Commun. Comput. Phys.
doi: 10.4208/cicp.110211.121211a

Vol. **12**, No. 4, pp. 1257-1274
October 2012

Computational Aspects of Multiscale Simulation with the Lumped Particle Framework

Omar al-Khayat[1,*] and Hans Petter Langtangen[2,3]

[1] *Computational Geosciences, CBC, Simula Research Laboratory, P.O. Box 134, NO-1325 Lysaker, Norway.*
[2] *Department of Informatics, University of Oslo, P.O. Box 1080, Blindern, NO-0316 Oslo, Norway.*
[3] *Center for Biomedical Computing, Simula Research Laboratory, P.O. Box 134, NO-1325 Lysaker, Norway.*

Received 11 February 2011; Accepted (in revised version) 12 December 2011

Available online 17 April 2012

Abstract. First introduced in [2], the lumped particle framework is a flexible and numerically efficient framework for the modelling of particle transport in fluid flow. In this paper, the framework is expanded to simulate multicomponent particle-laden fluid flow. This is accomplished by introducing simulation protocols to model particles over a wide range of length and time scales. Consequently, we present a time ordering scheme and an approximate approach for accelerating the computation of evolution of different particle constituents with large differences in physical scales. We apply the extended framework on the temporal evolution of three particle constituents in sand-laden flow, and horizontal release of spherical particles. Furthermore, we evaluate the numerical error of the lumped particle model. In this context, we discuss the Velocity-Verlet numerical scheme, and show how to apply this to solving Newton's equations within the framework. We show that the increased accuracy of the Velocity-Verlet scheme is not lost when applied to the lumped particle framework.

AMS subject classifications: 65L12, 65L70, 68W25, 86-08, 34E13, 34E99

Key words: Multiscale modelling, lumped particle model, Velocity-Verlet scheme.

1 Introduction

A wide range of scientific and engineering challenges involve physics on multiple time and length scales. This is particularly the case with the modelling of particle-laden fluids.

*Corresponding author. *Email addresses:* `omark@simula.no` (O. al-Khayat), `hpl@simula.no` (H. P. Langtangen)

These flows often involve physical processes on widely different length and time scales. One specific challenge lies in the fact that particle size rarely is uniform, often varying many orders of magnitude. Consequently, multiscale methodologies must be developed to be able to resolve the physics of these flows. From a computational point of view, new numerical strategies are required to deal with this class of problems [1, 5]. This paper attempts to answer some of the questions that arise in multicomponent modelling of sand-laden fluid flow. The analysis of these issues will be done within the lumped particle model. Specifically, we will focus on the computational challenges of multiscale modelling within this framework.

One of the challenges in modelling sand-laden flows is that the physical processes involved interact over many different length and time scales. For instance, the dynamics of the fluid and the suspended particles occur on time scales which may differ in many orders of magnitude. Moreover, the variable size of the suspensions adds another layer of complexity. The particle sizes range from smallest clay of micrometer in diameter to the much larger millimeter-sized gravel. Recall Fig. 1 that shows the relative size differences in these particles. The effect of a constant external force on the different particle types will typically lead to different resultant outcomes. Moreover, the fluid phase evolves in its own temporal scale which often differs from that of the particles. We will in this section describe one way of simulating such multiscale physics with the lumped particle model.

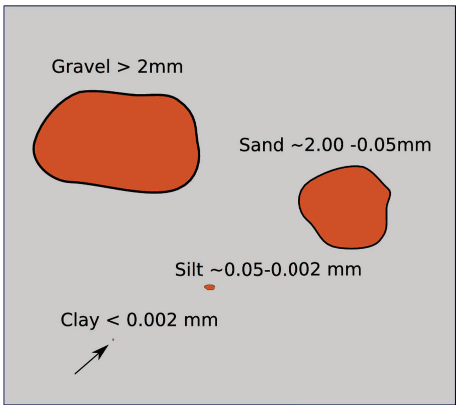


Figure 1: Typical sand particles suspended in fluids.

The lumped particle model is a flexible and numerically efficient framework for the modelling of particle transport in fluid flow, which takes into account fundamental features of particle flow, including advection, diffusion and dispersion of the particles. This framework reproduces particle flow properties inherent in both continuum and discrete approaches, and correctly reproduces advection and diffusion phenomena as special cases [2].

This paper will study computational aspects of multiscale modelling with the lumped particle framework. Specifically, we will study how an implementation of a sand-laden

Figure 2: Tank experiment of a sand-laden multicomponent flow. The image is from [4].

flow with many differently sized particle constituents can be accomplished. Here, we develop a set of protocols to handle computations ranging over physical length scales spanning many orders of magnitude. These simulation protocols are in essence a set of rules of thumb designed to make the framework more efficient. Moreover, since numerical accuracy is a key issue, we will investigate alternative methods for solving the particle equations of motion.

We will first repeat the most important aspects of the framework in Section 2. Moreover, in Section 3.1, we will study the Velocity-Verlet scheme for solving Newton's equations, and show how to apply the method within our framework. Thereafter, we will discuss our multiscale approach in Section 4. Finally, in Section 5, we present a series of numerical experiments for investigating the numerical accuracy and efficiency of the framework.

2 Overview of the lumped particle model

We will now give a short account of the lumped particle modelling framework. The framework is based on a mesoscopic hybrid continuum-particle approach, where groups of particles constitute a *particle lump*. Instead of tracking the individual dynamics of each particle, a weighted spatial averaging procedure is used to evolve the particles in the computational domain. The external forces are applied to the lump of particles, from which an average position and velocity is derived. Hence, the particles are in a sense considered as a continuum, but where the particle nature heavily influences the dynamics. In the following description, we will restrict our attention to a two-dimensional regular lattice.

When computing the evolution of the particles, the particle lumps are partitioned into smaller entities, known as quasi-particles, which are then transported according to local physical effects. These smaller entities recombine into new particle lumps at the target destinations. We partition the computational domain into a regular lattice, with physical

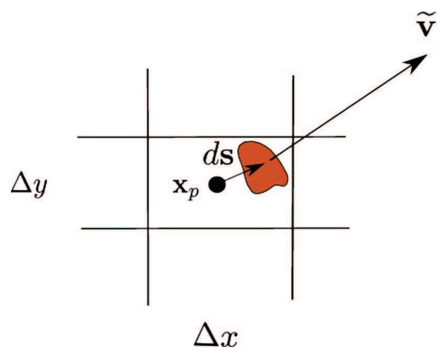


Figure 3: The lumped representation of particles. Particles within a grid cell are treated as a single entity. Here, $\tilde{\mathbf{v}}$ is the average velocity and d$\mathbf{s}$ is the offset from the particle centroid to the cell center.

spacing parameters Δx and Δy. Time is discretized into increments of Δt. As shown in Fig. 3, the lattice defines a set of grid cells, each having a center point $\mathbf{x}_p$. The distribution $N(\mathbf{x}_p,t)$, and the velocity $\mathbf{V}(\mathbf{x}_p,t)$ are defined as the number of particles inside the cell and the average velocity of these particles, respectively. Moreover, we define an error measure d$\mathbf{s}$ to track the distance of the particle centroid to the cell center. The computation of the temporal evolution of these variables consists of three distinct steps which replace the conventional full particle-tracking approach.

The first step is a *dispersion* step, where the particle lump is split into smaller parts, the quasi-particles. Each of these quasi-particles have a *dispersion velocity* $\mathbf{c}_m$, which quantifies the direction of the quasi-particle movement. It is in this step that the kinematics of these quasi-particles is calculated. The second step is the *recombination* step, where quasi-particles are recombined to form a new particle lump at the destination sites. In preparation for the next time step, the dispersion velocities are calculated at this stage as well. The third step is a *diffusion* step, which enables the modelling of Brownian motion and similar phenomena. In this paper, however, we will not discuss the diffusion step further. Interested readers are referred to [2].

In the dispersion step, we compute the force acting on the particle lump in each grid cell. Using the average velocity as a basis, an acceleration is calculated numerically from Newton's second law applied to the particle lump. Here, Newtons second law can be written as

$$\frac{\mathrm{d}\mathbf{v}_p}{\mathrm{d}t} = -\frac{1}{\tau_p}(\mathbf{v}_p - \mathbf{u}) + \left(1 - \frac{\varrho_f}{\varrho_p}\right)\mathbf{g}. \tag{2.1}$$

By solving the above equation, the particle position $\mathbf{x}$ can be found from

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{v}_p. \tag{2.2}$$

Here, we have defined

$$\tau_p = \frac{d_p^2 \varrho_p}{18\mu}. \tag{2.3}$$

The particle *relaxation time* τ_p is a measure of the particle's response to a changing fluid velocity. In Eq. (2.1), $\mathbf{v}_p$ and $\mathbf{u}$ are the particle and fluid velocities respectively, while d_p is the particle diameter, and $\mathbf{g}$ is the gravitational acceleration. Furthermore, ϱ_f and ϱ_p are the fluid and particle densities respectively, and μ is the dynamic fluid viscosity. The first term on the right hand side is the drag force on the particle, and the second term represents the combined effect of gravity and buoyancy.

A discrete solution of Eq. (2.1) is obtained by an explicit forward Euler method in time. Let $t^\ell = \ell\Delta t$ with ℓ being the time step index. A temporal discretization of Eq. (2.1) becomes

$$\frac{\Delta \mathbf{v}^\ell}{\Delta t} = -\frac{1}{\tau_p}\left(\mathbf{V}^{\ell-1} - \mathbf{u}^{\ell-1}\right) + \left(1 - \frac{\varrho_f}{\varrho_p}\right)\mathbf{g}, \tag{2.4}$$

where $\Delta\mathbf{v}^\ell/\Delta t$ is the acceleration of each individual particle in the current grid cell. We will apply this acceleration to the quasi-particles. The average velocity $\mathbf{V}^{\ell-1}$ is used as the basis for the force calculation, modeling the overall drift of the particle lump. Whereas the fluid velocity $\mathbf{u}^{\ell-1}$ is assumed to be known, either analytically or as a numerical approximation generated by a separate solver for fluid flow.

The displacement $\Delta\mathbf{x}^\ell$ of the quasi-particles will depend on the dispersion velocities. The above acceleration is then applied to the quasi-particles within a grid cell, thereby changing their dispersion velocities $\mathbf{c}_i$. Since quasi-particles attains the new velocity $\mathbf{c}_k + \Delta\mathbf{v}^\ell$, we can numerically integrate Eq. (2.2) to obtain the displacement,

$$\Delta \mathbf{x}_k^\ell = \int_{t^{l-1}}^{t^l} \mathbf{v}_p \mathrm{d}t = \left(\mathbf{c}_k + \frac{\Delta \mathbf{v}^\ell}{2}\right)\Delta t. \tag{2.5}$$

As a result, the quasi-particles are transported to their target cells, corresponding to traveling with their respective velocities in the given time increment Δt. Fig. 4 illustrates the dispersion procedure.

The distance a quasi-particle travels does not usually correspond to an integer multiple of one grid cell. As a means to track this error, an *error correction* vector $\mathrm{d}\mathbf{s}(\mathbf{x}_p)$ is introduced, which quantifies the offset of the centroid of the particle lump relative to $\mathbf{x}_p$, and can also serve as a measure of positional error. A quasi-particle displacement algorithm is used to calculate the target grid cell and the new error correction vector $\mathrm{d}\mathbf{s}^+$. Let $I_\mathbf{j}$ be the initial grid cell and $I_\mathbf{k}$ be the target grid cell, where $\mathbf{k}$ and $\mathbf{j}$ are grid cell index vectors. If $\mathrm{d}s_i$ and k_i denotes the i'th component of $\mathrm{d}\mathbf{s}$ and the grid cell index vector respectively, then these three are updated at each time step by

$$\mathrm{d}\tilde{s}_i = s_i - \left[\frac{s_i}{\Delta x_i}\right]\Delta x_i, \tag{2.6}$$

$$k_i = j_i + \left[\frac{s_i}{\Delta x_i}\right] + \left[\frac{\mathrm{d}s_i + \mathrm{d}\tilde{s}_i}{\Delta x_i}\right], \tag{2.7}$$

$$\mathrm{d}s_i^+ = \mathrm{d}s_i + \mathrm{d}\tilde{s}_i - \left[\frac{\mathrm{d}s_i + \mathrm{d}\tilde{s}_i}{\Delta x_i}\right]\Delta x_i, \tag{2.8}$$

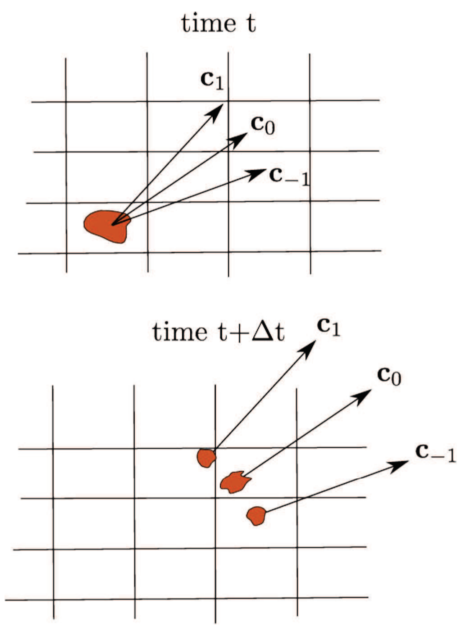


Figure 4: Dispersion of the particle lump. The particle lump is divided into three quasi-particles which are dispersed within the domain.

where s_i is i-th component of the displacement vector calculated by Eq. (2.5). Here, $[\cdot]$ denotes the closest integer value. The full derivation of this algorithm is given in [2].

In the recombination step, the quasi-particles entering the grid cells are recombined into a new lumped particle. Fig. 5 gives an illustration of this procedure. A new mean velocity $\mathbf{V}$ and error measure $\mathrm{d}\mathbf{s}$ are calculated as the averages of the respective velocities and error measures of the quasi-particles. Using a momentum balance approach, we also calculate new dispersion velocities. Further details about these calculations can be found in [2].

This completes the overview of the lumped particle model. It should be noted that many aspects of the framework has been omitted in this rendition, but we have included

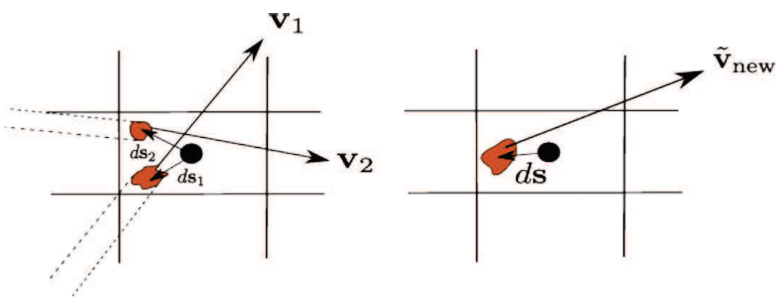


Figure 5: Recombination of quasi particles. Quasi-particles entering grid cells recombine into new particle lumps.

enough detail for the further development of the framework. For instance, recently we have extended the framework to include the effects of particle collisions in densely packed flows. In [3], we use the modelling framework to reproduce the hindered settling effect, the reduction of the effective drag force on a settling particle, to very good accuracy and with complete consistency with experiments.

3 Numerical improvements

An important issue with the simulation of particle clusters whose size range many orders of magnitude is the efficiency of the numerical model. And as outlined Section 2, the current discretization of Newton's equations uses an explicit Euler scheme. It is well known that this scheme is neither particularly accurate nor numerically efficient. The key issue here is the stiffness of the discretized equation (2.4), which places restrictions on the size of the time increment Δt. In this section, we will explore one alternative numerical scheme, and show how to apply it within the lumped particle framework.

3.1 Velocity-Verlet algorithm

First introduced in [6], the *Velocity-Verlet* numerical method for calculating velocities and positions discretely in time, which is widely used in molecular dynamics simulations [7]. The Velocity-Verlet method is often called a semi-implicit scheme, since in many cases the implicit nature of the derived equations is only superficial, meaning that explicit expressions are possible to obtain. We will first present a short derivation of the method, followed by an overview of how the Velocity-Verlet scheme is to be applied within our framework.

In the Velocity-Verlet scheme, the standard midpoint difference approximation for the derivative of the velocity v of a particle is used. We obtain

$$v(t+\Delta t)=v(t)+[v'(t+\Delta t)+v'(t)]\frac{\Delta t}{2}+\mathcal{O}(\Delta t^3). \tag{3.1}$$

Observe that all quantities at time $t+\Delta t$ are to be evaluated at spatial position $x(t+\Delta t)$, which is given by

$$x(t+\Delta t)=x(t)+\Delta t v(t)-\frac{\Delta t^2}{2\tau_p}\left[(v(t)-u(t))+\tau_p\bar{F}(t)\right]+\mathcal{O}(\Delta t^3). \tag{3.2}$$

Here, we have set $v'(t)=-1/\tau_p(v(t)-u(t))+\bar{F}(t)$, where $\bar{F}(t)$ are all the body forces acting on the particle that do not depend explicitly on the velocity. In general, this is not usually the case, specially when more complicated forces are included, such as the Saffman lift force. We are therefore assuming that the drag force is the only velocity dependent quantity. Inserting the expression for $v'(t)$ into Eq. (3.1) gives

$$\begin{aligned} v(t+\Delta t) = v(t) + \frac{\Delta t}{2\tau_p}\left[\tau_p \bar{F}(t) + \tau_p \bar{F}(t+\Delta t) + u(t) + u(t+\Delta t)\right] \\ - \frac{\Delta t}{2\tau_p}\left[v(t+\Delta t)+v(t)\right] + \mathcal{O}(\Delta t^3). \end{aligned} \tag{3.3}$$

Eq. (3.3) is in principle an implicit expression for the velocity. In its current form, however, it is possible to obtain an explicit equation by a straightforward rearrangement of the terms. We obtain

$$v(t+\Delta t) = \frac{1-\eta}{1+\eta} v(t) + \frac{\eta}{1+\eta}\left[\tau_p \bar{F}(t) + \tau_p \bar{F}(t+\Delta t) + u(t) + u(t+\Delta t)\right] + \mathcal{O}(\Delta t^3), \tag{3.4}$$

where $\eta = \frac{\Delta t}{2\tau_p}$.

3.2 Application of the Velocity-Verlet scheme in the lumped particle model

We will now show how to apply the Velocity-Verlet algorithm within the framework, effectively replacing Eqs. (2.5) and (2.4). Firstly, the body forces considered in this paper are restricted to gravity and buoyancy. Consequently, $\bar{F}(t) = \left(1-\frac{\varrho_f}{\varrho_p}\right)\mathbf{g} = \text{constant}$.

As we did previously in the explicit Euler scheme, we use the average velocity $\mathbf{V}^{\ell-1}$ as the basis for the force calculation on the quasi-particle instead of the dispersion velocities $\mathbf{c}_k$. Using Eq. (3.2), we obtain

$$\Delta \mathbf{x}_k^{\ell} = \mathbf{c}_k \Delta t - \frac{\Delta t^2}{2\tau_p^2}\left(\mathbf{V}^{\ell-1} - \mathbf{u}^{\ell-1}\right) + \frac{\Delta t^2}{2\tau_p}\left(1-\frac{\varrho_f}{\varrho_p}\right)\mathbf{g}. \tag{3.5}$$

This expression effectively replaces Eqs. (2.4) and (2.5). With this value for the displacement, we can obtain the new target grid cell with Eqs. (2.6)-(2.8), which corresponds to the spatial position where the value of $\mathbf{u}^{\ell} = \mathbf{u}(t+\Delta t)$ is evaluated. Consequently, the new quasi-particle velocity are calculated with

$$\mathbf{v}_k^{\ell} = \frac{1-\eta}{1+\eta}\mathbf{c}_k + \frac{\eta}{1+\eta}\left[2\tau_p\left(1-\frac{\varrho_f}{\varrho_p}\right)\mathbf{g} + \mathbf{u}^{\ell-1} + \mathbf{u}^{\ell}\right]. \tag{3.6}$$

These two equations are the only modifications done to the core algorithm of the lumped particle model. All other aspects of the framework is left intact. In Section 5.2, we will investigate the numerical accuracy and efficiency of this new approach.

4 Multiscale modelling approach

A fluid flow consisting of many particle types can almost automatically be accommodated within the framework of the lumped particle model. Particles of the same average size and shape can be divided into n distinct groups, each with a separate lumping procedure. With the inclusion of an external solver for the fluid phase, we need to run a set of $n+1$ simulation entities to be able to calculate the evolution of the sand-laden fluid flow. In addition to being coupled, these components require different time scales to resolve the physics they represent. What we need is a procedure to be able to calculate the solution of a series of components which represent physics on different scales.

The basic idea of our multiscale modelling approach is to resolve the physics on the shortest time scales first, then those processes that occur on higher time scales. Moreover, time averaged quantities are used in the coupling of the simulation components.

In this setting, we will denote the different simulation entities as a *data component* or simply a *component*. Instances of such components could be the simulator representing the fluid or one of the different particle constituents. Moreover, *member variables* or members are a set of quantities that is updated at each time integration step which describe the system modelled. For instance, member variables of a component describing a suspension can be the velocity and the concentration of the particles within a grid cell. Therefore, the data components can be viewed as a self-contained simulation engine responsible for calculating the state of their members. Calculating the members of a data component is what we will refer to as *to resolve* it. In this setting, resolving will mean to perform the time integration of the mathematical model represented by the data component. Note that these concepts of components and members in the simulation model directly onto an object-oriented implementation.

4.1 Sequential propagation of data components

The simulation process consists of calculating the member values of the data components during a time increment Δt, which we will refer to as the *global time step*. We assume that a characteristic time scale τ_c is associated with a given data component.

Assuming that we have m data components, we can define a natural hierarchy $\{\tau_c^1, \tau_c^2, \cdots, \tau_c^m\}$ consisting of the characteristic time scales, where $\tau_c^1 \leq \tau_c^2 \leq \cdots \leq \tau_c^m$. As will be explained further down, this set is used when prescribing the order the components are resolved. In the following discussion, the global time increment Δt is chosen to be equal to the maximum time scale of the components involved in the simulation. That is, $\Delta t = \tau_c^m$, but the method introduced here does not require this in general. In Section 5.2, we perform some experiments to compute the error of the numerical scheme. This calculation will give us some indication of how Δt should be chosen to minimize the numerical error.

The goal during the time increment is to advance the modeled system from time t to $t+\Delta t$. This is accomplished by what we call a *sequential propagation* of the components in the hierarchy. With this we mean that the first component is advanced to time $t+\tau_c^2$, after which the second component is advanced to that same time. Then the third component is resolved to $t+\tau_c^3$, after which the first and second component are advanced from $t+\tau_c^2$ to $t+\tau_c^3$. This process continues until all the components are resolved. Fig. 6 shows an

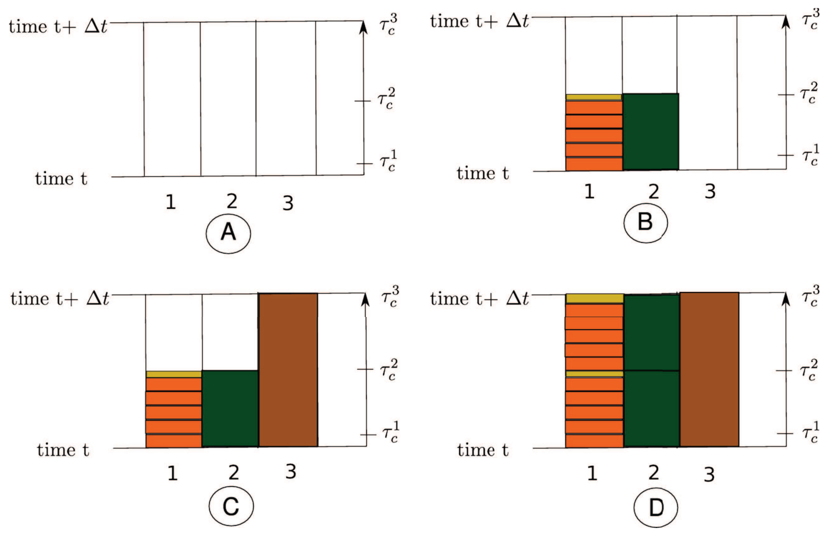


Figure 6: Multiscale time evolution of three data components. Three data components have a characteristic time scale τ_c^1, τ_c^2 and τ_c^3, respectively. During each time increment $\Delta t = \tau_c^3$, data component "1" is evolved to time $t+\tau_c^2$ in increments τ_c^1. After which, the data component "2" is evolved to time τ_c^2. This is shown in figure B above. Note the yellow bar representing the time increment $\tau_c^2/\tau_c^1 - [\tau_c^2/\tau_c^1]$. In figure C, the third data component is evolved to time $t+\tau_c^3$. Finally, data component 2 and 3 are evolved to time $t+\tau_c^3$, as shown in figure D.

example of this procedure with three data components. The order of resolving the components in this fashion is chosen to minimize the error of the calculation, since all of the components should in principle be resolved simultaneously. When resolving each data component, the time increment must be less or equal to the value required by the numerical scheme. Hence, higher time increment values sent to the component are partitioned into smaller quanta, requiring many calls to the numerical solver routine before the component is resolved. In modeling turbidity flow the difference in specific components is many orders of magnitude, often requiring 10^6 calls for the smallest particles. This *is* a big issue that will be handled in Section 4.3.

4.2 The coupling of data components

In general, the evolution of a data component is dependent on the states of other components in the simulation. For instance, a data component describing solid particles needs the velocity of the fluid to be able to calculate the drag force. Moreover, if the particle concentration is large, then the fluid component will be affected by the particle dynamics. Hence, within this multiscale framework, we require a mechanism to handle these effects. For simplicity, we will assume that the data components are coupled on the equation level only. This means that changes in the geometry, like deposition of particles on the ocean floor, will not be viewed as a data coupling, even though it is important for the components evolution.

We can define the *influence* of a component onto another component. In principle, this is a time averaged quantity of some of the members of the data components which *another* data component requires to be resolved. Usually, this is could be the fluid velocity, but it could also be some derived quantity like angular momentum. Hence, the influence of a data component are certain physical quantities that *must* be calculated during a time increment. In the beginning of the simulation, each data component instructs the others involved, which influence it requires and how to calculate it. Each time a data component is resolved, it calculates the influence required by other components. Note that an influence for a data component can be updated many times before it is actually used. Hence, a weighted time average is used when calculating the influence.

We will now outline how a coupling between data components can be accomplished. Consider three data components, each representing a sand constituent in steady fluid flow. Depending on the specific model considered, these components will interact in some manner. Assuming that the interactions can be modelled with a force term, Newton's second law on sand constituent i will be given by

$$\frac{\mathrm{d}\mathbf{v}_p^i}{\mathrm{d}t} = -\frac{1}{\tau_p^i}\left(\mathbf{v}_p^i - \mathbf{u}\right) + \left(1 - \frac{\varrho_f}{\varrho_p^i}\right)\mathbf{g} + \frac{1}{m_p^i}\sum_{k\neq i}\mathbf{F}_k. \tag{4.1}$$

Here, the force term $\sum_{k\neq i}\mathbf{F}_k$ represents the time averaged force from each of the other sand constituents. In our nomenclature, the time averaged force term $\mathbf{F}_k$ is the influence from component k on constituent i. Observe that since we are describing a flexible framework for coupling data components, it is beyond the scope of this paper to quantify the exact form of the coupling force $\mathbf{F}$. The exact form of the force $\mathbf{F}_k$ will depend on the modelling setting.

4.3 Accelerated resolving of data components

One challenge when resolving the data components is that the smallest particles usually have a relaxation time τ_p many of orders of magnitude smaller than the larger ones. This can easily be seen by noticing that $\tau_p \propto d_p^2$. Hence, the difference in relaxation time between gravel and small sand particles 6 orders of magnitudes, which roughly translates to 10^6 time loop calls for the small sand data component for every one of the gravel particles. We will in this section describe how to address this challenge, which would otherwise render our proposed method inefficient and impractical if left unresolved.

To remedy this problem, we can first consider some basic properties of the physical system. Assume that we have two data components, one representing large particles and the other the much smaller ones, with characteristic time τ_c^1 and τ_c^2, respectively. Here, $\tau_c^2 \gg \tau_c^1$. Since the forces on the smaller particles depend on these dynamical variables, the forces can be viewed as constant during the characteristic time τ_c^2. In many cases this can simplify the underlying equations for the smaller particles, even allowing for an exact solution of the equations to be found. We can therefore assume that only very small changes are possible. This is at least the case for Eq. (2.1).

Let $\mathbf{c}=(c_x,c_y)$ be the dispersion velocity of a quasi-particle, and $\mathbf{V}=(V_x,V_y)$. By assumption, all external forces $\mathbf{F}=(F_x,F_y)$ are constant in the time increment τ_c^2. Hence, the displacement $\mathbf{s}=(s_x,s_y)$ and the new dispersion velocity $\mathbf{c}^+=(c_x^+,c_y^+)$ for the quasi-particle can be obtained by the exact solution of Eqs. (2.2) and (4.1). This gives

$$c_x^+ = u_x+\tau_p^1 F_x+(V_x-u_x-\tau_p^1 F_x)e^{-\frac{\tau_c^2}{\tau_p^1}}, \tag{4.2}$$

$$c_y^+ = u_y+\tau_p^1 F_y+\tau_p^1\bar{g}+(V_y-u_y-\tau_p^1 F_y-\tau_p^1\bar{g})e^{-\frac{\tau_c^2}{\tau_p^1}}, \tag{4.3}$$

$$s_x = \left(u_x+\tau_p^1 F_x\right)\tau_p^1+(V_x-u_x-\tau_p^1 F_x)\tau_p^1\left(1-e^{-\frac{\tau_c^2}{\tau_p^1}}\right), \tag{4.4}$$

$$s_y = \left(u_y+\tau_p^1 F_y+\tau_p^1\bar{g}\right)\tau_c^2+(V_y-u_y-\tau_p^1 F_y-\tau_p^1\bar{g})\tau_p\left(1-e^{-\frac{\tau_c^2}{\tau_p^1}}\right). \tag{4.5}$$

These equations replace the standard numerical scheme described in Section 2. Note that we have also set

$$\bar{g}=-\left(1-\frac{\varrho_f}{\varrho_p}\right)g, \tag{4.6}$$

and $\mathbf{u}=(u_x,u_y)$ is the fluid velocity.

Care must be taken, however, when applying the exact solutions to the quasi-particles movement. If the quasi-particle passes through more than one grid cell during the time step, the forces from the other data components may be different from the grid cell of origin. Therefore, a simple check of the external forces is added to the framework. Here, we calculate the difference between the respective forces in the interceding grid cells. If a significant difference is found, the quasi-particle is moved to an intermediate grid cell, where the dispersion step is repeated for the respective quasi-particle. A significant difference is in this context a value higher than 5% relative difference between the forces in the two grid cells compared. In practice, however, we have yet to encounter this problem in simulations. The smaller particles typically travel one or two grid cells at the most during the time increment τ_c^2.

5 Numerical experiments

In this section we will present a series of applications of the multiscale lumped particle model, aimed as a proof of the concepts presented in this paper.

5.1 Simulation of three sand constituents in still water

Consider a set of sand particles suspended in a fluid. These sand particles vary greatly in diameter, but we can categorize them into groups with the same average size. Fig. 1 shows the relative size differences in these particles. In a sand-laden flow, the largest particles is usually called *gravel* particles, with size ranging from about 2 mm to 5mm. Intermediate sized particles with diameter from about $2.0-0.05$mm in size, are often referred to as *sand*. *Silt* are an even smaller particle group with diameters as small as 0.002mm. The smallest particles in a sand-laden flow, known as *clay*, are of size less than 0.002mm. Note that this is a very rough partitioning of the suspensions, but will suffice for the cases discussed in this paper. This simulation will serve to exemplify the coupling of data components and to show simple but important results concerning suspension flows.

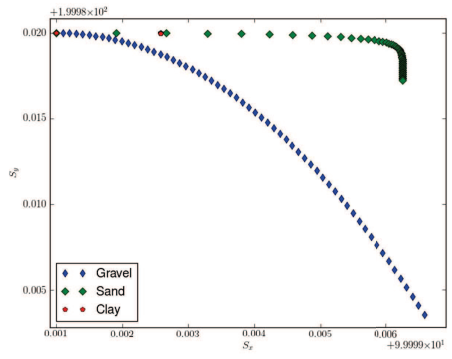
There are three particle types involved in the simulation, which are small sized gravel, large and medium sized sand. The particles are to be released with an initial vertical velocity in a fluid at rest. Hence, there are in principle four data components to be resolved at each time step. The member values of the particle components are the velocity and position. Moreover, none of the constituents interact at this stage, being only influenced by gravity and the fluid. The particles are assumed not to influence the fluid, which allows us to omit the fluid component in the simulation.

All of the particles obey Newton's second laws as described in Section 2, with the diffusion step being turned off for all constituents. The size of the particle constituents are set to 3.0 mm, 0.3mm and 0.03mm, respectively. With a particle density of $\varrho_p=2634\text{kg/m}^3$, and a fluid viscosity $\mu=1.002\times10^{-3}\text{m}^2/\text{s}$, we calculate the relaxation time for each particle constituent to be $\tau_p=0.525$s, $\tau_p=5.25\cdot10^{-1}$s and $\tau_p=5.25\cdot10^{-3}$s. The relaxation time will be used as the characteristic time scale of the data components. Observe that the relaxation time of the clay particles is 4 orders of magnitude smaller than that for the gravel particles.

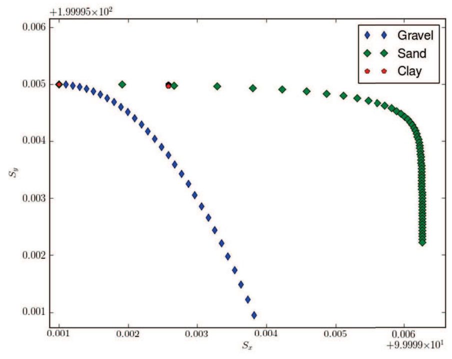
Moreover, a global time step length of $\Delta t=0.05s$ is chosen, with about 300 time steps being simulated. Note that this time step length is larger than the relaxation time for the smallest particles. To resolve the physics of these particles, we must choose a time step length less than the value of τ_p. We tackle this issue by letting the data components partition the global time step into smaller time increments consistent with this criteria, as explained in Section 4.

We use the physical domain $[0.0,20.0]\times[0.0,20.0]$, which is partitioned in 251×201 grid cells. The particles, of which there are 102500 of each constituent, are distributed evenly in a rectangular area. The fluid is set to be at rest and the gravitational acceleration g is set to 9.81m/s^2. We initialize the particles with a velocity $\mathbf{v}=(v_0,0.0)$ with, $v_0=0.1$m/s for the Gravel, $v_0=1.0$m/s for the sand particles and $v_0=30.0$m/s for the clay particles. Note that since the clay particles will be severely impeded by the fluid, we use an unnaturally high speed to be able to display their evolution easier. The boundary conditions are set to simple "no-slip" bounce back, which in this context means that quasi-particles entering a wall are simply reversed in direction. Note, however, that boundary effects are not the focus of this paper. Consequently, we will not discuss the behaviour of particles at wall sites.

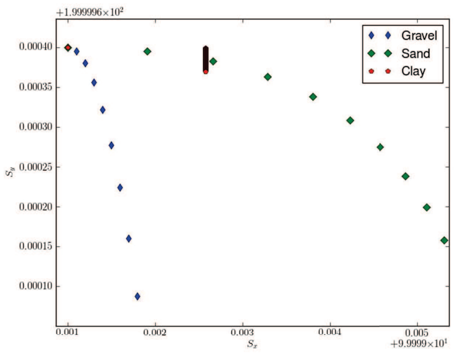
Fig. 7 depicts the results of the simulation. Here, we present three figures in different magnifications showing the center of mass displacement of the different particle



(a) Initial view



(b) A magnified view



(c) An even larger magnified view

Figure 7: Centre of mass displacement for three particle constituents in still water. Gravel, sand and clay particles are distributed in a rectangular area within a domain filled with still fluid, where only the center of mass displacement is shown. All of the particles were initialized with a horizontal velocity and are effected by gravity as well. What can be observed is the faster deposition of the larger particles compared to the smaller ones.

constituents. Firstly, we can observe a graded deposition of the particles. The larger and heavier particles gets deposited before the smaller ones, which is what to expect since one can show that the limiting velocity in the y-direction is $v_y = \tau_p \bar{g}$. Thirdly, we note that the heavier particles travel further than the lighter ones. At first glance this may be a surprise, but by solving the Newtons' equations analytically for this simple case, we can show that particles in still water will not travel further than $S_D = \tau_p v_0$. After this distance, the particles will deposit onto the ocean floor with no horizontal movement. This can also be seen in Fig. 7, where the yellow particles almost seem to stop in the x-direction. This result would change if we apply a constant fluid velocity to the particles. The smaller particles would travel much farther than the larger ones.

Finally, we tested the efficiency of the multiscale approach described in Section 4. The computation time required to simulate the system described above is usually of the order of hours. Hence, it serves as a good case to illustrate the gain in efficiency of our proposed approach. We will study two situations, one with two constituents in the physical domain, and one with three constituents, like the one described above. In both of these settings, we will investigate three scenarios consisting of different efficiency strate-

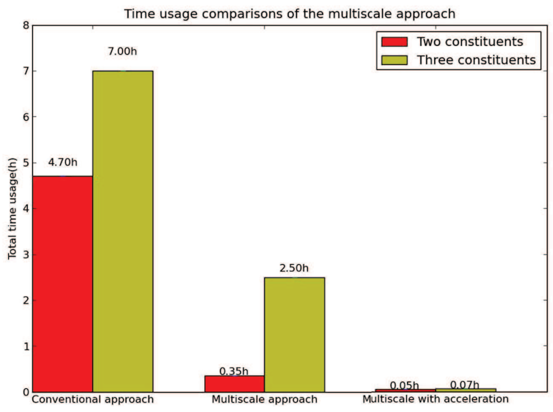


Figure 8: Total computation time of the lumped particle framework using different approaches.

gies. The particles will evolve from $t=0.0s$ to $t=15.0s$ as before, but with varying time increments depending on the chosen strategy. The first strategy is actually the "default" approach, where we use the minimum time increment required to accurately simulate the evolution of the smallest particles. For the clay particles, this time increment is $\Delta t=0.025s$ (More on accuracy in Section 5.2). The second strategy is the multiscale approach described in Section 4. Finally, the third strategy is using the accelerated approach where the exact solution is used which was discussed in Section 4.3.

Fig. 8 shows the timing saved on using the accelerated procedure as compared to using the traditional algorithm. As is shown on the chart, we can observe a substantial time saving in using the accelerated resolution of data components compared to the conventional approach. For instance, for three particle constituents, the conventional approach takes over 7 hours to complete the simulation, while it takes only a few minutes for the accelerated approach. This is approximately a 96 fold increase in efficiency for the model, which we think is a substantial improvement.

Observe that the testing of the algorithm is carried out on a high end laptop with no substantial optimization of code. It will be possible to further improve the efficiency of the lumped particle modelling framework, like moving to parallel architectures. More testing is, however, required in more physically complex settings, where more non-trivial forces on the particles are prevalent.

5.2 Numerical error trials

We want to quantify the error of the numerical model for the lumped particle framework. This has implications on how small we must choose Δt for a given particle relaxation time τ_p. Moreover, these experiments will serve as a verification of the numerical model. We will compare the results of the original numerical scheme with that of the Velocity-Verlet implementation.

We saw in the previous section how inefficiency of the explicit Euler scheme, which would be impractical to use in other than highly trivial physical settings. Consequently, we clearly require an alternative method for solving Newton's equations. In Section 3, we discussed the Velocity-Verlet scheme, and described how to apply this method within the lumped particle framework. In principle, the system of equations described by Eqs. (3.3) and (3.4) are of second order accuracy. However, with the added complexities arising with the lumping procedure, it is not clear that this property is conserved within the framework.

We will now test the accuracy and efficiency of the two different numerical schemes within our modelling framework. Restricting our attention to one particle constituent, we perform as series of experiments aimed at computing the horizontal displacement of particles with the added effects of gravity and buoyancy. Similar to the previously studied case, the displacement will converge towards the finite value $S_D=\tau_p v_0$. The particles are initiated with a horizontal speed $v_0=5.0$, with the relaxation time set to $\tau_p=2.314$s. Moreover, the physical domain is the rectangular area $[0.0,400.0]\times[0.0,40.0]$, which is partitioned in 201×201 grid cells. In each experiment, we measure the deposition distance S_D at time $t=30.0$.
For the complete series of experiments, we calculate the convergence rate

$$r_n=\frac{\log(e_n/e_{n+1})}{\log(\Delta t_n/\Delta t_{n+1})}. \tag{5.1}$$

Here, e_n is the relative error in the computed deposition distance, and Δt_n is the corresponding time increment for experiment n. For completeness, we conduct the same series of experiments using the explicit Euler scheme.

Table 1: Comparison between the explicit Euler scheme and the Velocity-Verlet scheme.

Δt	Time steps	e_n : E-Euler	r_n : E-Euler	e_n : V-Verlet	r_n V-Verlet
1.0	30	0.214	1.000	0.045	1.999
0.8	37	0.170	1.000	0.029	1.999
0.6	50	0.128	1.000	0.016	1.999
0.5	60	0.106	1.000	0.011	1.999
0.4	75	0.085	1.000	0.007	1.999
0.3	100	0.064	1.000	0.004	1.999
0.25	125	0.053	0.999	0.003	2.000
0.2	150	0.043	1.000	0.002	1.999

Table 1 shows the combined results of the numerical experiments. Moreover, Fig. 9 shows the error of the two schemes as a function of decreasing time increment. For the explicit Euler scheme, we observe that the error in the numerical scheme is substantial for high values of Δt. For $\Delta t=1.0$, which corresponds to a relative time increment $\Delta t/\tau_p=$ 0.43, the relative error is approximately 20%. Notice that the relative error is very high. As expected, the results also show that we can reduce this error by reducing the time

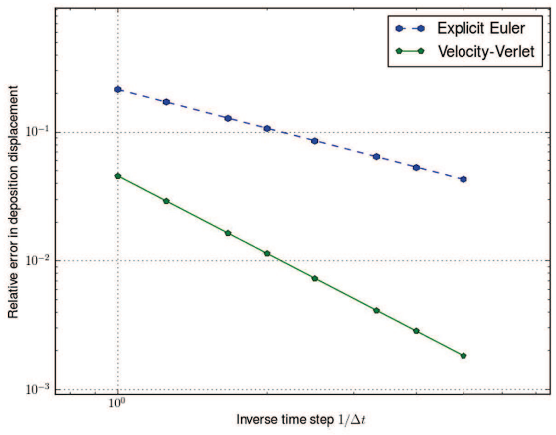


Figure 9: Relative error in the deposition distance.

increment. We can conclude that if a given physical setting allows for a relative error of 5%, the time increment for each particle constituent would have to be

$$\Delta t \lesssim 0.13\tau_p. \tag{5.2}$$

This is approximately a tenth of the particle relaxation time for each respective constituent.

If we consider the Velocity-Verlet scheme, two main results can be concluded from these data. Firstly, the Velocity-Verlet scheme is second order convergent compared to the first order nature of the explicit Euler scheme. Hence, the second order convergence rate of the Velocity-Verlet scheme is not lost when applied within our framework. Secondly, we notice a great increase in the accuracy of the results for the Velocity-Verlet scheme compared to the explicit Euler scheme. In contrast to the explicit Euler scheme, if a given physical setting allows for a relative error of 5%, the time increment for each particle constituent would have to be

$$\Delta t \lesssim 0.43\tau_p, \tag{5.3}$$

which is approximately 4 times higher than the value for the explicit Euler scheme. This signifies a substantial increase in the applicability of the lumped particle framework.

6 Conclusion

In this paper, we presented extensions to the lumped particle framework for the simulation of multicomponent particle-laden fluid flow. By considering a multiphase fluid comprised of sand of different size, we showed how to couple the time evolution of the particle constituents. The concepts developed within this paper also fit very well with an object-oriented implementation. Moreover, we presented a sequential propagation protocol for the effective computation of particle transport, and showed how to speed up these calculations for components which scale differ many orders of magnitude. Some aspects of the interactions between constituents was briefly discussed, exemplified by a force coupling term in Newtons second law.

Furthermore, a few numerical experiments on a three-component sand laden flow were discussed. These trials agreed qualitatively with results on deposition of sand particles. We discussed the numerical error of the lumped particle modelling framework. We also studied the Velocity-Verlet scheme within the framework, and showed that the efficiency and accuracy of our modelling framework were substantially increased when compared to the previously used explicit Euler scheme. Moreover, the second order convergence rate of the Velocity-Verlet scheme was not lost when applied within the framework. Consequently, for a given relaxation time τ_p and error tolerance, we obtained an upper bound for the time increment Δt.

Acknowledgments

The presented work was funded by a research grant from Statoil. The work has been conducted at Simula Research Laboratory as part of CBC, a Center of Excellence awarded by the Research Council of Norway.

References

[1] J. E. Aarnes, V. Kippe, and K-A. Lie. Mixed multiscale finite elements and streamline methods for reservoir simulation of large geomodels. Advances in Water Resources, 28(3):257–271, 2005.

[2] O. al Khayat, A. M. Bruaset, and H. P. Langtangen. A lumped particle modeling framework for simulating particle transport in fluids. Communication in Computational Physics, 8(1):115–142, 2010.

[3] O. al Khayat, A. M. Bruaset, and H. P. Langtangen. Particle collisions in a lumped particle model. Communication in Computational Physics, 10(4):823–843, 2010.

[4] J. Neufeld. The experimental nonlinear physics group, the university of toronto. `http://www.physics.utoronto.ca/ nonlin/turbidity/turbidity.html`, 2007.

[5] W. Ren and W. E. Heterogeneous multiscale method for the modeling of complex fluids and micro-fluidics. Journal of Computational Physics, 204(1):1–26, 2005.

[6] W. C. Swope, H. C. Andersen, P. H. Berens, and K. R. Wilson. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. The Journal of Chemical Physics, 76(1):637–649, 1982.

[7] M. E. Tuckerman, B. J. Berne, and A. Rossi. Molecular dynamics algorithm for multiple time scales: Systems with disparate masses. Journal of Chemical Physics, 94(2):1465–1469, 1991.