hold input and output data. Accessible by all threads, with a bandwidth of 192 GB/s and a higher latency (\(\approx 400-800\) cycles) than shared memory and registers. All accesses to global memory pass through the L2 cache. The latter is 512 KB large and has a bandwidth of 512 B/clock cycle.

**Constant memory** (read only per grid): 48 KB per Kepler SM. Used to hold constants, serviced at the latency and bandwidth of the L1 cache upon a cache hit, or those of the global memory upon a cache miss.

The parallel execution paradigm best supported on GPUs is “single instruction multiple data” (SIMD). In this model, if for instance two arrays of length one million are to be added, one million threads are launched with each executing the same instruction; i.e., adding two numbers, but on different data—each thread adding two different numbers. Fortunately, SIMD computing is very prevalent in the solution methodology proposed, where each SPH marker is handled by a thread in the same way in which hundreds of thousands of other threads handle their markers using different data. While strongly leveraging the SIMD model owing to the fine grain parallelism that it exposes, the methodology adopted is prone to lead to memory access patterns that do not display high spatial and/or temporal locality. This is because the SPH markers move in time leading to less structured memory accesses that adversely impact the effective bandwidth reached by the code.

3.2. Time integration

Chrono::Fluid uses a second order explicit mid-point Runge-Kutta (RK2) scheme [2] for the time integration of the fluid and solid phases, the latter in its rigid or flexible representation. Algorithm 1 summarizes the steps required for the calculation of the force on the SPH markers, rigid bodies, and deformable beams at time step \( k \). The variables \( N_m, N_r, N_f \) denote the number of markers, rigid bodies, and flexible beams, respectively. The arrays \( p, x, v, \text{ and } \tilde{v} \) store the density, position, velocity, and modified velocity for all markers, respectively; for example, \( p = \{ p_i \}_{a = 0, 1, 2, ..., N_m - 1} \).

The external forces on the rigid and flexible bodies include the FSI forces captured via BCE markers at distributed locations on the solid surfaces. The distributed forces need to be accumulated into a single force and torque at the center of each rigid body, or point forces at node locations of each flexible body. The reduction (summation of the forces and torques) is handled by parallel scan operations available through the Thrust library [11], which exposes a scan algorithm that scales linearly. The RK2 integration scheme requires the calculation of the force at the beginning as well as middle of the time step. Algorithm 2 lists the steps required for the time integration of a typical FSI problem.
To improve the code vectorization and use of fast memory, i.e., L1/L2 cache, shared memory, and registers, each computation task was implemented as a sequence of light GPU kernels. For instance, different computation kernels are implemented to update the attributes of the solid bodies, including force, moment, rotation, translation, linear and angular velocity, and locations of the associated BCE markers. A similar coding philosophy was maintained for the density re-initialization, boundary condition implementation, and mapping of the marker data on an Eulerian grid for post-processing.

3.2.1. Multi-rate integration

Stable integration of the SPH fluid equations requires step sizes which are also appropriate for propagating the dynamics of any rigid solids in the FSI system. However, integration of the dynamics of deformable bodies, especially as their stiffness increases, calls for very small time steps. To alleviate the associated computational cost, we use a dual-rate integration scheme using intermediate steps for the integration of the flexible dynamics EOMs, typically with Δt_{SPH}/Δt_{ANCF} = 10, although stiffer problems may require ratios of up to 50. This aspect is noteworthy given that typical FSI models involve a number of ANCF nodal coordinates required for the flexible bodies. Without a multi-rate implementation, the numerical solution would visit the fluid phase at each integration step, an approach that led to very large solution times. This aspect is further discussed in Sect. 4.

### Algorithm 1: Force Calculation

<table>
<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Calculate modified XSPH velocities</td>
</tr>
<tr>
<td>2</td>
<td>for ( a = 0 ) to ( (N_m - 1) ) do</td>
</tr>
<tr>
<td>3</td>
<td>( \ddot{\mathbf{v}}_a \equiv \dot{\mathbf{v}}_a (\rho^f, \mathbf{x}^f, \mathbf{v}^f) )</td>
</tr>
<tr>
<td>4</td>
<td>end for</td>
</tr>
<tr>
<td>5</td>
<td>SPH forces</td>
</tr>
<tr>
<td>6</td>
<td>for ( a = 0 ) to ( (N_m - 1) ) do</td>
</tr>
<tr>
<td>7</td>
<td>( \rho_a^f \equiv \rho_a (\rho^f, \mathbf{x}^f, \mathbf{v}^f) )</td>
</tr>
<tr>
<td>8</td>
<td>( \mathbf{x}_a^f = \dot{\mathbf{v}}_a )</td>
</tr>
<tr>
<td>9</td>
<td>( \mathbf{v}_a^f = v_a (\rho^f, \mathbf{x}^f, \mathbf{v}^f) )</td>
</tr>
<tr>
<td>10</td>
<td>end for</td>
</tr>
<tr>
<td>11</td>
<td>Rigid body forces</td>
</tr>
<tr>
<td>12</td>
<td>for ( i = 0 ) to ( (N_v - 1) ) do</td>
</tr>
<tr>
<td>13</td>
<td>( \mathbf{V}_i^v \equiv \mathbf{V}_i (\mathbf{v}^f) )</td>
</tr>
<tr>
<td>14</td>
<td>( \mathbf{X}_i^v = \mathbf{V}_i )</td>
</tr>
<tr>
<td>15</td>
<td>( \mathbf{w}_i^v \equiv \omega_i (\mathbf{v}^f, \mathbf{w}^v) )</td>
</tr>
<tr>
<td>16</td>
<td>( \mathbf{q}_i^v \equiv \mathbf{q}_i (\mathbf{w}_i^v, \mathbf{q}_i) )</td>
</tr>
<tr>
<td>17</td>
<td>end for</td>
</tr>
<tr>
<td>18</td>
<td>ANCF forces</td>
</tr>
<tr>
<td>19</td>
<td>for ( j = 0 ) to ( (N_m - 1) ) do</td>
</tr>
<tr>
<td>20</td>
<td>( \mathbf{Q}_j^m = \mathbf{Q}_j^m (\mathbf{v}^f) - \mathbf{Q}_j^m (\mathbf{\hat{v}}^f) + \mathbf{Q}_j^m (\mathbf{\hat{v}}^f) )</td>
</tr>
<tr>
<td>21</td>
<td>( \mathbf{\dot{e}}_j^m = \mathbf{\dot{e}}_j^m )</td>
</tr>
<tr>
<td>22</td>
<td>( \mathbf{\dot{\mathbf{e}}}_j^m = \mathbf{\dot{\mathbf{e}}}_j^m )</td>
</tr>
<tr>
<td>23</td>
<td>end for</td>
</tr>
<tr>
<td>24</td>
<td>RK2 Time Integration</td>
</tr>
<tr>
<td>25</td>
<td>Force calculation at beginning of step (Algorithm 1)</td>
</tr>
<tr>
<td>26</td>
<td>for ( a \in { \alpha } ) do</td>
</tr>
<tr>
<td>27</td>
<td>( \psi_a^{k+1/2} = \psi_a^k + \dot{\psi}_a^k \Delta t/2 ) where ( \psi_a \in { \rho_a, x_a, v_a } )</td>
</tr>
<tr>
<td>28</td>
<td>end for</td>
</tr>
<tr>
<td>29</td>
<td>Half-step updates for fluid, rigid body, and flexible body states</td>
</tr>
<tr>
<td>30</td>
<td>for ( a \in { \beta } ) do</td>
</tr>
<tr>
<td>31</td>
<td>( \dot{\psi}_a^{k+1/2} = \psi_a^k + \dot{\psi}_a^{k+1/2} \Delta t/2 ), where ( \psi_a \in { \mathbf{V}_i, \mathbf{X}_i, \omega_i, \mathbf{q}_i } )</td>
</tr>
<tr>
<td>32</td>
<td>end for</td>
</tr>
<tr>
<td>33</td>
<td>Half-step update for BCE marker positions and velocities</td>
</tr>
<tr>
<td>34</td>
<td>for ( \alpha \in { \alpha } ) do</td>
</tr>
<tr>
<td>35</td>
<td>Calculate ( \mathbf{x}_a^{k+1} ) and ( \mathbf{v}_a^{k+1/2} ) associated to rigid, flexible, or immersed boundary motion.</td>
</tr>
<tr>
<td>36</td>
<td>end for</td>
</tr>
<tr>
<td>37</td>
<td>Half-step update for FSI model</td>
</tr>
<tr>
<td>38</td>
<td>for ( \alpha \in { \alpha } ) do</td>
</tr>
<tr>
<td>39</td>
<td>( \psi_a^{k+1} = \psi_a^k + \dot{\psi}_a^{k+1/2} \Delta t )</td>
</tr>
<tr>
<td>40</td>
<td>end for</td>
</tr>
<tr>
<td>41</td>
<td>Force calculation at half-step (Algorithm 1)</td>
</tr>
<tr>
<td>42</td>
<td>for ( j = 0 ) to ( (N_m - 1) ) do</td>
</tr>
<tr>
<td>43</td>
<td>( \dot{\mathbf{e}}_j^{k+1/2} = \dot{\mathbf{e}}_j^k + \dot{\mathbf{e}}_j^k \Delta t/2 )</td>
</tr>
<tr>
<td>44</td>
<td>( \dot{\mathbf{e}}_j^{k+1/2} = \dot{\mathbf{e}}_j^{k+1/2} + \dot{\mathbf{e}}_j^{k+1/2} \Delta t/2 )</td>
</tr>
<tr>
<td>45</td>
<td>end for</td>
</tr>
<tr>
<td>46</td>
<td>Full-step updates for fluid, rigid body, and flexible body states</td>
</tr>
<tr>
<td>47</td>
<td>for ( a \in { \alpha } ) do</td>
</tr>
<tr>
<td>48</td>
<td>( \psi_a^{k+1} = \psi_a^k + \dot{\psi}_a^{k+1/2} \Delta t )</td>
</tr>
<tr>
<td>49</td>
<td>end for</td>
</tr>
<tr>
<td>50</td>
<td>RK2 Time Integration</td>
</tr>
<tr>
<td>51</td>
<td>Force calculation at beginning of step (Algorithm 1)</td>
</tr>
<tr>
<td>52</td>
<td>for ( a \in { \alpha } ) do</td>
</tr>
<tr>
<td>53</td>
<td>( \psi_a^{k+1} = \psi_a^k + \dot{\psi}_a^{k+1/2} \Delta t/2 ) where ( \psi_a \in { \rho_a, x_a, v_a } )</td>
</tr>
<tr>
<td>54</td>
<td>end for</td>
</tr>
<tr>
<td>55</td>
<td>Half-step updates for fluid, rigid body, and flexible body states</td>
</tr>
<tr>
<td>56</td>
<td>for ( a \in { \beta } ) do</td>
</tr>
<tr>
<td>57</td>
<td>( \dot{\psi}_a^{k+1/2} = \psi_a^k + \dot{\psi}_a^{k+1/2} \Delta t/2 ), where ( \psi_a \in { \mathbf{V}_i, \mathbf{X}_i, \omega_i, \mathbf{q}_i } )</td>
</tr>
<tr>
<td>58</td>
<td>end for</td>
</tr>
<tr>
<td>59</td>
<td>Half-step update for BCE marker positions and velocities</td>
</tr>
<tr>
<td>60</td>
<td>for ( \alpha \in { \alpha } ) do</td>
</tr>
<tr>
<td>61</td>
<td>Calculate ( \mathbf{x}_a^{k+1} ) and ( \mathbf{v}_a^{k+1/2} ) associated to rigid, flexible, or immersed boundary motion.</td>
</tr>
<tr>
<td>62</td>
<td>end for</td>
</tr>
<tr>
<td>63</td>
<td>Force calculation at half-step (Algorithm 1)</td>
</tr>
<tr>
<td>64</td>
<td>for ( j = 0 ) to ( (N_m - 1) ) do</td>
</tr>
<tr>
<td>65</td>
<td>( \dot{\mathbf{e}}_j^{k+1/2} = \dot{\mathbf{e}}_j^k + \dot{\mathbf{e}}_j^k \Delta t/2 )</td>
</tr>
<tr>
<td>66</td>
<td>( \dot{\mathbf{e}}_j^{k+1/2} = \dot{\mathbf{e}}_j^{k+1/2} + \dot{\mathbf{e}}_j^{k+1/2} \Delta t/2 )</td>
</tr>
<tr>
<td>67</td>
<td>end for</td>
</tr>
<tr>
<td>68</td>
<td>Full-step updates for fluid, rigid body, and flexible body states</td>
</tr>
<tr>
<td>69</td>
<td>for ( a \in { \alpha } ) do</td>
</tr>
<tr>
<td>70</td>
<td>( \psi_a^{k+1} = \psi_a^k + \dot{\psi}_a^{k+1/2} \Delta t )</td>
</tr>
<tr>
<td>71</td>
<td>end for</td>
</tr>
<tr>
<td>72</td>
<td>3.3. Proximity calculation and neighbor search</td>
</tr>
</tbody>
</table>

In the current implementation, each marker has a list of neighbor markers; i.e., markers that fall within its support domain. Given this set of lists, the calculation of \( \mathbf{v}_a^k \equiv \mathbf{v}_a (\rho^f, \mathbf{x}^f, \mathbf{v}^f) \), \( \rho_a^k \equiv \rho_a (\rho^f, \mathbf{x}^f, \mathbf{v}^f) \), and \( \mathbf{v}_a^k \equiv \mathbf{v}_a (\rho^f, \mathbf{x}^f, \mathbf{v}^f) \) is straightforward. The loop iterations in Algorithms 1 and 2 have no overlap and can be executed in parallel. The computational bottleneck thus becomes the determination of the neighbor lists through proximity calculation, a step that requires about 70% of the entire computational budget and thus critically impacts the overall performance of the simulation. Herein, we adopt a spatial binning algorithm which is sub-optimal in terms of amount of net work but superior in terms of memory usage. In this approach, the computation domain is divided into a collection of bins. The bins have side lengths equal to the maximum influence distance of a marker, i.e., \( kh \). This localizes the search for the possible interacting markers to the bin and all of its 26 immediate 3D
neighbors. Figure 3 shows the binning approach in 2D.

In the neighbor search approach adopted herein, neighbor lists are not saved in memory; instead, neighbors are evaluated whenever required. Alternatively [28], the principle of action and reaction could be leveraged to calculate and store the acceleration terms for each inter-penetration. Although the second algorithm reduces the amount of work by re-using the acceleration terms for each inter-penetration, it can not be applied to the SPH method due to the massive amount of memory required to store the data associated with all inter-penetration events. Another advantage of the adopted neighbor search approach is the coalesced memory access achieved by sorting and accessing the data based on the markers location. The steps required for accessing the neighbors are summarized in Algorithm 3.

Algorithm 3 Inner loop: accessing neighbor markers

1. Divide the solution space into \( n_b \) bins of \((\Delta x, \Delta y, \Delta z)\) dimensions, where \( n_b = n_x \times n_y \times n_z \), and \((n_x, n_y, n_z)\) is the number of grid cells along \((x, y, z)\) axis.
2. Construct the hash array: \( s = \{ s_a \mid a = 0, 1, 2, ..., N_a - 1 \} \) according to \( s_a = i \times n_y \times n_z + j \times n_z + k \), where \((i, j, k)\) is the location of the bin containing the marker \( a \).
3. Sort \( s \) into \( s_{\text{sorted}} \) and obtain corresponding \( p_{\text{sorted}}, x_{\text{sorted}}, y_{\text{sorted}}, \) and \( v_{\text{sorted}} \).
4. Construct \( c^1 = \{ c^1_p \mid p = 0, 1, 2, ..., n_b - 1 \} \) and \( c^2 = \{ c^2_p \mid p = 0, 1, 2, ..., n_b - 1 \} \), where \( c^1 \) and \( c^2 \) denote the two indices in \( s_{\text{sorted}} \) that bound the sequence of hash values \( s_a = p \).
5. Access markers data in bin \((i, j, k)\) by loading \( \left[ c^1_p, c^2_p \right] \) portions of the sorted arrays \( p_{\text{sorted}}, x_{\text{sorted}}, y_{\text{sorted}}, \) and \( v_{\text{sorted}} \), as needed.

The data sorting in step 3 of Algorithm 3 is performed using the linear-complexity radix sort available in the Thrust library. As a result, this solution component does not affect the overall linear scaling of Algorithms 1 and 2. Working with the sorted arrays for the bulk of the computation has the additional advantage of increasing the memory spatial locality: SPH markers that share a neighborhood in the physical space, do so in their memory location as well.

4. Results

The simulation approach described in Sect. 3 was implemented to execute in parallel on GPU cards using the CUDA programming environment [26]. All simulations reported in this section were run on an NVIDIA GeForce GTX 680 GPU card described in sub-section 3.1. In all simulations, if present, the flexible beams were modeled using \( n_b = 4 \) ANCF beam elements, while the integrals appearing in the elastic forces \( \mathbf{Q}^e \) in Eq. (25) were evaluated using 5 and 3 Gauss quadrature points for the axial and bending elastic forces, respectively.

A snapshot from a typical FSI problem involving flexible bodies is shown in Figure 4. Additional Chrono::Fluid simulation examples can be found at [32]. Comprehensive validation studies of the Chrono::Fluid simulation package are provided in [31]. The focus here is on numerical experiments that illustrate the performance and scalability of the proposed algorithm and its parallel implementation.

We begin by analyzing the efficiency and scaling attributes of the solution for systems composed exclusively of rigid bodies, flexible bodies, or a fluid phase. In each of these three scenar-
Table 1. Time required for advancing the rigid body dynamics simulation by one time step as function of problem size (number of rigid bodies, \(N_r\))

<table>
<thead>
<tr>
<th>(N_m = 0, N_f = 0)</th>
<th>(N_r) ((\times 10^3))</th>
<th>(t) (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.49</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>2.87</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>16.59</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>56.77</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>118.23</td>
<td>78</td>
</tr>
</tbody>
</table>

Table 2. Time required for advancing the flexible body dynamics simulation by one time step as function of problem size (number of flexible bodies, \(N_f\))

<table>
<thead>
<tr>
<th>(N_m = 0, N_r = 0)</th>
<th>(N_f) ((\times 10^3))</th>
<th>(t) (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.78</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>3.51</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>17.55</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>56.94</td>
<td>122</td>
</tr>
<tr>
<td></td>
<td>115.05</td>
<td>238</td>
</tr>
</tbody>
</table>

Table 3. Time required for advancing a fluid dynamics simulation by one time step as function of problem size (number of SPH markers, \(N_m\))

<table>
<thead>
<tr>
<th>(N_m = 0, N_f = 0)</th>
<th>(N_m) ((\times 10^6))</th>
<th>(t) (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.06</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>0.32</td>
<td>121</td>
</tr>
<tr>
<td></td>
<td>0.93</td>
<td>121</td>
</tr>
<tr>
<td></td>
<td>1.79</td>
<td>538</td>
</tr>
<tr>
<td></td>
<td>4.13</td>
<td>1150</td>
</tr>
</tbody>
</table>

The two sets of results provided in Table 5 for different values of \(\tau = \Delta t_{SPH}/\Delta t_{ANCF}\) show a small increase in the simulation time when choosing a very small integration step size for flexible body dynamics. This illustrates the efficiency of the multi-rate integration scheme in improving the time integration stability at a small increase in computational cost. The small changes in the simulation times provided in Tables 4 and 5 are mainly due to the deviations in the magnitude of \(N_m\) as the number of solid objects changes. Linear scaling in Chrono::Fluid is also demonstrated in an experiment where a combined FSI problem, i.e. involving rigid and flexible bodies and including a lubrication force model and two-way coupling with fluid, is solved on domains of increasing size (see Figure 6). As the simulation domain volume is increased by factors from 2 up to 32, the...
5. Conclusions

This contribution discusses details of an HPC approach to the simulation of FSI problems. Relying on: (i) a Lagrangian/Lagrangian formalism for the simulation of the fluid and solid phases, and (ii) GPU parallelism, Chrono::Fluid, the software implementation of the proposed solution methodology, is suitable for the study of multi-phase/multi-component problems such as particle suspensions, polymer flow, and biomedical applications.

Chrono::Fluid has been shown to scale linearly in the dynamics of each phase independently as well as in the overall FSI solution implementation. The dynamics of the fluid phase; i.e. the time propagation of the SPH markers, controls the overall simulation time, and neither cross-phase communications nor solid phase simulation have a notable effect on the simulation time. This represents one advantage of the adopted unified FSI methodology over co-simulation and asynchronous approaches. Chrono::Fluid, whose preliminary validation is discussed in [30], is open source and freely available under a BSD3 license.

Several directions of future work are as identified as follows: (a) revisit the SPH marker numbering scheme in order to improve the spatial and temporal memory access patterns, thus increasing the effective bandwidth of Chrono::Fluid; (b) augment the current implementation for use on cluster supercomputers that adopt a non-uniform memory access model and rely on the Message Passing Interface standard; (c) investigate problems at macroscale such as vehicle fording operations, which call for large scale simulations at low Reynolds numbers and possibly in the presence of very large viscosity; and (d) gauge the potential of Chrono::Fluid in biomechanics applications such as blood flow in deformable arteries or heart, channel occlusion in stroke, etc.

Acknowledgments

Financial support for this work was provided by National Science Foundation award CMMI-0840442. Support for the second and third authors was provided in part by Army Research Office awards W911NF-11-1-0327 and W911NF-12-1-0395.
References


Lagrangian framework for the simulation of rigid and deformable bodies in fluid pp. 33–52 (2014)


