Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 68 (2008) 717–728
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

# Parallel and distributed simulation of sediment dynamics in shallow water using particle decomposition approach☆

W.M. Charles*, E. van den Berg, H.X. Lin, A.W. Heemink, M. Verlaan[1]

*Delft University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science, Department of Applied Mathematical Analysis, Mekelweg 4, 2628 CD Delft, The Netherlands*

Received 4 November 2005; received in revised form 21 March 2007; accepted 12 September 2007
Available online 8 March 2008

**Abstract**

This paper describes the parallel simulation of sediment dynamics in shallow water. By using a Lagrangian model, the problem is transformed to one in which a large number of independent particles must be tracked. This results in a technique that can be parallelised with high efficiency. We have developed a sediment transport model using three different sediment suspension methods. The first method uses a modified mean for the Poisson distribution function to determine the expected number of the suspended particles in each particular grid cell of the domain over all available processors. The second method determines the number of particles to suspend with the aid of the Poisson distribution function only in those grid cells which are assigned to that processor. The third method is based on the technique of using a synchronised pseudo-random-number generator to generate identical numbers of suspended particles in all valid grid cells for each processor. Parallel simulation experiments are performed in order to investigate the efficiency of these three methods. Also the parallel performance of the implementations is analysed. We conclude that the second method is the best method on distributed computing systems (e.g., a Beowulf cluster), whereas the third maintains the best load distribution.
© 2008 Elsevier Inc. All rights reserved.

*Keywords:* Lagrangian particle model; Stochastic differential equation; Sediment transport; Parallel processing; Speed up; Load balance; Efficiency

## 1. Introduction

The morphology of many shallow waters areas is characterised by a complex pattern in both space and time [15]. This behaviour is caused by the feedback among the *water motion*, *sediment transport* and *bottom changes*. It is clear that the morphological behaviour is very sensitive to changes in external conditions [19]. The change can be caused, for example, by the rise of sea level or human activities [4,5]. Therefore, there is a need to simulate and predict morphological processes and their sensitivity to changing conditions both for economic and ecological reasons.

In recent years, different morphodynamic models have been developed (see e.g. [11]). They are based on the physical laws describing water motion, sediment transport and bottom evolution. However, because of a lack of understanding of the processes, such models are yet not suited for simulation of morphological processes over time scales of several years and more.

In order to get accurate results from Monte Carlo simulations of sediment transport, a large number of particles is often needed. However, the computation time in a particle model increases linearly with the number of particles. In addition, as memory requirements grow, this too may be a limiting factor. Using parallel processing it is possible to both significantly reduce the simulation time as well as to alleviate memory problems. As the particle tracks are inherently independent of one another it is natural to compute these in parallel (see [8,13]), for example). It is known that parallel particle computations themselves are quite trivial. However, the sediment transport model considered in this paper has a number of features

☆ This work is partly supported by the UDSM as well as TUDelft under MHO project and the NUFFIC Fellowship award.
* Corresponding author.
*E-mail addresses:* maheraii@yahoo.com, C.W.Mahera@ewi.tudelft.nl (W.M. Charles).
[1] National Institute for Coastal and marine Management, The Hague, The Netherlands.

0743-7315/$ - see front matter © 2008 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2007.09.007

that require special attention to efficient suspension and sedimentation of particles.

For particle models two basic decomposition methods exist; those in which each processor simulates particles only in a predetermined or potentially adaptive region of the domain and communicates them to the processors managing neighbouring domains. This approach is called domain decomposition. In the second approach, particle decomposition, the particles are evenly distributed to different groups and each group assigned to a processor. This approach is very attractive because it requires only a minimal amount of communication and a good load balance is achieved relatively easy. This in contrast to domain decomposition where subdomain needs to be carefully determined to evenly spread the load. In addition, communication between processors on a parallel architecture may inhibit the speed up potential (see e.g., [3,12]). The disadvantage of the particle decomposition approach is that it requires each processor to store flow data of the entire domain which can be huge in volume.

In [8], parallelisation employed in the implementation of Lagrangian stochastic model for pollutant Dispersion is based on domain decomposition. In [13], the parallel approach is based on particle decomposition (i.e., particles are uniformly distributed across the processors). This is because they want to avoid the need of dynamic load balancing with domain decomposition after each time step and when the number of processors is increased. However, for fixed number of particles the load balance is automatically there. In our work we apply particle decomposition parallel approach on the sediment transport. However, the number of particles is not known in advance, it depends on flow dynamics. Our parallel implementation in C in Section 4 is based on particle decomposition and uses the message-passing interface (MPI) implementation for deployment on a cluster (see [3]).

For optimal use of parallel machines, it is important to maintain a good load balance as well as minimise the required communication time. Therefore, in this paper we have introduced three different suspension technique. Three different approaches to particle suspension techniques were considered (see Section 3.2). The first method consider each grid cell $i, j$ and divides expected number of suspended particles by the total number processors to obtain its local expectation over all processors. This local expectation is used as a parameter in the poisson distribution function to determine the expected number of the suspended particles that particular grid cell over all available processors. The second method is based on a (crude) form of domain decomposition approach in the sense that each processor is allotted with grid cells in which to determine the number of particles to suspend in each particular processor. The third method uses a synchronised pseudo-random-number generator to determine the number of particles to suspend in each grid cell $i, j$ and equally divides these among the processors. Since all processors draw the same numbers, they know the global state regarding to the particles to be distributed. As a result, each processor can determine the number of particles it should take care of.

The simulation of sediment transport can be done by the numeric integration of a set of stochastic differential equations augmented with erosion and deposition terms. These simulations give rise to tracks of sediment particles along their path in time, and are known as the Lagrangian particle approach. In this work we develop a parallel Lagrangian particle model for efficient sediment transport simulation in shallow waters. The inputs for our particle models, such as velocity and water depth, are time dependent and are computed by the two-dimensional hydrodynamic model known as WAQUA [16]. It is one of the oldest and most commonly used computation models at the National Institute for Coastal and Marine Management (RIKZ) in the Netherlands.

In this paper we only consider a parallel particle model for transport of sediments, the flow computation itself is not considered. The parallel computation of the flow velocity has been implemented by Roest [14], and Vollebregt [20] using a domain decomposition approach. We do not yet consider the two-ways coupling involving the flow and the Lagrangian particle model. Nevertheless, the depth changes are slow and happen after long simulation time [2] unlike the dynamics of the flows. Therefore, depth changes need not be communicated frequently. However, for application in real life (such as stratification due to concentration differences) it is possible and necessary to couple the flow and the particle model. However, a prototype for two-way coupling of the flow and particle transport models was developed in our group [10].

This paper is arranged as follows, we discuss the Lagrangian particle model for sediment transport in Section 2 followed by the parallel processing considerations in Section 3. The three different sediment suspension approaches are discussed in Section 3.2. The numerical implementation of the Lagrangian particle model is described in Section 4. Sections 5 and 5.1 respectively cover the application and the results. As a realistic case study, the parallel Lagrangian approach is applied to the Dutch coastal waters, notably the Wadden Sea.

## 2. Sediment transport using Lagrangian particle models

A Lagrangian particle model consists of particles that eventually undergo a displacement. This displacement is split into a drift part and a random part called diffusion.

Let the position of a sediment particle at time $t$ along the $x$ and $y$ directions be denoted by $(X(t), Y(t))$. The numerical integration of movement of such a particle in a flow, taking into account the effect of sea bed slope, proceeds by the following two-dimensional stochastic differential equations (SDEs):

$$\begin{aligned} \mathrm{d}X(t) &\overset{\text{Itô}}{=} \left[U\left(1+\operatorname{sign}(U)\delta\frac{\partial H}{\partial x}\right)+\frac{D}{H}\left(\frac{\partial H}{\partial x}\right)+\frac{\partial D}{\partial x}\right]\mathrm{d}t \\ &\quad +\sqrt{2D}\,\mathrm{d}B_1(t) \\ \mathrm{d}Y(t) &\overset{\text{Itô}}{=} \left[V\left(1+\operatorname{sign}(V)\delta\frac{\partial H}{\partial y}\right)+\frac{D}{H}\left(\frac{\partial H}{\partial y}\right)+\frac{\partial D}{\partial y}\right]\mathrm{d}t \\ &\quad +\sqrt{2D}\,\mathrm{d}B_2(t) \end{aligned} \tag{1}$$

where $U$ and $V$ are the flow velocities in m/s along respectively the $x$ and $y$ direction. $H$ is the sum of the (interpolated) depth

and the relative water levels due to waves at $(x, y)$. Next, $\mathrm{d}B_1(t)$ and $\mathrm{d}B_2(t)$ represent independent increments of (Gaussian) Brownian processes $B_1(t)$ and $B_2(t)$, and have mean $(0, 0)^T$ and covariance $\mathbb{E}[\mathrm{d}B_1(t)\mathrm{d}B_2(t)^{\mathrm{T}}] = I\mathrm{d}t$ with $I$ the identity matrix as in [6,1]. (More information on this concept can be found in [7]). Finally, $D(x, y, t)$ is the horizontal dispersion coefficient for sediment transport, $\delta$ is a constant coefficient for the slope of the sea bed. Typically, $D = \mathcal{O}(10-100)$ m$^2$/s [15]. The effects of the slope of the sea bed is incorporated in the drift part and therefore influences the flow velocity. By setting $\delta = 0$, we can choose to omit this dependency.

The drift term consists of the flow fields as well as a correction term. This prevents particles from accumulating in a location with low diffusivity [18]. A common choice for the random part is the Gaussian probability function [1,7].

For sediment transport modelling, a number of subprocesses have to be added to the description of pure movement to cover erosion, deposition and behaviour at the sea bed [11]. In the remainder of this section we study these additional processes and consider suspension, sedimentation, and in and outflow at domain boundaries. This is followed by a discussion about the resulting changes in bed level.

### 2.1. Deposition of sediment particles

We associate with each sediment particle a binary state which at any time $t$ is given by

$$\mathrm{S}_t = \begin{cases} 1 & \text{particle is in suspension} \\ 0 & \text{otherwise (particle is on the sea bed).} \end{cases}$$

Given a particle in suspension, we are interested in the transition of state 1 to state 0. In a continuous form, this transition can be modelled by the following equation

$$\frac{\mathrm{d}P(S_t = 1)}{\mathrm{d}t} = -\gamma(x, y, t)P(S_t = 1), \quad \text{where initially}$$
$$P(S_0 = 1) = 1 \tag{2}$$

where $\gamma(x, y, t)$ is the deposition coefficient, in this paper $\gamma = \gamma(x, y, t)$ is constant, $P(S_t = 1)$ is the probability that the state of the particle at time $t$ is 1. When the particle is in suspension, its evolution is described by the following transition probability equation in discrete form:

$$\begin{aligned} P(S_{t+\Delta t} = 1 \mid S_t = 1) &= P(S_0 = 1) \cdot [1 - \gamma \cdot \Delta t] \\ &= [1 - \gamma \Delta t] \\ &\approx \exp(-\gamma \cdot \Delta t). \end{aligned} \tag{3}$$

Assuming the system state (e.g. flow field and turbulence patterns) to be constant during the period of the time step, it follows that the probability that the particle will be sedimented is given by

$$P(S_{t+\Delta t} = 0 \mid S_t = 1) = 1 - [P(S_{t+\Delta t} = 1 \mid S_t = 1)].$$

### 2.2. Suspension of sediment particles

Mass represents concentration of a group of particles at a certain location. In order to satisfy the law of conservation of mass, a source term is included in our particle model such that the expected number of suspended particles ($enp$) in grid cell $i, j$ at time $t$ is given by

$$enp_{(i,j,t)} = \frac{\Delta x \cdot \Delta y \cdot \Delta t \cdot \left(U^2 + V^2\right) \cdot \lambda_s}{\mathcal{M}_p}, \tag{4}$$

where $\mathcal{M}_p$ is the mass of each particle, $\Delta x$ and $\Delta y$ are the width of the grid cells along $x$ and $y$ directions respectively, $\Delta t$ is the time step size, and $\lambda_s$ is the erosion coefficient. To determine the actual number of particles to be suspended in a grid cell $i, j$, we draw a number from a Poisson distribution function with mean $enp_{(i,j,t)}$ (see Section 3.2).

#### 2.2.1. Particle flux at open boundaries

We now consider particle flux at open boundaries. This flux is the difference between particles flowing into and out of the domain. The number of particles flowing out should not be controlled as it is a natural consequence of the movement of a particle. As soon as a particle crosses an open boundary it is considered to be gone and further integration is no longer possible as no data outside the domain is given. For this very reason, however, we do need to explicitly model the particles flowing in. We determine the expected number of particles entering the domain as follows:

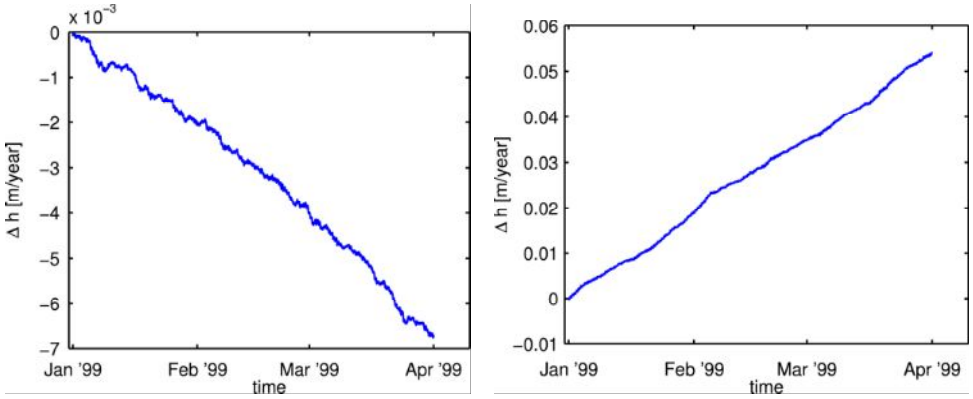
$$enp_{(i,j,t)} = \begin{cases} \dfrac{\Delta y \cdot \Delta t \cdot V \cdot (U^2 + V^2) \cdot \lambda_s}{\gamma \mathcal{M}_p} \\ \quad \text{inflow parallel to } y\text{-axis} \\ \dfrac{\Delta x \cdot \Delta t \cdot U \cdot (U^2 + V^2) \cdot \lambda_s}{\gamma \mathcal{M}_p} \\ \quad \text{inflow parallel to } x\text{-axis,} \end{cases} \tag{5}$$

where $\gamma$ is the deposition coefficient. The actual number of particles added at the domain boundary at each iteration is obtained by drawing a value from the Poisson distribution parameterised by the above expectation value.

### 2.3. Boundaries

One problem with numerical integration of particle positions arises in the vicinity of boundaries. Given the current location, $(X(t), Y(t))$, we may find that the new location, $(X(t + \Delta t), Y(t + \Delta t))$, is on the other side of a boundary, i.e. the particle has crossed a boundary. Depending on the type of boundary this may be physically impossible. We consider two types of boundaries. The first type, closed boundaries, represents boundaries intrinsic to the domain such as banks, sea bed, and coast lines. The second type of boundaries are open boundaries, which arise from the modeller's decision to artificially limit those regions outside which are not of interest or simply because no domain information is available at those locations. It is clear that it is undesirable to have particles cross the first type of boundary, whereas for the second type it is quite natural. Based on this classification, we apply the following rules to particles crossing borders during integration;

- In case an open boundary is crossed by a particle, the particle remains in the sea but is now outside the scope of the model and is therefore removed.




(a) Erosion. (b) Deposition.

Fig. 1. Cumulative local changes in depth by using data of 90 days for two selected grid cells.

- In case a closed boundary is crossed by a particle during the drift step of integration, the step taken is cancelled and the time step halved until the boundary is no longer crossed. However, because of the halving, say $n$ times, the integration time is reduced to $2^{-n}\Delta t$, leaving a remaining $(1-2^{-n})\Delta t$ integration time, which, at a constant step size, requires at least another $2^n - 1$ steps in order to complete the full time step $\Delta t$. Note that at each of these steps it may be needed to further reduce the step size. This further reduction applies only to the current time step, leaving the step size of following substeps unaffected. This method effectively models shear along the coastline.
- If a closed boundary is crossed during the diffusive part of integration, the step size halving procedure described above is maintained with the modification that in addition to the position, the white noise process is also restored to its state prior to the invalidated integration step. The process of halving the time step and continuing integration with the reduced step size is repeated until the full $\Delta t$ time step has been integrated without crossing a boundary.

### 2.4. Determination of bed-level changes using Lagrangian models

Recall that each particle represents a mass of sediment material. For Lagrangian particle models, the approximate change in mass with respect to time, in each grid cell $i, j$ is determined by the following equation:

$$\frac{\partial m}{\partial t} = \frac{\Delta \mathcal{N}_p}{\Delta t}\frac{1}{\Delta x \Delta y}\mathcal{M}_p. \tag{6}$$

whereas the change in water depth can be done using:

$$\frac{\partial h}{\partial t} = \frac{\Delta \mathcal{N}_p}{\Delta t}\frac{1}{\rho_s(1-po)\Delta x \Delta y}\mathcal{M}_p.$$

Here, $\mathcal{M}_p$ is the mass of each particle, where $\rho_s$ and $po$ denote respectively the density of an individual grain particle and the sea bed porosity. $\Delta \mathcal{N}_p$ is the difference between the number of sedimented and suspended particles at each iteration. From [15], we find $\rho_s = 2650\ \mathrm{kg/m^3}$ and $po \approx 0.4$. The cumulative change in water depth $\Delta h$ (or sea bed level $-\Delta h$) in metres over all iterations is determined by the following equation:

$$\Delta h = \int_0^T \frac{\partial h}{\partial t}\mathrm{d}t, \qquad \Delta h \approx \sum \frac{\Delta \mathcal{N}_p}{\rho_s(1-po)\Delta x \Delta y}\mathcal{M}_p. \tag{7}$$

As an illustration of the above, the local depth change in two of the selected cells were plotted in Fig. 1. The first grid cell, located at $(x, y) = (138\ \mathrm{km}, 600\ \mathrm{km})$, shows a steady erosion as seen in part (a) of the figure. The second cell at $(x, y) = (136\ \mathrm{km}, 595\ \mathrm{km})$, on the other hand shows a steady deposition as is shown in part (b).

For the implementation details of this particle model, see Section 4.

## 3. Parallel processing

Having discussed the particle transport model for sediment transport, we now move on to introducing a parallel approach for speeding up the sediment transport simulation.

### 3.1. Work decomposition

In our work, the computation domain of the particle model is characterised by a grid representing the geometry of the coastal waters and particles representing sediment [9]. There are several ways in which the computation can be divided and assigned to processes. The most notable two of these are domain decomposition and particle division. Domain decomposition is often applied in Eulerian hydrodynamic models [9] such as the two-dimensional hydrodynamic model WAQUA [20]. We mention this particular model here not only as an example but also because it was used to compute the velocity of water movement used in our simulation. Note that these data such as velocity, depth, salinity, pressure gradient and so forth, are computed offline and stored in files which are subsequently read in by our program.

In the Lagrangian particle simulation the amount of work is essentially proportional to the number of particles to be tracked. The loading of flow information from files is considerable but does not dominate the processing time.

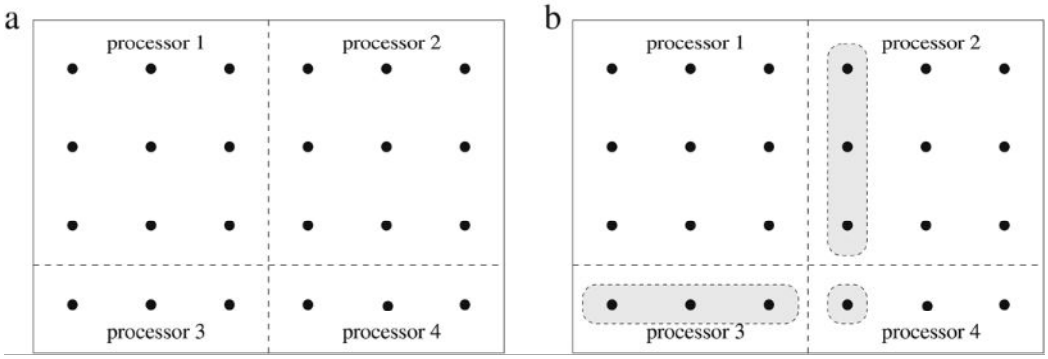



Fig. 2. (a) Domain decomposition with different parts of the domain assigned to different processor, (b) ghost regions for processor 1 for the same decomposition, indicated by the shaded boxes.

Below we consider domain and particle decomposition approaches. The goal is to find a partitioning in which the amount of parallel computation is maximal and the communication minimal.

*3.1.1. Domain decomposition*

In domain decomposition the domain is divided into a number of subdomains and the work corresponding to each such subdomain is assigned to a processor (see Fig. 2(a)). The domain decomposition approach is relatively easy to implement and can use the same data distribution over the processors as those in the hydrodynamic model e.g., [9]. There are also a number of notable disadvantages with domain decomposition (more information on parallel processing concepts can be found in e.g., [12]);

(1) Differences in flow velocity and particle density make it difficult to partition the domain in such a way that each processor handles the same amount of computation.
(2) The domain may need to be re-partitioned after a number of time steps to maintain a good load balance. It also needs to be done whenever the number of processors used is changed.
(3) Particles (as well as their tracks) have to be communicated between the processors when particles cross the subdomain boundary. Besides, very irregular size of the communication requires sending both size and data. Note that the communication of the particle data is between the neighbouring processors only, but when the subdomain becomes small (*i.e.* as more processors are used), the amount of particles moving in and out a subdomain can be large, and resulting in a relative large communication overhead.

The management of the number of sedimented or suspended particles and their influence on the bed level, or likewise base depth of the water, can be done locally since processes handled the particles that sedimented or suspended. Particles near region boundaries can fluctuate in such a way that they need to be repeatedly communicated between the neighbouring processes. To avoid this undesirable situation, one commonly introduces a small amount of overlap known as ghost regions which are indicated in Fig. 2(b) by the shaded boxes. If the boundary of a ghost region itself is crossed, the particle will end up well within the domain handled by the other process. In order to keep the state of the ghost regions coherent, changes in depth within this region on one processor need to be communicated to processes sharing that boundary region.

*3.1.2. Particle decomposition*

In this approach the number of particles is evenly divided across the available processes. This way the workload will be balanced under the assumption that the amount of calculations for each particle is more or less equal. Only slight load imbalance arises from particles stranding or flowing out of the domain and differences in the number of particles sedimented on each processor. Note that these processes can be considered stochastic with certain mean, so that over longer periods of time this amount will be the same on each processor leading to an overall stable load distribution without any need for communication. A similar argument applies for suspension.

*3.2. The process of sediment suspension*

In this section we discuss three parallel particle suspension approaches. The relative performance in terms of speed up and efficiency will be considered in Section 5.1.

*3.2.1. Method I — Modified expectation*

For each grid cell $i$, $j$, the global expectation, i.e., the total number of suspended particles expected in that cell over all processors, is computed and divided by the total number of processors to obtain the local expectation. Using this local expectation, we drawn a number from a Poisson distribution with appropriate parameter and use this as the number of local particles, *i.e.* on this processor, to be suspended in that grid cell.

(i) For each grid cell $i$, $j$ that is part of the defined domain, calculate the expected number of particles to be suspended by

$$\mathrm{enp}_{i,j} = \frac{\Delta x \cdot \Delta y \cdot \Delta t \cdot \lambda_s}{\mathcal{M}_p}. \tag{8}$$

(ii) Determine the local number of particles to suspend in each grid cell $i$, $j$

$$\mathrm{nLocal}_{i,j} = \left\lfloor \mathrm{Poisspdf}\left(\frac{enp_{i,j}}{np} \cdot (U^2 + V^2)\right)\right\rfloor$$

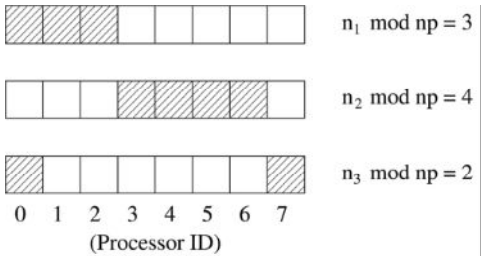



Fig. 3. Assigning $n_k \mod \text{np}$ particles equally among processors for three successive cells $k$. For the first cell we have a remaining 3 particles to assign to 8 processors, which is indicated by the shaded blocks. For the second cell we have 4 which are assigned to processors 3–6. Finally, a remaining 2 particles are assigned to processors 7, and 0.

where Poisspdf denotes the Poisson distribution function. The Poisson distribution was chosen because it has an unbiased mean and is memoryless.

(iii) Add the given number of particles to the data structure.

### 3.2.2. *Method II — Assign domain regions to processors*

This method applies a rudimentary form of domain decomposition in which each processor is systematically assigned a number of grid cells. Suspension is done only for those cells assigned to the processor, the other cells are handled by other processors. The method is easy to implement but the domain again needs to be decomposed with the inherent problem of maintaining a non-biased load distribution. The global expectation of the Poisson distribution of the grid cells $i, j$ is determined as in the sequential version, but used only to determine the number of suspended particles on one particular processor. The approach can be summarised as follows.

(i) Determine the number of grid cells available and assign them to processors according to the following equation:

$$p_{ij} = (i + j \cdot M) \bmod \text{np},$$

with $0 \leq i < M$ and $0 \leq j < N$ the row and column index, $M$ the number of rows and $N$ the number of columns in the cell grid. This way processors are randomly spread over the domain and may lead to relatively good load balance.

(ii) Next compute $enp_{i,j}$ in those grid cells allocated to processor ID using Eq. (8).

(iii) Determine the local number of particles to suspend in local grid cell $i, j$

$$\text{nLocal}_{i,j} = \left\lfloor \text{Poisspdf}(enp_{i,j} \cdot (U^2 + V^2)) \right\rfloor.$$

(iv) Finally add the number of particles to the data structure.

### 3.2.3. *Method III — 0 Globally synchronised*

For each cell $i, j$ a random number is drawn from a special, globally synchronised random number generator according to the distribution used in the sequential program (see Section 3.2.4). This random number, say $n$, represents the total number of particles to suspend in that cell and is identical on each processor, without communication. Each processor is initially assigned $\lfloor \frac{n}{\text{np}} \rfloor$ particles. The remaining $n \mod \text{np}$ particles that cannot be evenly distributed are circularly assigned to processors (one to each candidate processor, see Fig. 3). This way the number of particles is guaranteed to remain perfectly balanced with differences of at most one. In summary; for each grid cell $i, j$ within the domain, compute $\text{enp}_{i,j}$ using Eq. (8) and perform the following steps.

(i) Save the state of the current random number generator and switch to the synchronised global number generator.

(ii) Determine the global number of particles to suspend for this cell by drawing from the following distribution:

$$\text{nGlobal}_{i,j} = \left\lfloor \left( \text{Poisspdf}(enp_{i,j} \cdot (U^2 + V^2)) \right) \right\rfloor.$$

(iii) Restore the state of the local random number generator.

(iv) Determine the local number of particles to suspend for each grid cell inside the domain

$$\text{nLocal}_{i,j} = \left\lfloor \frac{\text{nGlobal}_{i,j}}{\text{np}} \right\rfloor.$$

Add one particle for processors $p$, whenever $(p + \text{np} - \text{offset}) \bmod \text{np} < (\text{nGlobal}_{i,j} \bmod \text{np})$.

(v) Set offset $:= (\text{offset} + \text{nGlobal}_{i,j}) \bmod \text{np}$.

(vi) Initialise and add the particles to the data structure.

### 3.2.4. *Global random number generator*

In order to maintain good load balance throughout, it is important that the number of particles suspended on each processor differs only a little and ideally be equal. By explicitly synchronising the number of particles intended for suspension globally, it is possible to achieve this ideal distribution. This does come at a cost however, because of the communication overhead. By careful manipulation of the pseudo-random-number generator we can draw random numbers that are the same on each processor. We assume that uniform (pseudo-)random numbers are generated according to the following equation:

$$x_{k+1} \equiv a x_k + b \bmod c,$$

where $a, b$ and $c$ are suitable chosen constants, $x_k$ the previously drawn random number, $x_{k+1}$ is the next random number and $x_0$ represents the initial seed value. Given this, we proceed as follows.

Upon program initialisation we choose (i.e., based on the current time, an input parameter or otherwise) a seed value, say $s$, on the root processor which is broadcasted to all other processors. They then manipulate this number using their processor identifier to obtain an initial local seed value (*e.g.* $s_p = s + p + 1$, where $p$ is the processor ID). The global seed value that was broadcasted however, is stored as well for global random number generation. All we then need to do is switch between the generators, based on these two seeds, at appropriate times. If the mathematics library of the programming environment stores the most recently drawn random number, $x_{k+1}$, it suffices to have one global variable storing this number when switching between random number generators (obviously, not until after the present value holding the current seed of the generator to switch to has

been copied). No further modifications to the standard random number generator is needed. If, on the other hand, no such functionality is provided a wrapper function posing as the random number generator needs to be written. This implies that the standard generator cannot be used anymore which has the disadvantage that other functions such as drawing from a normal distribution that rely on it have to be rewritten. This will be especially difficult when they are provided in standard libraries that do not offer the possibility of setting the random number generation function.

### 3.3. Sedimentation

Sedimentation itself is done on a particle basis and can therefore be perfectly done in parallel. The only disadvantage is that we have no control over the number of particles sedimented and this can therefore differ slightly between processors. This effect is minimal however, because of the law of big numbers in statistics. We expect, after all, to have many particles, otherwise parallel simulation would not be needed in the first place. Note that when the probability of sedimentation is very small, hardly any particles are sedimented (see Section 2.1), and large deviations in load balance are highly unlikely.

### 3.4. Equilibrium determining the expected number of particles in flow

An equilibrium in the number of particles can be attained when the expected number of suspended and sedimented particles are equal:

$$\bar{n}_t = \frac{\lambda_s \Delta t}{\mathcal{M}_p(1 - e^{-\gamma \Delta t})} \left[ \Delta x \Delta y \sum_{(i,j)\in\mathcal{D}} w + \frac{\Delta x}{\gamma} \sum_{(i,j)\in\mathcal{B}} U w + \frac{\Delta y}{\gamma} \sum_{(i,j)\in\mathcal{B}} V w \right],$$

where $w$ is defined as

$$w = U^2(x, y, t) + V^2(x, y, t),$$

while $\mathcal{B}$ and $\mathcal{D}$ represent the set of grid cells at the boundary respectively within the domain such that $\mathcal{B} \cap \mathcal{D} = \emptyset$ and $\mathcal{B} \cup \mathcal{D}$ encompasses all valid grid cells. The number of particles at equilibrium determines the amount of work to be done per time step. Most terms in the above equation for $\bar{n}_t$ are independent of $\Delta t$ and therefore, the expected amount of work per iteration is $\mathcal{O}\left(\frac{\Delta t}{1-e^{-\gamma \Delta t}}\right)$. Given the fact that there are $\mathcal{O}(1/\Delta t)$ iterations in a simulation of fixed time interval, the total amount of work is $\mathcal{O}\left(\frac{1}{1-e^{-\gamma \Delta t}}\right)$. As expected, the total amount of work goes down as the step size increases.

### 3.5. Depth update

The global effect of suspension and sedimentation on the sea bed level cannot be determined without communication, since each processor handles only a subset of all particles. As a result, they only know about the changes in bed level resulting from particles under their control. There are several ways in which the depth can be updated globally.

(a) Communicate all sedimented and suspended particles between all processor pairs (an optimization can be done here because we can already know the number of particles suspended in a certain region as this is done the same on each processor). This approach is preferable when the number of particles is smaller than the number of grid cells. However, all-to-all communication of the sedimented and suspended particles becomes impractical when the total number of particles is large. In other words, this way of handling particle updates does neither scale very well with the number of particles nor with the number of processors. The advantage is that communication does not need to be done after each iteration, unless the morphology changes during each iteration are significantly large.

(b) Locally determine the depth changes in the grid cells and collect this information. This is another way to reduce communication by locally computing of the grid cell depth changes and use a reduction scheme to aggregate all changes. The accumulation of the depth changes can be done over multiple time steps per processor before they are communicated as there is a large difference in time scales. This approach is preferable when the number of particles is larger than the number of grid cells.

(c) Determine the depth changes in the grid cells and communicate only those grid cells that have a change of depth. Here, the cell index along with the number of particles sedimented or suspended needs to be communicated to the other processors.

*Advantages of particle decomposition*

The advantages of the particle decomposition approach are:

- The particle division parallel approach is domain independent.
- No lengthy computations prior to the particle simulation are needed.
- It scales well with the number of processors used, without having to re-initialise the domain.
- Load balance can be maintained very well when using the third approach discussed in (Section 3.2.3); suspension can be done while maintaining a perfect load balance without any communication by careful manipulation of the seed value used in the (pseudo-)random-number generator [3].

However, the disadvantages of the particle decomposition approach are

- It requires each processor to store flow data of the entire domain which can be huge in volume.
- When using a two-way coupling of flow and particles the situation arises that domain decomposition is used for the flow computation while particle decomposition is used for sediment transport simulation. This leads to two related problems. First, in order to compute depth changes needed by the flow computation, all local influences

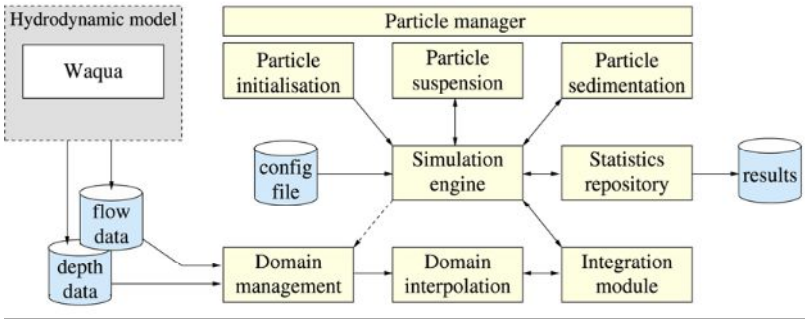



Fig. 4. An overview of the software for the parallel particle model for sediment transport in shallow water.

(as computed on each processor) need to be summed among processors and communicated to the processor dealing with this region of the domain. This global summation must be done to obtain the depth information needed by each processor, constituting a global all-to-all reduction operation. The second problem arises from the fact that new flow information is known only to the processor which was assigned to compute them while they are needed on each processor. This requires yet another expensive communication step in which all this local information is broadcasted to all other processors. During the transport simulation, however, no additional communication is needed.

## 4. Implementation

The following, briefly describes the implementation of the Lagrangian particle model for sediment transport in shallow water as described in Section 2. The implementation consists of a set of modules, shown in Fig. 4, working together through predefined interfaces, simplifying replacement of one model implementation by another. The integration module, for example, is given a set of particles, the current time, and the step size and performs numerical integration of the SDEs governing particle movement. The particular choice of scheme used does not affect the other modules. When, for example, a different integration scheme is needed, one can either replace the implementation of the original integration code, or have the integration module accept parameters indicating which of the available schemes should be used. The latter approach of module specific parameters can be implemented in each module and allows for a high level of flexibility, without having to recompile the simulation. This facilitates automatic processing of simulations with different parameters, and can be used to analyse what-if scenarios.

### 4.1. Modules

*Hydrodynamic model*: The hydrodynamic model carries out the flow computation and saves the flow velocity at every chosen time step as well as the depth information and write them in a collection of files. It provides all the domain information required by the transport model for simulation. In our case, the hydrodynamic software used is known as water quality model (WAQUA) see e.g., [16,20], a software package of the directorate General for public works and water management in the Netherlands.

*Simulation engine*: The simulation engine forms the core of the whole simulation. It starts by processing the command line argument and, if needed, loading one or more configuration files. It then triggers the initialisation of all modules and provides them with the parameters given. After all modules are successfully initialised, it enters the main time stepping loop. At each iteration it requests the domain module to ensure the current data are available, does the suspension, integration and sedimentation and instructs the statistics repository to extract all relevant data from this iteration. Finally it requests all modules to finalise and stop the application. Throughout, the simulation engine does all the required (parallel) error checking, and if needed, the graceful abortion of all parallel instances.

*Domain management*: This module performs the loading, caching and overall management of domain data and is triggered by the simulation engine to prepare data for a specific time, say, $t$. In addition it also provides interpolation of data at two consecutive times, although this is completely hidden from all other modules and can be requested by specifying the appropriate parameters. Upon request is provides the flow and depth data at grid points.

*Domain interpolation*: The domain interpolation takes care of interpolating the data given at grid points to flow, depth and domain information at any location. The domain information describes whether the given location is within or outside of the domain, whether the grid cell it is contained in is an open or closed boundary or some region within the domain.

*Particle initialisation*: The purpose of this module is twofold; it generates the particles that exist at the start of the simulation, and it initialised the data associated with each particle, such as its weight, type, or for example history of a noise process driving the particle.

*Particle manager*: Memory for particles is allocated dynamically and therefore needs to be freed when it is no longer needed, to avoid running out of available resources. In order to avoid repeated allocation and freeing of particles, the particle memory recycles them and allocates new ones only if no recycled particles are available.

*Integration*: Provided with a list of particles, the current time and the duration of the time interval the integration module

Table 1
Average run times of the sediment transport model on the Beowulf cluster

| CPU | 1 | 2 | 8 | 16 | 25 | 28 |
|---|---|---|---|---|---|---|
| Time of method I | 8223.14 | 4055.10 | 1050.78 | 548.82 | 370.20 | 341.07 |
| Time of method II | 8172.41 | 4020.95 | 997.23 | 503.64 | 329.76 | 296.83 |
| Time of method III | 8188.08 | 4101.05 | 1058.31 | 569.42 | 393.28 | 364.86 |

numerically applies the SDEs governing particle motion on the particles.

*Particle suspension*: At each iteration, this module determines how many particles to suspend in each grid cell. This is done according to the description given in Section 2.2. The particles are then created and initialised (with random location within the cell) and returned to the statistics engine.

*Particle sedimentation*: After integration, this module is given a list of all local particles and it determines which of those particles are subject to sedimentation (see Section 2.1). As such, the list is split into one of the particles to keep and one of the particles to be sedimented.

*Statistics repository*: The statistics repository collects information about, amongst others; the number of particles in the flow at each iteration, the particle tracks, and the amount of sedimentation and suspension in cells, The information is globally synchronised at the ending of the program and output to a data file.

## 5. Application

In this section, the Lagrangian particle model designed for the simulation of sediment transport is applied to the Dutch coastal waters in the Wadden sea (see Fig. 7). The parallel processing experiments are carried out on a distributed memory parallel architecture called DAS-2 [17]. It is a 200-node wide-area distributed system spread out over five clusters throughout the Netherlands with a total of 400 processors. The system was built by IBM and runs on RedHat Linux. It is funded by NWO (the Netherlands organization for scientific research) and the participating five universities. Each node contains:

- Two 1-Ghz Pentium-IIIs of at least 1 GB RAM on each node.
- A Myrinet interface card connected with a high-level 3 switch.
- A Fast Ethernet interface (on-board).

The nodes within a local cluster are connected by a Myrinet-2000 network. It is used as a high-speed interconnection, mapped into user-space. In addition, Fast Ethernet is used as OS network (file transport). A Myrinet-2000 is a switched network, capable of full-duplex data rates up to 2+2 Gbits/s and has low latencies in range of a few micro-seconds.

### 5.1. Experimental results

In this section we carry out a number of experiments using realistic input data of the Wadden sea for the simulation of the sediment transport. The domain consists of $201 \times 225$ grid cells of $\Delta x = 800$ m by $\Delta y = 800$ m. Many other simulation parameters are summarised in Table A.1. In this article we have only used the DAS-2 cluster based at the Delft University of Technology which has 32 nodes. The experiments for a time period of 2 days have been carried out using one CPU on each node. For each processor we measure the total execution time (see Table 1 and Fig. 5(d)).

Just to give an idea about execution time of the part of the flow computation for a time period of 2 days, the parallel execution time using 2 processors is roughly 975 s compared to more than 4000 s for the sediment transport. Therefore, the particle simulation requires more computing time in this case.

The speed up, efficiency and load balance for different suspension methods are measured using the minimum total execution times and the results are presented in Fig. 6. The total number of particles in each processor summed over the iterations when 25 processors were used is shown in Fig. 5(a). The difference between maximum and the minimum number of particles among the processors is determined at every iteration for each suspension methods (see Fig. 5(b)). From Fig. 5(b) it can be noticed that the suspension due to method III has relatively good load balance at every iteration. The load distribution in all three approaches at every iteration is almost the same (see Fig. 5(c)). In addition, Fig. 6(a) shows the parallel speed ups achieved by the three methods. It can be seen that they are all close to the ideal linear speed up. Especially method II attained a very good speed up even for a large number of processors. The super linear speed arises from the use of cache. The efficiency for all three methods is shown in Fig. 6(b).

Table 1 shows that method II requires less computation time than method III. This seems to contradict the somewhat better load balance achieved by method III mentioned above. This can be explained by looking at the typical number of sediment particles in the flow and comparing this to the number of grid cells. Taking for instance the execution on 25 processors, we found that there were on the average 25,000 particles per processor. Each particles requires an integration in each time step, corresponding to at least one computation of velocity and depth at the current location as well as the numerical integration itself. After each integration step we need to determine the particle sedimentation and suspension. The time required for sedimentation depends on the number of particles in the flow giving a slight advantage for method III. For suspension within a grid cell we need to compute the flow information at the centre of that cell and determine how many particles will be suspended. Computing the flow and depth information takes a significant amount of time and requires access to the large arrays storing domain information. This is the point where method II excels, because each processor needs compute this information only for the fraction $1/np$ of all cells (which totalled to about 40,000 in our case) whereas method III

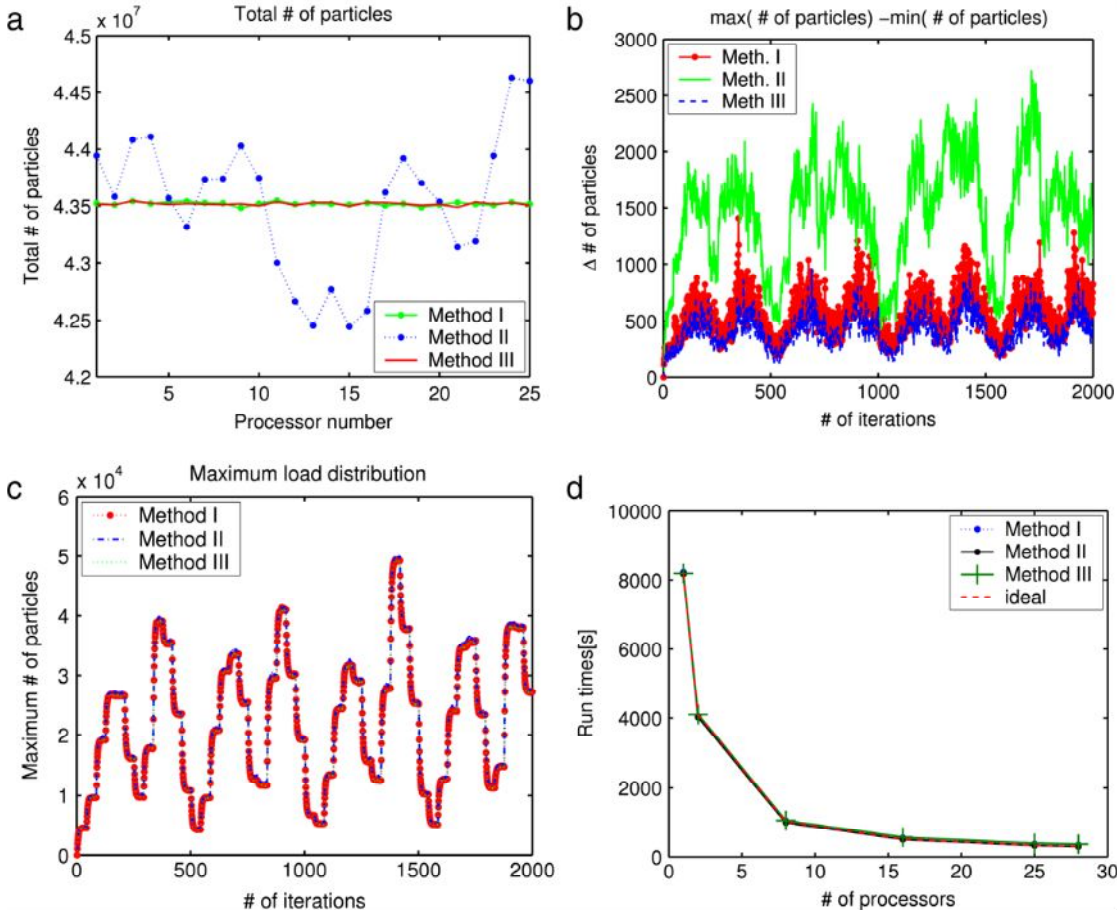



Fig. 5. (a) The total number of particles in each processor summed over all the iterations for each approach (b) The difference between the maximum and the minimum number of particles among the processors at every iteration for each approach; (c) The maximum number of particles in a processor at every iteration for each approach.

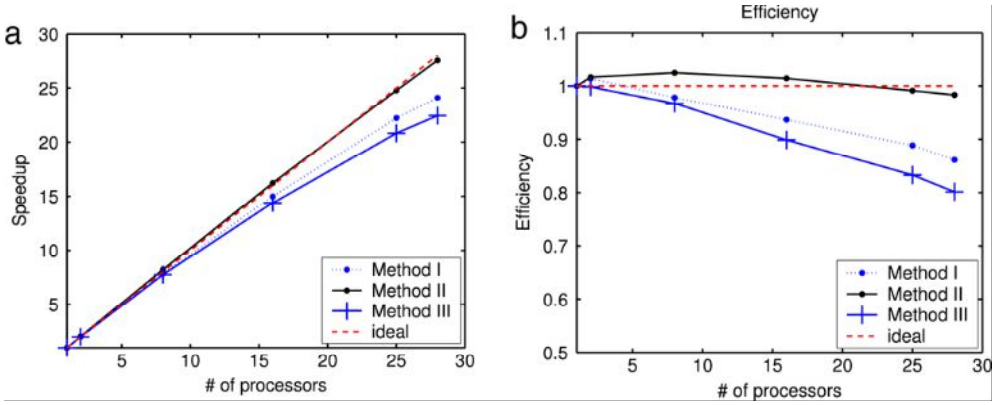
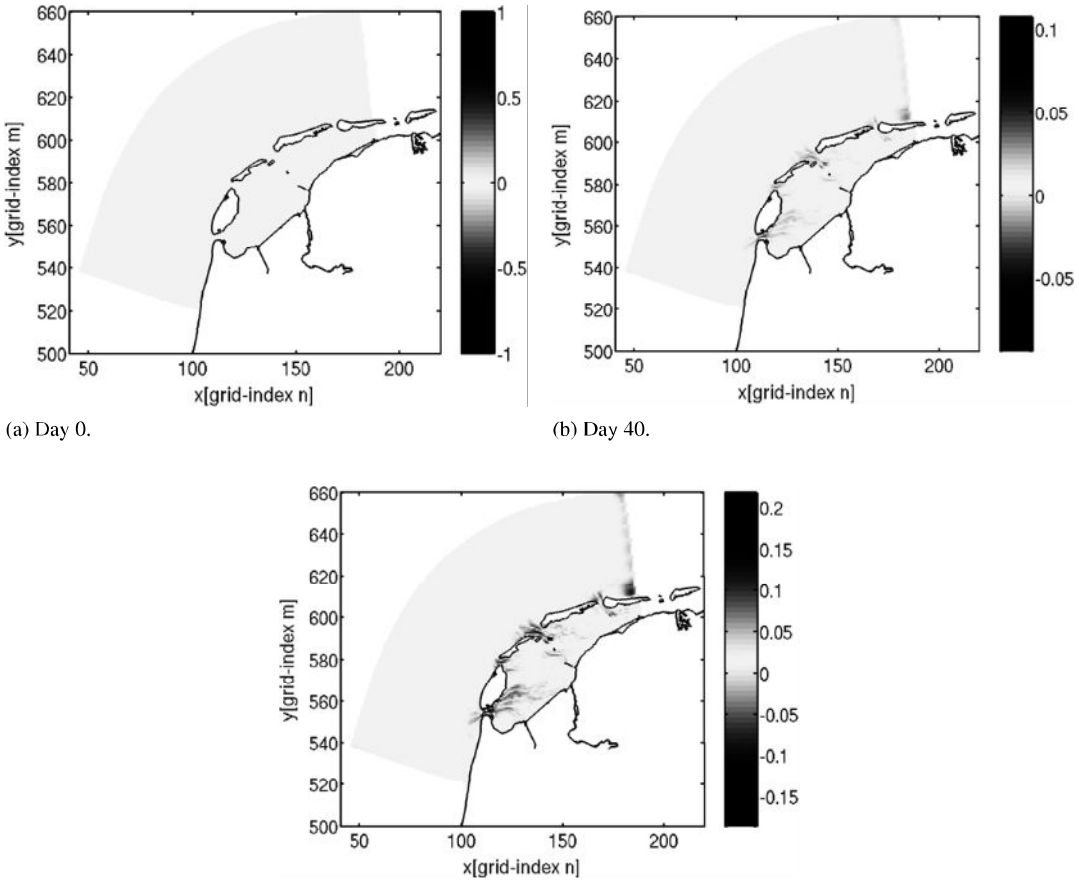



Fig. 6. (a) Parallel speed ups obtained by the three sediment suspension methods (b) Efficiency of the three methods.

requires each processor to compute the flow and depth for all cells. This constant amount of time is not parallelised, also explaining the growing gap in execution times between method II and III in Table 1. Better (relative) results for method III can be expected when the ratio between the number of particles in the simulation and the number of grid cells grows.

It is known that for morphodynamic processes, water flows play an important role which cause erosion and deposition of sediments resulting into bathymetrical changes. The simulation time scales can be categorised as long simulation time scale known as morphodynamic time scale, varies in (weeks, months or years. Another scale, known as a tidal time, varies in minutes, hours (see de Jong [2]), for example. In this paper for practical purposes we use only data for 90 days which are stored on the disk. The experiment for bed level changes $\Delta h$ in metres in Fig. 7(a)–(c) was carried out using the same simulation parameters as those in the Table A.1, except that the mass of each particle $\mathcal{M}_p$ is now set to 10,000 kg, $\Delta t = 864$ s, the probability of sedimentation $= 0.1062$, and the number of iterations $= 8999$ for 90 days.




(a) Day 0.

(b) Day 40.

(c) Day 80.

Fig. 7. Sedimentation in the Dutch North sea over 80 days. The effect is most pronounced in the regions between the islands.

## 6. Discussion and conclusions

In this paper we developed a parallel Lagrangian particle model and applied it to the real life. We have implemented three different suspension approaches in which the technique of globally synchronising random number generators has been introduced in method III.

Computation experiments have been carried out and the parallel performances of the three methods have been measured. The results look promising with good speed up for all three approaches. Method III has a relatively good load balance while methods I and II, have relative a higher speed ups than Method III. The reason for this behaviour has been explained in detail in Section 5.1 It can be explained by looking at the typical number of sediment particles in the flow and comparing it to the number of grid cells. For suspension within a grid cell we need to compute the flow information at the centre of that cell and determine how many particles will be suspended. Computing the flow and depth information takes a significant amount of time and requires access to the large arrays storing domain information. Method III requires each processor to compute the flow and depth for all cells so as to determine the expected number of particles to be suspended for all valid cells. While method II selects only some grid cells and assign them to the available processors. This is the point where method II excels, because each processor needs to compute this information only for the fraction $1/np$ of all cells. Moreover, when you look at the balance of load distribution, the method III clearly excels the two other approaches see Fig. 5(a)–(b). Suspension process using method III leads to perfect load balance. However, the gain in method III by better load balance is offset by the more expensive suspension costs made in all grid cells. Nevertheless, the good speed up obtained in all three methods arises from the fact that unlike in the domain decomposition, communication is greatly reduced in the particle decomposition approach. Furthermore, the use of cache effects has played an important role in speeding up potential. From the results of the studies we recommend the use of method II because it is the most efficient approach in the sediment transport model. The two way coupling between the flow model and particle model for sediment transport in shallow water using a fully domain decomposition approach, is the next step of this research.

## Acknowledgments

The authors would like to thank the Advanced School for Computing and Images(ASCI) for providing us with the computing facilities. Furthermore, we thank the Dutch National Institute for Coastal and Marine Management of Rijkswaterstaat(RIKZ) for supplying us with data.

Table A.1
Simulation parameters in the sediment transport model

| Constant | Unit | Value | Constant | Unit | Value |
|---|---|---|---|---|---|
| Iteration | – | 1999 | $\Delta t$ | s | 86.4 |
| Porosity | – | 0.40 | $\gamma$ | $s^{-1}$ | 0.00013 |
| Grid offset | m | (40550, 500000) | $\lambda_s$ | kg s $m^{-4}$ | 0.0001 |
| Grid size | – | 201 × 225 | $D$ | $m^2$ $s^{-1}$ | 3 |
| Cell size | m | 800 m × 800 m | $\delta$ | – | 1.7 |
| Sand density | kg $m^{-3}$ | 2650 | $\mathcal{M}_p$ | kg | 500 |
| Initial location | m | (135000, 570000) | $f = \frac{\lambda_s}{\gamma}$ | kg $s^2$ $m^{-4}$ | $\mathcal{O}(10^{-4})$ |
| Sedimentation prob. | – | 0.0112 | | | |

## Appendix. Summary of the simulation parameters

See Table A.1.

## References


[1] L. Arnold, Stochastic Differential Equations: Theory and Applications, Wiley, London, 1974.
[2] K. de Jong, Tidally averaged transport models, Ph.D. Thesis, Delft University of Technology, The Netherlands, 1998.
[3] I.T. Foster, Designing and Building Parallel Programs: Concept and Tools for Parallel Software Engineering, Addison-Wesley Publishing Company, New York, 1995.
[4] J. Fredsøe, R. Deigaard, Mechanics of Coastal Sediment Transport, World Scientific, Singapore, 1992.
[5] P. Groen, On the residual transport of suspended load matter by an alternative tidal current, Neth. J. Sea Res. 3 (1977) 564–674.
[6] A.H. Jazwinski, Stochastic Processes and Filtering Theory, Academic Press, New York, 1970.
[7] P.E. Kloeden, E. Platen, H. Schurz, Numerical Solutions of SDE through Computer Experiments, Springer, New York, 2003.
[8] H.X. Lin, A.W. Heemink, J.W. Stijnen, Parallel simulation of the transport phenomena with the particle model Simpar, in: B.H. Li (Ed.), Proc. of the 4th Inter. Confer. on System Simulation and Scientific Computing, Academic Press, Beijing, 1999, pp. 17–22.
[9] H.X. Lin, H.H. ten Cate, L. Dekker, A.W. Heemink, M.R.T. Roest, E.A.H. Vollebregt, Th.L. van Stijn, J.B. Berlamont, Parallel simulation of 3-d flow and transport models within the nowesp project, Int. J. Federation of the European Simulation Societies: Simulation Practice Theory 3 (1995) 257–271.
[10] H.X. Lin, H.H. ten Cate, M.R.T. Roest, E.A.H. Vollebregt, Technical Documentation Online Coupling of TRIWAQ and SIMPAR. Contract simona sys/parkop01, code rkz-336, Project Report to National Institute for Coastal and Marine Management (RIKZ), 1997.
[11] A. Matheja, Sediment transport using the random walk method on top of cobra, in: Proc. of the 4th of Inter. Confer. on Hydroinformatics, Iowa City, USA, 2000, pp. 1–7.
[12] M.J. Quinn, Parallel Computing: Theory and Practice, McGraw-Hill, 1994.
[13] D.R. Robert, R.P. Souto, H.F. de Campos Velho, G.A. Degrazia, D. Anfossi, Parallel implementation of lagrangian stochastic model for pollutant dispersion, Internat. J. Parallel Programming 33 (5) (2005) 485–498.
[14] M.R.T. Roest, Partitioning for parallel finite difference computations coastal water simulation, Ph.D. Thesis, Delft University of Technology, The Netherlands, 1997.
[15] H.M. Schuttelaars, H.E. de Swart, Multiple morphodynamic equilibria in tidal embayments, J. Geophys. Res. 105 (c10) (2000) 14105–14118.
[16] G.S. Stelling, Communications on construction of computational methods for shallow water flow problems, Ph.D. Thesis, Delft University of Technology, The Netherlands, 1983.
[17] The distributed ASCI supercomputer home page. http://www.cs.vu.nl/das2, 2005.
[18] H.F.P. van den Boogaard, M.J.J. Hoogkamer, A.W. Heemink, Parameter identification in particle models, Stochast. Hydrology Hydraulics 7 (2) (1993) 109–130.
[19] L.C. van Rijn, Principles of Sediment Transport in Rivers, Estuaries and Coastal Seas, Aqua Publications, Amsterdam, 1993.
[20] E.A.H. Vollebregt, Parallel software development techniques for shallow water models, Ph.D. Thesis, Delft University of Technology, The Netherlands, 1997.




**W.M. Charles** was born on 20th October 1969 at Musoma in Tanzania. Received his B.Sc(educ) from University of Dar-Es-salaam (Tanzania in 1999) and M.Sc. degree in Mathematics from University of Harare (Zimbabwe in 2001). He got his Ph.D in Mathematics from Delft University of Technology (Netherlands in March 2007). He is now a Lecturer at the University of Dar-Es-salaam in the department of Mathematics in Tanzania. His areas of interest include Stochastic differential equations and their application in modelling such as finance options and stock prices; and transport modelling problems in shallow water; High-performance computing.



**E. van den Berg** received his B.Sc. and M.Sc. degrees in Computer Science from Delft University of Technology (Netherlands). He is now a Ph.D. candidate at the University of British Columbia (Canada). His areas of interest include Parallel and Distributed Computing, Optimization and Geometry.



**H.X. Lin** Associate professor at the Department of Applied Mathematics of the Delft University of Technology (Netherlands). His Research interests are Sparse Matrix Solvers (iterative and direct methods); Domain decomposition and graph partitioning, Monte Carlo simulation and Variance reduction techniques, Scalability analysis of parallel algorithms, Scheduling techniques, Communication systems, Virtual shared Memory systems.



**A.W. Heemink** got his Ph.D. in 1992 at The University of Twente in the Netherlands. He is a Professor at the Department of Applied Mathematics at Delft University of Technology. His areas of interest include Stochastic differential equations; Data assimilation; Inverse modelling; High-performance computing; Applications in computational fluid dynamics.



**M. Verlaan** was born in the Netherlands on December 22. In 1988 he graduated from high school. After which he studied Applied Mathematics at the University of Twente. In 1993 he graduated Cum Laude with the Systems and control group which is headed by Prof. Kwakernaak. The research for his Masters' thesis was carried out at the Technical University of Lulea in northern Sweden. The topic of this work was that an algorithm for the classification of bacteria.

From September 3, 1993 to June 30 1997 he was a Ph.D. Student at the applied mathematics department of Delft University of Technology. His supervisor at the research group for Analysis of Large Scale Systems was Prof. Heemink. The research was carried out in close cooperation with the National Institute for Coastal and Marine Management (RIKZ) which is part of Rijkswaterstaat.

Since July 1, 1997 he has been working as a project-leader with the Hydrodynamics and Data-assimilation research group of RIKZ. In addition he has a part-time research position at the mathematics department of Delft University of Technology.