Parallel implementation of a Lagrangian-based model on an adaptive mesh in C++: Application to sea-ice

Abdoulaye Samaké, Pierre Rampal, Sylvain Bouillon, Einar Ólason

Journal of Computational Physics 350 (2017) 84–96

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Parallel implementation of a Lagrangian-based model on an adaptive mesh in C++: Application to sea-ice

Abdoulaye Samaké *, Pierre Rampal, Sylvain Bouillon, Einar Ólason

Nansen Environmental and Remote Sensing Center, Thormøhlens gate 47, 5006 Bergen, Norway

ARTICLE INFO

Article history:
Received 15 August 2017
Accepted 25 August 2017
Available online 1 September 2017

Keywords:
Parallel computing
Finite-element
Lagrangian advection
Sea-ice

ABSTRACT

We present a parallel implementation framework for a new dynamic/thermodynamic sea-ice model, called `neXtSIM`, based on the Elasto–Brittle rheology and using an adaptive mesh. The spatial discretisation of the model is done using the finite-element method. The temporal discretisation is semi-implicit and the advection is achieved using either a pure Lagrangian scheme or an Arbitrary Lagrangian Eulerian scheme (ALE). The parallel implementation presented here focuses on the distributed-memory approach using the message-passing library MPI. The efficiency and the scalability of the parallel algorithms are illustrated by the numerical experiments performed using up to 500 processor cores of a cluster computing system. The performance obtained by the proposed parallel implementation of the neXtSIM code is shown being sufficient to perform simulations for state-of-the-art sea ice forecasting and geophysical process studies over geographical domain of several millions squared kilometers like the Arctic region.

1. Introduction

Sea ice is a unique material in the geophysical context. It can be viewed as an extremely thin solid, relative to its extent, which clearly displays the behaviour of a solid body through faulting and fracturing, but also moves and flows on the scales of the underlying ocean gyres and the pressure systems of the atmosphere. Its most closely related counterpart is probably the earth's crust, but the Arctic sea-ice cover is composed of floes many orders of magnitude more numerous than the crustal plates and the amount of deformation a single ice floe undergoes in a winter is comparable to the amount of deformation a crustal plate undergoes for the entire lifetime of the planet. As such, modelling sea-ice ideally requires expertise and knowledge of both solid and fluid dynamics, as well as an understanding of fracture mechanics. The mixture of solid and fluid behaviour displayed by the ice makes modelling it demanding, both from a physical and a numerical perspective.

Traditional sea-ice models trace their origin to the seminal paper of [1], which introduces the viscous-plastic model for sea-ice dynamics. Virtually all sea-ice models used today by the climate and ocean modelling communities are based on the viscous-plastic model. The original model, as well as many of its derivatives, was tightly coupled to the underlying ocean model, running on the same Eulerian grid and using the same time step as the ocean model. The momentum equation was solved using a finite difference scheme with an implicit solver (SOR) acting on a linearised version of the original non-linear equation. Advection was performed using a simple up-wind scheme. Later incarnations of the viscous-plastic model have

* Corresponding author.
E-mail addresses: abdoulaye.samake@nersc.no (A. Samaké), pierre.rampal@nersc.no (P. Rampal), sylvain.bouillon@nersc.no (S. Bouillon), einar.olason@nersc.no (E. Ólason).

http://dx.doi.org/10.1016/j.jcp.2017.08.055
0021-9991/

used an explicit solver with an additional relaxation term [2], improved implicit solvers [3], higher order advection schemes [4], and model set-ups where the sea-ice model is separated from the ocean model. Current uses of the viscous-plastic model range from uses in paleo-climate studies, run at a grid resolution down to about 100 km with runtime up to thousands of years, climate simulations and predictions at about 50 km resolution and run for up to 100 years, basin scale Arctic and Antarctic wide simulations at down to about 10 km resolution and run for up to tens of years, and regional downscaling and short term forecasting simulations at down to a few kilometers resolution and run for up to a year.

The `neXtSIM` model breaks from the traditional viscous-plastic approach, and it was developed to model the ice as a solid, as opposed to the more fluid mechanical approach taken in the viscous-plastic model. The internal ice strength is modelled in an elasto-brittle manner, giving rise to cracking via sharp discontinuities in the velocity field [5]. In order to preserve these discontinuities the model uses a pure Lagrangian advection scheme with a moving triangular mesh. `NeXtSIM` is unique in the sea-ice modelling community in that it is the only model to use a moving mesh and the only large scale model using the elasto-brittle rheology [6]. `NeXtSIM` has also been shown to reproduce well the observed statistics of sea-ice deformation, and is the only model so far demonstrated to reproduce the multi-fractal scaling of deformation [6]. Initial versions of the model focused on short time scales [5,6] where mesh distortions could be ignored. A subsequent version that includes a remeshing mechanism has provided a full year simulation, covering the Arctic basin at a 10 km resolution, requiring 2.5×10^5 finite element nodes [7]. In order to run the model at higher resolution, i.e. like 5 km as in some of the state-of-the-art sea ice–ocean platforms that produce operational forecasts nowadays (e.g. TOPAZ [8], ACNFS [9], GIOPS [10]), or for time spans of several decades as if coupled into the future generation of climate models, it is necessary to parallelise it and run it on a potentially large number of processor nodes.

The main challenge in parallelising a model running on a moving and adaptive mesh resides in the choice of a strategy for domain decomposition, as well as on how to treat the communications between the domains when their boundaries are moving. Also, the set-up of these communications must ensure to have no impact on the sea ice properties generated by the physical model, like e.g. the sharp gradients in the modelled sea ice fields that are associated to the simulation of cracks, leads and ridges in the ice cover. In this paper, we present a parallel implementation framework that tackles this challenge, and apply it to the sea ice model `neXtSIM`. In section 2, we briefly describe the equations of the model. In section 3, we detail the spatial and temporal discretisation of the equations. Section 4 is focusing on the parallel implementation of the model, especially the parallelisation strategies we chose for the mesh processing, finite-element analysis and mesh adaptation. Section 5 presents a scalability analysis of the parallel code of `neXtSIM` that uses the approach described in section 4 in a configuration using a 5 km-resolution mesh. Our main conclusions are recapped in section 6.

2. Model description

The `neXtSIM` model has been firstly introduced in a simplified form, mainly focused on the non-linear dynamical core, in [6]. A more advanced and comprehensive formulation of the model, including the thermodynamical components, was presented later in [7]. The sea ice variables, which can be scalar, vectorial or tensorial, defined in the model are listed in Table 2.1.

The evolution equation for sea ice velocity comes from vertically integrating the sea ice momentum equation as follows:

$$\rho_i h \frac{D\boldsymbol{u}}{Dt} = \nabla \cdot \left(h\boldsymbol{\sigma}\right) - \nabla P + A\left(\boldsymbol{\tau}_a + \boldsymbol{\tau}_w\right) - \rho_i h\left(f\boldsymbol{k} \times \boldsymbol{u} + g\nabla\eta\right) \tag{2.1}$$

where $\frac{D\phi}{Dt}$ is the material derivative that is defined for any scalar and vector as

$$\frac{D\phi}{Dt} = \frac{\partial\phi}{\partial t} + (\boldsymbol{u} \cdot \nabla)\phi. \tag{2.2}$$

The parameter ρ_i is the ice density, $\boldsymbol{\tau}_a$ and $\boldsymbol{\tau}_w$ are the surface wind (air) and ocean (water) stresses, respectively. These stresses are defined in equations (2.3a) and (2.3b). The parameter f is the Coriolis frequency, $\boldsymbol{k}$ the upward pointing unit vector, g the gravity acceleration and η the ocean surface elevation. The function P, pressure term parameterized in equation (2.4), is a vertically integrated sea ice pressure term that provides a resistance to the compaction.

$$\boldsymbol{\tau}_a = \rho_a c_a |\boldsymbol{u}_a| \left(\boldsymbol{u}_a \cos\theta_a + \boldsymbol{k} \times \boldsymbol{u}_a \sin\theta_a\right) \tag{2.3a}$$

$$\boldsymbol{\tau}_w = \rho_w c_w |\boldsymbol{u}_w - \boldsymbol{u}| \left(\left(\boldsymbol{u}_w - \boldsymbol{u}\right)\cos\theta_w + \boldsymbol{k} \times \left(\boldsymbol{u}_w - \boldsymbol{u}\right)\sin\theta_w\right) \tag{2.3b}$$

In the equation (2.3a), the field $\boldsymbol{u}_a$ denotes the air velocity. The parameters ρ_a, c_a and θ_a are the air density, the air drag coefficient and the air turning angle, respectively. The field $\boldsymbol{u}_w$ used in the equation (2.3b) denotes the ocean velocity and the parameters ρ_w, c_w and θ_w are the reference density of seawater, the water drag coefficient and the water turning angle, respectively.

The pressure P is parameterized as follows:

$$P = \max\left\{0, -\frac{P^* h^2 e^{-\alpha(1-A)}\nabla \cdot \boldsymbol{u}}{|\nabla \cdot \boldsymbol{u}| + \epsilon_{\min}}\right\}, \tag{2.4}$$

Table 2.1
Sea ice variables used in neXtSIM.

Symbol	Name	Description	Unit
h	thickness	volume of ice per unit area	m
A	concentration	surface of ice per unit area	–
d	damage	from 0 (undamaged) to 1 (completely damaged)	–
$\boldsymbol{u}$	velocity	horizontal sea ice velocity vector	ms^{-1}
$\boldsymbol{\sigma}$	internal stress	planar internal stress tensor	Pa

where P^* is the pressure parameter and α the compactness parameter. The parameter $\epsilon_{\min}$ is set to a small value to regularise the transition when the divergence rate is close to 0.

The evolution equation for the internal stress combines a term coming from Hooke's law (planar stress and linear elasticity) and a sink term S_σ as follows:

$$\frac{\mathcal{D}\boldsymbol{\sigma}}{\mathcal{D}t} = E(A,d)\boldsymbol{K} : \dot{\boldsymbol{\epsilon}}(\boldsymbol{u}) + S_\sigma \tag{2.5}$$

where $E(A,d)$ is the effective elastic modulus, $\dot{\boldsymbol{\epsilon}}$ is the deformation rate tensor defined as

$$\dot{\boldsymbol{\epsilon}}(\boldsymbol{u}) = \frac{1}{2}\left(\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^T\right) \tag{2.6}$$

and $\boldsymbol{K}$ is a stiffness tensor defined such that

$$\begin{bmatrix} (\boldsymbol{K}:\boldsymbol{\epsilon})_{11} \\ (\boldsymbol{K}:\boldsymbol{\epsilon})_{22} \\ (\boldsymbol{K}:\boldsymbol{\epsilon})_{12} \end{bmatrix} = \frac{1}{1-\nu^2}\begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \dfrac{1-\nu}{2} \end{pmatrix}\begin{bmatrix} \epsilon_{11} \\ \epsilon_{22} \\ 2\epsilon_{12} \end{bmatrix} \tag{2.7}$$

for any symmetric tensor ϵ_{ij} $(i,j=1,2)$. The effective elastic stiffness $E(A,d)$ is defined as:

$$E(A,d) = (1-d)Yf(A), \qquad f(A) = e^{\alpha(1-A)} \tag{2.8}$$

where Y is the sea ice elastic modulus (Young's modulus) and α is the compactness parameter. The sink term S_σ represents the damaging process and is here defined so that the internal stress is decreased when the ice is damaged as in [7]. In the case of large drift and deformation, the material derivative of the internal stress tensor is defined as

$$\frac{\mathcal{D}\boldsymbol{\sigma}}{\mathcal{D}t} = \frac{\partial\boldsymbol{\sigma}}{\partial t} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{\sigma} + \beta_a(\nabla\boldsymbol{u},\boldsymbol{\sigma}) \tag{2.9}$$

where β_a is a non-linear term that represents the effect of rotation and deformation when advecting a tensor.

The evolution equations for h, A and d are given in the following form:

$$\frac{D\phi}{Dt} = -\phi(\nabla\cdot\boldsymbol{u}) + S_\phi \tag{2.10}$$

where $\phi \in \{h, A, d\}$ and S_ϕ is a source/sink term. In this paper, thermodynamics are disabled. As a consequence, S_h is set to 0 and S_A is only active to ensure that $A \leq 1$. S_d is composed of two parts, a constant healing term $-\frac{1}{t_d}$ and a damaging term that is only active and positive when the internal stress is out of the Mohr–Coulomb failure envelope (see [7]). The damage values are bounded between 0 (undamaged) and 1 (fully damage).

3. Model discretization

The choice of spatial and temporal discretization are dictated by the characteristics of large scale sea ice dynamics, which are dominated by intermittent and localised deformation events, similar to those caused by stick-slip motion between faults in Earth crust dynamics. These dynamics imply that sea ice deformation is scale dependent over a large range of temporal scales (from hours to months) and spatial scales (from hundreds of meters to hundreds of kilometres). This scale dependence follows a multi-fractal behaviour, where the moments of the distribution scale as power-law functions of the spatial and temporal scales with exponents increasing as a quadratic function of the moment order. In other words, multifractality means that the extreme values of deformation rate are more localised in time and space than the lower values. Reproducing these high values of deformation as observed from space and their impacts on the sea ice state (mainly on the thickness and concentration) is crucial for many applications and is one of the main motivation for building the neXtSIM model.

The neXtSIM model is designed to run at resolutions that are within the range for which multi-fractal scaling is observed. At these resolutions, sea ice motion is discontinuous in space and its gradient (i.e., the deformation or strain rate) then localises at the scale of the model element. The rheology implemented in neXtSIM allows the strain rate to localise

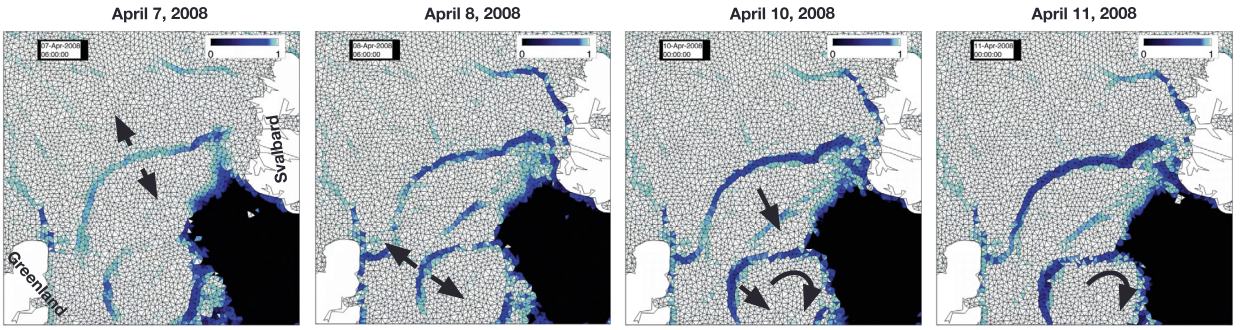


Fig. 3.1. Example of sea ice concentration fields at Fram Strait and the underlying moving mesh coming from a one-year simulation using neXtSIM in a full Arctic configuration and with a resolution of about 10 km. The maps show how the localized divergence generate discontinuities in the ice cover at the scale of the mesh resolution and how these discontinuities are preserved over time by the Lagrangian advection scheme.

at the scale of the model element and to vary very rapidly in time. Having such a model that explicitly reproduces localisation implies that the simulated thickness, concentration and damage fields will also exhibit one cell wide localisation (see an example for the concentration in Fig. 3.1) and rapid variations in time. These characteristics need to be preserved by the discretization and this requires specific choices for the spatial and temporal discretization as well as for the advection scheme as described here after.

3.1. Spatial discretization

`neXtSIM` is discretized on a triangular mesh having a uniform resolution. The velocity field is defined as a piecewise linear function with nodal values defined at the element vertices. The velocity gradient (i.e., the deformation rate) is a constant within each mesh element. The sea ice thickness, concentration and damage are defined as constants over each mesh element. The internal stress tensor is also constant per element as its evolution is a function of the deformation rate.

To limit numerical diffusion that often arises with Eulerian advection schemes, a Lagrangian advection scheme has been implemented [7], where the position of the nodes themselves are a variable of the model. The nodes positions evolve as

$$\frac{D\boldsymbol{X}}{Dt} = \boldsymbol{u}_m \tag{3.1}$$

where $\boldsymbol{u}_m$ is the mesh nodes velocity. When the mesh nodes velocity is defined as $\boldsymbol{u}_m = \boldsymbol{u}$, the mesh moves with the same velocity as sea ice and the advection scheme is purely Lagrangian. A remeshing procedure is then required as described in section 4.2.3.

In this paper we present a modification of the advection scheme that allows to use either a purely Lagrangian, an Eulerian or an Arbitrary Lagrangian–Eulerian (ALE) advection scheme. The choice of advection scheme is simply dictated by the definition of the mesh nodes velocity $\boldsymbol{u}_m$. For the Eulerian scheme, $\boldsymbol{u}_m$ is set to zero and the mesh does not move, meaning that no remeshing is required. In the case of ALE advection, the mesh velocity is defined as a smoothed version of the simulated sea ice velocity, so that the frequency of remeshing is reduced. The smoothing is applied by an iterative process where for each node, except the boundary nodes, the value of $\boldsymbol{u}_m^i$ is updated to the mean value of $\boldsymbol{u}_m^{i-1}$ over its directly connected nodes. The iterative process is initiated by setting $\boldsymbol{u}_m^0$ to $\boldsymbol{u}$. We find that using 2 steps (until $i = 2$) typically reduces the remeshing frequency by a factor 10, which is enough for our application. The results with the ALE scheme discussed in this study are all obtained with 2 smoothing steps.

For the scalar quantities $\phi \in \{h, A, d\}$, the advection scheme is derived as in the finite volume approach by integrating ϕ over each model element and taking the time derivative of this local quantity (e.g., when ϕ is h, the local quantity is the volume of ice contained in each element). The discrete formulation is obtained by applying then the Reynolds transport theorem, introducing in equation (2.10) and regrouping some terms, as follows

$$\begin{aligned}
\frac{d}{dt}\int_{S(t)} \phi dS &= \int_{S(t)} \frac{\delta\phi}{\delta t} dS + \int_{\delta S(t)} \phi\,(\boldsymbol{u_m}\cdot\boldsymbol{n})\,dL \\
&= \int_{S(t)} \Big(-\nabla\cdot(\phi\boldsymbol{u}) + S_\phi\Big) dS + \int_{\delta S(t)} \phi\,(\boldsymbol{u_m}\cdot\boldsymbol{n})\,dL \\
&= \int_{S(t)} S_\phi dS + \int_{\delta S(t)} \phi\Big((\boldsymbol{u_m}-\boldsymbol{u})\cdot\boldsymbol{n}\Big) dL
\end{aligned} \tag{3.2}$$

where $S(t)$ and $\delta S(t)$ are the surface and boundary of each triangle of the mesh, and $\boldsymbol{n}$ is the normal to $\delta S(t)$. As ϕ and S_ϕ are constant per element, the discrete form of equation (3.2) is

$$\frac{d}{dt}S_e(t)\phi_e = S_e(t)S_{\phi e} + \sum_{j=1:3} F(\phi)_{ej}L_{ej} \tag{3.3}$$

where e is the index of the element, S_e is its surface that may evolve in time, j is the local index of the element edges, L_{ej} is the length of the edges and $F(\phi)_{ej}$ is the flux through the edge of index j.

To ensure conservation, the flux $F(\phi)_{ej}$ is, by construction, the same on both sides of an edge. We here use an upwind scheme where

$$F(\phi)_{ej} = \phi_{ej\text{up}}\ (\boldsymbol{u_m} - \boldsymbol{u})|_c \cdot \boldsymbol{n}_{ej} \tag{3.4}$$

and $\phi_{ej\text{up}}$ is the upwind value of the advected field. The velocity $(\boldsymbol{u_m} - \boldsymbol{u})|_c$ is evaluated at the middle of the element edges. When used with an explicit scheme, the upwind scheme has the advantage of being purely local, with no linear system to solve (see section 3.2).

This formulation of the advection scheme is also used for the open boundaries, but with a slight modification to avoid the model domain to change during a simulation. On these boundaries, the computed velocity $\boldsymbol{u}$ may not be equal to zero, meaning that the model domain would change with the ALE and Lagrangian schemes. To avoid that, we set $\boldsymbol{u}_m$ to zero on the open boundary nodes (i.e., the nodes connected to two open boundary edges). The fluxes are then computed as for any other edge by equation (3.4). The only difference with the other edges is the definition of the upwind value of the advected fields when the fluxes are incoming. One could either impose a value coming from another model run or from observations, or replace the upwind value by the downwind value. For the study, we chose the second option to do not have to set up a value on the open boundaries. This treatment of open boundary could become problematic in a domain with large open boundaries where one can expect considerable influx through the boundary. In the current setup, however, the only open boundary where any influx can be expected is the Bering Strait, and there only limited influx for a limited time is to be expected.

For the sea ice velocity, strict conservation is usually not required as the advection of momentum is negligible compared to the other terms of equation (2.1) [11]. In this implementation, the material derivative of the ice velocity is computed as

$$\frac{D\boldsymbol{u}}{Dt} = \left.\frac{\partial \boldsymbol{u}}{\partial t}\right|_{\boldsymbol{X}(t)} \tag{3.5}$$

where $\boldsymbol{X}(t)$ the position of the nodes evolves as in equation (3.1). In the purely Eulerian case, the advection of momentum is then completely neglected, whereas it is fully preserved in the purely Lagrangian case. The equation (2.1) is then discretized by using the finite-element methods as described in section 3.3.

For the internal stress, the effect of advection is also minor compared to the other terms of equation (2.9). In this implementation, the material derivative for the internal stress is computed as

$$\frac{\mathcal{D}\boldsymbol{\sigma}}{\mathcal{D}t} = \left.\frac{\partial \boldsymbol{\sigma}}{\partial t}\right|_{\boldsymbol{X}(t)} \tag{3.6}$$

where the effect of rotation and deformation are neglected. The effect of constant advection is only fully represented for the purely Lagrangian case.

3.2. Temporal discretization

At the temporal scale of interest (about 1 day), the evolution of sea ice velocity, internal stress and damage may be discontinuous and then requires a specific temporal discretization. A classical approach in progressive damage modelling is to involve sub-iterations to allow discontinuities to develop during one model time step. In [6], we found that an equivalent behaviour can be obtained without sub-iteration by simply using a sufficiently small time step (typically 200 seconds) compared to the time scale of interest (typically 1 day).

In this implementation, the equations for the velocity and internal stress are first solved together with an implicit scheme. Using an explicit scheme would require a much smaller time step that 200 seconds, as the speed of propagation of (shear) elastic waves in sea ice is about 500 m/s. The scalar quantities, such as the concentration, thickness and damage are then updated with an explicit scheme.

The momentum equation (2.1) is discretised in time as follows:

$$\begin{aligned}
\rho_i h^n \frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^n}{\Delta t} =& \nabla \cdot \left(h^n \boldsymbol{\sigma}'\right) \\
&+ A^n \rho_a c_a \left|\boldsymbol{u}_a\right|_e \left(\boldsymbol{u}_a \cos\theta_a + \boldsymbol{k} \times \boldsymbol{u}_a \sin\theta_a\right) \\
&+ A^n \rho_w c_w \left|\boldsymbol{u}_w - \boldsymbol{u}^n\right|_e \left(\boldsymbol{u}_w - \boldsymbol{u}^{n+1}\right) \cos\theta_w \\
&+ A^n \rho_w c_w \left|\boldsymbol{u}_w - \boldsymbol{u}^n\right|_e \left(\boldsymbol{u}_w - \boldsymbol{u}^{n+1}\right) \sin\theta_w \\
&- \rho_i h^n \left(f \boldsymbol{k} \times \boldsymbol{u}^* + g \nabla \eta\right)
\end{aligned} \tag{3.7}$$

where

$$\boldsymbol{\sigma}' = \boldsymbol{\sigma}^n + \Delta t E(A^n, d^n)\boldsymbol{K} : \dot{\boldsymbol{\epsilon}}^{n+1} \tag{3.8}$$

and the operator $|\cdot|_e$ denotes the norm over a mesh element. The sea ice velocity $\boldsymbol{u}^*$ is defined by

$$\boldsymbol{u}^* = \beta_0 \boldsymbol{u}^n + \beta_1 \boldsymbol{u}^{n-1} + \beta_2 \boldsymbol{u}^{n-2}, \tag{3.9}$$

where the parameters β_0, β_1 and β_2 are the coefficients of the third order Adams–Bashfort scheme, given by $\beta_0 = 23/12$, $\beta_1 = -16/12$ and $\beta_2 = 5/12$, respectively. These parameters are chosen for the stability of the scheme, see [12]. Using lower order schemes could be sufficient in the case of sea ice but it has not been investigated in the present study.

Remark 1. The symmetric part of the ocean drag term is treated implicitly, whereas the anti-symmetric part is treated explicitly to preserve the symmetry of the system that we need to solve. The Coriolis term is also treated explicitly.

The internal stress is updated using

$$\boldsymbol{\sigma}^{n+1} = \boldsymbol{\sigma}^n + \Delta t E(A^n, d^n)\boldsymbol{K} : \dot{\boldsymbol{\epsilon}}^{n+1} + S_{\boldsymbol{\sigma}}^{n+1} \tag{3.10}$$

The sea ice thickness, concentration and damage are then updated by using an explicit time integration, leading to the following expression for the update of ϕ:

$$\phi^{n+1} S^{n+1} = \phi^n S^n + \Delta t S_\phi S^n + \Delta t \sum_{j=1:3} F(\phi)_j L_j \tag{3.11}$$

where the monotonicity (positivity for the ice thickness, concentration and damage) is ensured by limiting the outward fluxes through one edge to be maximum 1/3 of the quantity stored in the giving element. The fluxes are then limited to fulfil the equation (3.12).

$$3\left|F(\phi)_j\right| \Delta t L_j \leq (\phi_{up}^n + \Delta t S_\phi) S_{up}^n \tag{3.12}$$

This upwind explicit scheme is conservative, monotonous and local (i.e., no system needs to be solved). However if used with the purely Eulerian approach, this scheme is known to be very diffusive. In this study, we only use this upwind scheme for the ALE method, so that numerical diffusion is only active where the sea ice velocity field has sharp gradients that are smoothed out in the definition of $\boldsymbol{u}_m$. In the purely Lagrangian case, the fluxes are zero by construction but numerical diffusion may still arise during the necessary remeshing steps that are applied when the mesh is highly deformed (see section 4.2.3). No quantitative analysis of the numerical diffusion has been made but visual inspection shows that the ALE scheme and the Lagrangian scheme exhibit similar numerical diffusion that is limited, and only located where large deformation occurs.

3.3. Finite-element method

The finite-element method [13,14] is used for the spatial discretization of the problem. This discretization method is one of the most common techniques used to compute the approximate solution of complex problems, usually expressed in terms of partial differential equations (PDEs) describing physical processes in science and engineering. As this method is rarely used in the case of sea ice and ocean modelling, we here recap its main steps. Readers used to play with the classical finite element method can directly jump to section 4.

The main steps involved in the finite-element analysis are:

(i) the problem definition (ii) the discretization: triangulation and definition of approximation space (iii) the definition of the variational (or weak) formulation (iv) the system assembling: local (on each mesh element) and global (on the entire mesh) (v) the solution of the resulting linear system.

The problem is defined on a bounded domain Ω of $\mathbb{R}^2$. We denote $\partial\Omega$ the boundary of the domain Ω, that is decomposed as $\partial\Omega = \partial\Omega_{\mathrm{D}} \bigcup \partial\Omega_{\mathrm{N}}$ with $\partial\Omega_{\mathrm{D}} \bigcap \partial\Omega_{\mathrm{N}} = \emptyset$, where $\partial\Omega_{\mathrm{D}}$ and $\partial\Omega_{\mathrm{N}}$ are respectively the closed (Dirichlet) and open (Neumann) boundaries.

From the temporal discretization (see equation (3.8)), the resolution of the momentum equation consists in solving the following problem: find $\boldsymbol{u}^{n+1}$ such that

$$\begin{cases} k\boldsymbol{u}^{n+1} + \nabla \cdot \left(h^n \boldsymbol{\sigma}'\right) + \boldsymbol{f} = 0 \ \text{ in } \ \Omega \\ \boldsymbol{u}^{n+1} = 0 \ \text{ on } \ \partial\Omega_{\mathrm{D}} \\ \boldsymbol{n} \cdot \left(h^n \boldsymbol{\sigma}'\right) = 0 \ \text{ on } \ \partial\Omega_{\mathrm{N}} \end{cases} \tag{3.13}$$

where $\boldsymbol{n}$ is the outward-pointing normal on $\partial\Omega_{\mathrm{N}}$, k a scalar function that does not depend on $\boldsymbol{u}^{n+1}$ and $\boldsymbol{f}$ is the vector regrouping all the terms that do not depend on $\boldsymbol{u}^{n+1}$ nor $\boldsymbol{\sigma}'$.

Let $\mathcal{T}_\delta$ be a triangulation (or, equivalently mesh) of Ω, where δ refers to the maximum characteristic length of the mesh. We denote $\mathcal{V}_\delta$ the piecewise linear finite-element space defined on $\mathcal{T}_\delta$ with vanishing value at closed boundary $\partial\Omega_D$:

$$\mathcal{V}_\delta = \left\{ v \in C^0(\overline{\Omega}) \,\middle|\, v \text{ is linear on } K,\, K \in \mathcal{T}_\delta \right\} \bigcap H^1_{0,\partial\Omega_D} \tag{3.14}$$

where $H^1_{0,\partial\Omega_D}$ is the Hilbert–Sobolev space of order 1, satisfying an homogeneous Dirichlet condition on the closed boundary $\partial\Omega_D$. $C^0(\overline{\Omega})$ is the set of continuous function defined on $\overline{\Omega}$ (the closure of Ω).

The variational form of the problem (3.13), by using the expression of $\boldsymbol{\sigma}'$ as linear combination of $\boldsymbol{\sigma}^n$ and the deformation rate tensor $\dot{\boldsymbol{\epsilon}}$ introduced in equation (2.9), is written:

find $\boldsymbol{u} \in \mathcal{V}_\delta$ such that

$$\int_\Omega \left(k\boldsymbol{u} + \nabla \cdot \left(h^n \left(\boldsymbol{\sigma}^n + \Delta t E(A^n, d^n) \boldsymbol{K} : \dot{\boldsymbol{\epsilon}}(\boldsymbol{u}) \right) \right) + \boldsymbol{f} \right) \boldsymbol{v} = 0 \qquad \forall\, \boldsymbol{v} \in \mathcal{V}_\delta \tag{3.15}$$

where $\boldsymbol{u}$ refers to $\boldsymbol{u}^{n+1}$.

By integrating by parts the equation (3.15) and taking into account the boundary (closed and open) conditions as defined in the problem (3.13), the following weak form holds:

$$k \int_\Omega \boldsymbol{u}\boldsymbol{v} - h^n \Delta t \int_\Omega \left(\dot{\boldsymbol{\epsilon}}(\boldsymbol{v}) : E(A^n, d^n) \boldsymbol{K} : \dot{\boldsymbol{\epsilon}}(\boldsymbol{u}) \right) = h^n \int_\Omega \left(\nabla \boldsymbol{v} \right) : \boldsymbol{\sigma}^n - \int_\Omega \boldsymbol{f}\boldsymbol{v} \tag{3.16}$$

Equation (3.16) is ill defined where no ice is present, as no sea ice velocity is defined. The classical approach is to set the variable $\boldsymbol{u}$ to zero when no ice or too little ice is present in a cell. Some implementations set the velocity of cells with very low ice content equal to the ocean velocity. We propose another approach where a Laplacian-like equation is solved on the ocean cells (here defined as the cells where sea ice concentration is less than 1%) so that the velocity linearly decreases from the ice edge to the nearest coast. This approach is applied whatever the advection scheme is. In the case of the purely Eulerian approach, it limits potential artificial accumulation of sea ice in the first ocean cells. In the case of the purely Lagrangian scheme, it reduces the problem of large mesh deformation occurring in the first ocean cells and inducing large numerical diffusion due to frequent remeshing. In general though, the treatment of the low concentration cells at the ice edge does not affect the interior of the domain because information is only transmitted between cells when concentration is high (see equations (2.1), (2.4), and (2.8)). The "moving empty-cells" method however does not help where the ice edge is close to the coast, which is always occurring when running on Arctic wide domain. To limit the frequency of mesh adaptation, we found that using the ALE advection in conjunction with the "moving empty cell" works particularly well and this has become the default option when running in parallel. Numerically, the "moving empty cells" method is simply applied by replacing (3.16) over ocean cells, by this equation

$$\Delta t \int_\Omega \left(\dot{\boldsymbol{\epsilon}}(\boldsymbol{v}) : E_0 \boldsymbol{K} : \dot{\boldsymbol{\epsilon}}(\boldsymbol{u}) \right) = 0 \tag{3.17}$$

where E_0 is set to a much smaller value than the Young modulus, so that this additional term does not impact the calculation of $\boldsymbol{u}$ over non-empty cells.

The weak form (3.16) can also be written:

$$Q(\boldsymbol{u}, \boldsymbol{v}) = R(\boldsymbol{v}). \tag{3.18}$$

Let $\mathcal{B} = \left\{ \phi_i \right\}_{i=1}^{N_\delta}$ be a set of basis functions of V_δ. The approximate solution $\boldsymbol{u} \in V_\delta$ can be expressed as

$$\boldsymbol{u} = \sum_{i=1}^{N_\delta} \boldsymbol{u}_i \phi_i \tag{3.19}$$

where $\boldsymbol{u}_i, (i = 1, \ldots, N_\delta)$ are nodal values. By using the equations (3.19) and (3.18), the following discrete form of the problem holds:

$$\sum_{i=1}^{N_\delta} \boldsymbol{u}_i Q(\phi_i, \phi_j) = R(\phi_j), \quad \forall j = 1, \ldots, N_\delta. \tag{3.20}$$

4. Parallel implementation

4.1. Computational issues and challenges

The main objective with the development of neXtSIM is to reproduce the mechanical behaviour and state of the Arctic sea ice cover. This task requires the model to have grid resolutions of about a few kilometers. With a computational domain of about 14 million km^2, resolving the entire Arctic at such resolutions requires around one million mesh elements. Solving a problem of such size in a timely manner requires the use of parallel computing [15,16]. In practice this requires at least a distributed memory parallelisation, where the model domain is partitioned and the equations are solved on the computational subdomains in parallel.

The key difficulty in the parallel implementation of neXtSIM lies in the fact that since the model uses a Lagrangian advection scheme, the mesh nodes move with the sea ice velocity $\boldsymbol{u}$ (see equation (3.1)). The computational mesh therefore deforms as the ice cover itself deforms. If the mesh becomes too deformed the finite element method will fail, requiring an adaptation of the mesh, referred to here as a remeshing. For this first implementation, we used the same library for remeshing as in the serial code presented in [7]. This library called BAMG (Bidimensional Anisotropic Mesh Generator, [17]) does not have parallel capabilities and is then called by the root processor on the whole domain. Once the mesh has been adapted the model fields need to be remapped from the old mesh to the new one. The remeshing process and the remapping that follows are particularly challenging for the parallel implementation, since each remeshing may require a new partitioning that no longer matches with the previous one. Hence the remapping can not be carried out locally in each computational subdomain. The mesh partitioning and the parallel finite element data structure remain unchanged between remeshings, and therefore the MPI communication patterns.

To avoid to constantly update the forcings terms and the spatial operators such as the gradients, we adopt a mixed approach, where the position of the model nodes $X(t)$ are decomposed into two parts, $X(t) = X_m + U_m(t)$, where X_m is the position of the computational mesh that is kept fixed between two remeshing steps, and $U_m(t)$ is the actual displacement of the nodes relative to the computational mesh. The terms that are computationally expensive to evaluate, such as the forcings that need to be interpolated in space and the spatial derivative operators, are computed with the position of the computational mesh. The forcings are then spatially interpolated only after each remeshing, or when new forcing fields are required. The spatial derivative are computed only after each remeshing. We checked that this method gives almost identical results as when we update the computational mesh every time step. Indeed, as the remeshing criterion is global the error in the position of the nodes is in practice never larger than the size of a single model element. Given a relatively high resolution mesh, the forcing fields are generally too smooth and too coarsely resolved for this error to have any substantial effect. These features are particularly interesting for the parallel implementation, and contribute strongly to the relevance and efficiency of the proposed framework. In a coupled system where the ocean or atmosphere would respond to sea ice changes at the scale of the model mesh, one should develop another approach to efficiently handle communication between the moving mesh of the sea ice model and the fixed meshes of the ocean and atmosphere models.

4.2. Parallelization strategy

The main parallel programming paradigms are shared memory (OpenMP), distributed memory with message passing (MPI), and hybrid computing (MPI and OpenMPI). In this framework we consider the distributed-memory programming approach based on MPI [18]. The inter-process communications are handled by using Boost.MPI, a Boost library [19] that is an abstraction layer over the standard MPI for simplifying the usage interface. The parallel code of neXtSIM is based on modern C++ (C++11/14) programming and is structured as follows: (i) the mesh processing (including mesh generation and mesh partitioning) (ii) the parallel finite-element analysis (iii) the mesh adaptation and spatial interpolation.

4.2.1. Mesh processing

The initial computational mesh is generated in pre-processing over a pan-Arctic region by using Gmsh [20], a three-dimensional finite-element mesh generator. This mesh is loaded by the root process (that has the rank 0) and then partitioned using the partitioning library METIS [21], which is called from within Gmsh using the existing Gmsh data structures. The number of mesh partitions corresponds to the size of the MPI communicator. The partitioned mesh is saved on the hard disk and loaded independently by all processor cores, each of which extracts only the partition number corresponding to its rank plus the inter-process, or ghost elements. This way all the elements needed for the finite-element basis functions are locally available, removing the need for communication during the finite-element assembly. The root process keeps the information on the entire mesh in memory, so that it can be used in the next remeshing. This runtime mesh partitioning can be considered as a pre-processing step, which is performed automatically and on-line instead of manually and off-line before the simulation starts.

4.2.2. Parallel finite-element analysis

Let us start by constructing the set of the degrees of freedom (DOFs) required for the finite-element discretization. This step is done locally in each processor core on its local mesh. Each processor has its own numbering of DOFs in the local

mesh, called local numbering which is associated to a global numbering (that requires inter-process communications) on the entire problem. These associative relationships are called the local-to-global maps (LtGMs).

The linear algebra operations are managed using the PETSc [22] wrappers and the parallel sparse matrices are stored in the CSR (Compressed Sparse Row) format. The finite-element assembly is performed locally on each processor core, by computing the matrix entries (corresponding to the local DOFs) and adding them to the global parallel matrix, using the map LtGM. For applying the homogeneous Dirichlet boundary conditions (see equation (3.13)) we modify the matrix after assembly, using the PETSc function `MatZeroRowsColumns`, that sets to zero all entries (except the main diagonal which is set to 1) of the row and column corresponding to each node on the closed boundary $\partial\Omega_D$, and update the vector to keep the system consistent. These operations are carried out locally and preserve the symmetry of the matrix.

4.2.3. Mesh adaptation

When using a Lagrangian advection scheme the computational mesh moves and deforms with the ice cover. The movement of the ice inevitably leads to distortions of the grid that are too large for the finite element method to produce an accurate result. The mesh therefore needs to be periodically adapted to counteract mesh distortions and ensure the quality and numerical integrity of the mesh [23].

The main mesh adaptation techniques discussed in the literature [24,25] are the *h-adaptive*, *p-adaptive* and *r-adaptive* methods. We will consider here the h-adaptive approach, which is more flexible for our context, and adapts the mesh (locally or globally) by adjusting the size of its elements (isotropically or anisotropically) according to a given criterion, such as the mesh characteristic lengths. The mesh connectivity changes dynamically when nodes are added or deleted. In `neXtSIM` the mesh adaptation is performed using a global, h-adaptive based, and anisotropic mesh generator called BAMG (Bidimensional Anisotropic Mesh Generator). This software has many powerful mesh adaptation capabilities and can importantly preserve as many of the nodes from the old mesh as possible. We use a `C++` object-based implementation of this library from the Ice Sheet System Model [26] and inspired to the original approach introduced in [17]. This BAMG implementation provides various interesting features and simplified user interfaces which are useful and adapted to our model.

In `neXtSIM`, we consider the mesh too distorted if the smallest angle of any triangle of the mesh is smaller than a critical angle θ_{min}, the remeshing is triggered when this criterion is reached during the simulation. The new mesh is generated by BAMG according to the constraints based on the minimum and maximum characteristic lengths of the initial mesh, ℓ_{min} and ℓ_{max}, respectively. This means that in the new mesh, the length of any edge of any triangle is bounded by ℓ_{min} and ℓ_{max}. The new mesh satisfies the Delaunay criterion and has the same spatial resolution as the initial mesh.

Algorithm 4.1 Remeshing and interpolation procedures.

Let $t^n = t^0 + \Delta t$ be the current model time
if minimum angle threshold is reached **then**
  `#Gather all fields required on the root process (rank 0)`
  for each process p **do**
    send $\boldsymbol{TU}^{n,(p)}, \boldsymbol{u}^{n,(p)}, \boldsymbol{\sigma}^{n,(p)}, h^{n,(p)}, A^{n,(p)}, d^{n,(p)}$ to root process (#0)
  end for
  synchronize all processes
  on the root process:
    `#Get fields from each process`
    receive $\boldsymbol{TU}^{n,(p)}, \boldsymbol{u}^{n,(p)}, \boldsymbol{\sigma}^{n,(p)}, h^{n,(p)}, A^{n,(p)}, d^{n,(p)}$ from each process p

    `#Mesh adaptation`
    $\boldsymbol{TU}^n \longleftarrow (\boldsymbol{TU}^{n,(0)}, \ldots, \boldsymbol{TU}^{n,(np)})$ `# global displacement vector on root`
    $\boldsymbol{X}^{n+1} \longleftarrow \boldsymbol{X}^n + \boldsymbol{TU}^n$ `# update mesh nodes positions`
    call BAMG `# remeshing process`
    partition the new mesh (in memory) `# using METIS from Gmsh`
    save the new partitioned mesh on the disk `# in Gmsh format`

    `#Spatial interpolation from old mesh to new mesh`
    linear (P_1) interpolation for $\boldsymbol{u}^n$ `# nodal interpolation for velocity`
    P_0 interpolation for $\boldsymbol{\sigma}^n, h^n, A^n, d^n$ `# conservative using cavities`

    `#Dispatch interpolated fields from root to each process`
    scatter $\boldsymbol{u}^{n,(p)}, \boldsymbol{\sigma}^{n,(p)}, h^{n,(p)}, A^{n,(p)}, d^{n,(p)}$ to each process p

  `#Receive fields from root process (rank 0)`
  for each process p **do**
    receive $\boldsymbol{u}^{n,(p)}, \boldsymbol{\sigma}^{n,(p)}, h^{n,(p)}, A^{n,(p)}, d^{n,(p)}$ from root process
  end for
end if

During the mesh adaptation, the mesh is only modified in a limited number of locations, the so-called *cavities*, since we instruct BAMG to preserve as many of the nodes from the old mesh as possible. This allows the model to track large expanses of drifting ice that is deforming very little, without any artificial diffusion, since it is only necessary to interpolate

values from the old mesh to the new one inside the cavities. Outside the cavities the tracer values are not affected by the mesh adaptation (see [7] for more details).

For the variables defined at the nodes (i.e. the sea ice velocities, etc.), a non-conservative linear interpolation is performed for the new mesh nodes. For the quantities that are defined at the centre of the elements, a conservative interpolation scheme is applied to each cavity independently. The cavities are defined as the smallest partitions of the mesh for which the external edges are the same before and after the mesh adaptation.

We implemented a fast algorithm using the information provided by BAMG for efficiently detecting the cavities and the intersections between the mesh elements of the old and new meshes. For each intersection, the corresponding integrated quantities are transferred from the old mesh to the new one. This process is fully conservative.

5. Numerical experiments

The simulations have been performed at the Bergen High Technological Center (HiB) on the supercomputer machine Hexagon, which is managed and operated by the University of Bergen (UiB). Hexagon is a supercomputer with 22272 processor cores on 696 compute nodes interconnected with a high-bandwidth low-latency switch network (Gemini, 2.5D torus). Each node has two 16-core AMD Opteron "Interlagos" processors (2.3 Ghz) and 32GB of memory. The operating system is Cray Linux Environment CLE 5.2 (Based on Novell Linux SLES11sp3) and the file system is Lustre. Hexagon has a peak performance of 204.9 TFlops.

In order to measure the performance of our parallel algorithms, we perform the strong scalability (also called speedup) analysis [27, chapter 1]. This speedup, that allows to evaluate the quality of a parallel algorithm running on a parallel computing platform, is defined as the time for running a problem on one processor core divided by the time taking to run the same problem (fixed size) using p ($p > 1$) processor cores.

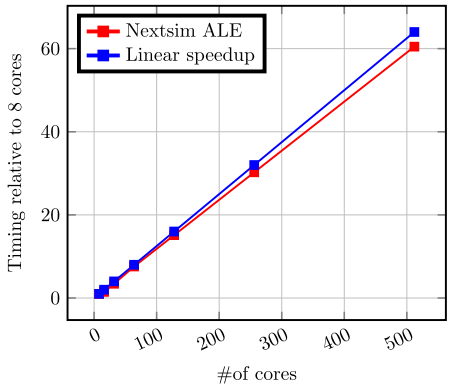
The numerical experiments were carried-out using a computational mesh with the mean spatial resolution in x-direction (Δx) of about 5 km. This mesh is defined on a domain that covers the whole Arctic Ocean and extends to the South until the Gulf of Saint Laurent (see an example in Fig. 5.1). This mesh is comparable in terms of the region covered and spatial resolution to the ones used nowadays for state-of-the-art coupled sea ice–ocean simulations, to either produce operational forecasts or study climate processes in Arctic. The number of mesh elements (triangles) is $\approx 8 \times 10^5$ and the number of nodes is $\approx 4 \times 10^5$. We consider both advection schemes (ALE and pure Lagrangian) described above. The physical time-step (Δt) is set to 100 s and the duration of the simulation is 25 days. The total number of unknowns (the size of the linear system) at each time-step is roughly equal to 8×10^5. The physical parameters used for the simulations are reported in the Table 5.1. The computations are achieved using $2^p (p = 3, \ldots, 9)$ processor cores.

Using the ALE advection (see the Fig. 5.2.1), the speedup relative to 8 cores when employing 512 cores is 60.48 while the linear speedup is 64. This performance represents an efficiency of 94.50%. This result shows that our parallel algorithm scales well until 512 processor cores and demonstrates the good computational properties for the `neXtSIM` model when using the ALE advection scheme.

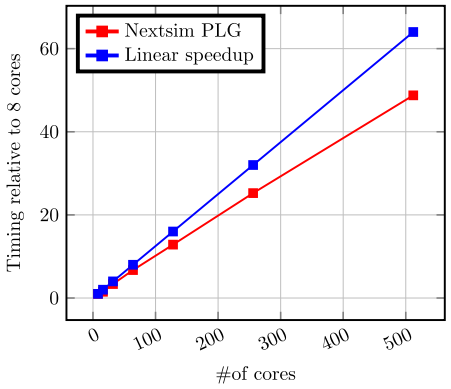
Fig. 5.1. Example of computational mesh at 30 km resolution covering roughly the same domain as used for our numerical experiments. The colours correspond to the 4 subdomains obtained after a mesh partitioning processed by METIS.

Table 5.1
Parameters used in neXtSIM.

Symbol	Name	Value	Unit
ρ_a	air density	1.3	$\mathrm{kg\,m^{-3}}$
c_a	air drag coefficient	7.6×10^{-3}	–
θ_a	air turning angle	0	°
ρ_w	water density	1025.0	$\mathrm{kg\,m^{-3}}$
c_w	water drag coefficient	5.5×10^{-3}	–
θ_w	water turning angle	25.0	degree
ρ_i	ice density	917	$\mathrm{kg\,m^{-3}}$
ν	Poisson coefficient	0.3	–
μ	internal friction coefficient	0.7	–
Y	elastic modulus	9.0	GPa
T_d	damage relaxation time	28.0	days
c	cohesion parameter	8.0	kPa
α	compactness parameter	−20.0	–

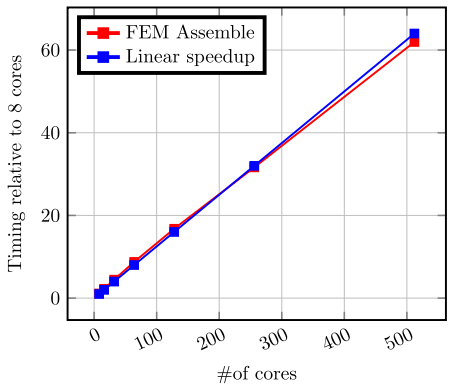


5.2.1. Arbitrary Lagrangian Eulerian

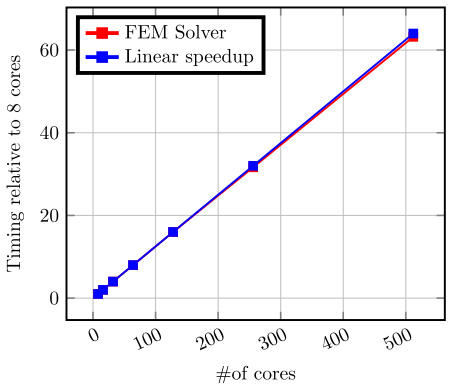


5.2.2. Pure Lagrangian

Fig. 5.2. Strong scalability analysis.



5.3.1. Speedup for assemble



5.3.2. Speedup for solver

Fig. 5.3. Strong scalability analysis: total timer for assemble and solver.

Using the pure Lagrangian advection (see the Fig. 5.2.2), the speedup relative to 8 cores when employing 512 processor cores is 48.76. This speedup is significantly lower than the linear speedup 64 and the corresponding efficiency is 76.18%. The poverty of this result can be explained by the too high frequency of remeshing, which takes place on the root processor, and is consequently not scalable. In addition, the remeshing is followed by a new mesh partitioning and, as already mentioned in previous section, by a local and conservative interpolation procedure in “cavities”, both also performed on the root process.

The scalability results presented above are summarised in the Fig. 5.2.

Usually in a finite element code, the most expensive parts in computing time are assembly and solver processes. In addition to the strong scalability analysis for the entire model presented above, we report the strong scaling (speedup) for the finite element assembly and solver in Fig. 5.3.1 and Fig. 5.3.2, respectively. As we can see, these results show that the

assembly and solver scale well respectively with 86.15% and 98.75% on up to 512 computing cores, compared to the linear speed-up . This is a good indicator that the degradation observed in the strong scaling for the entire model, especially using the pure Lagrangian advection (see the Fig. 5.2.2) is mainly due to the other parts of the code, particularly the remeshing and repartitioning processes.

6. Conclusion

In this paper we have presented a parallel computational framework for the next generation sea-ice model, `neXtSIM`. `NeXtSIM` is a stand-alone dynamic/thermodynamic model based on the elasto-brittle rheology, using a moving adaptive mesh to better preserve the discontinuities produced by this rheology. The moving mesh, while central to the good performance of the model, makes the parallelisation of the model more challenging than otherwise. `NeXtSIM` is the only large-scale sea-ice model using a moving mesh, consequently making this the first parallel implementation of a sea-ice model on a moving mesh.

The framework presented here was shown to scale with acceptable efficiency for the specified requirements. The finite element assembly and solver were shown to scale with 86.15% and 98.75% efficiency on up to 512 cores, compared to the linear speed-up. However, as the remeshing is always done on the root processor, the efficiency of the scaling for the entire model is much lower. For the pure Lagrangian scheme the scaling efficiency is 76%, while using the arbitrary Lagrangian Eulerian (ALE) scheme increases that efficiency to 95%.

These results show that the best way to improve the model scalability when using the pure Lagrangian advection scheme is to parallelise also the remeshing. Parallel remeshing has already been proposed [e.g. by 28], but it remains to be seen to which extent this work is applicable to the current model. The remeshing parallelisation will be in particular needed to go to higher spatial resolutions than presented in this paper with the `neXtSIM` model, e.g. $\lesssim 1$ km. As it stands, the current parallelised model can be run at a reasonable cost for several years at a resolution as high as approximately 5 km over the entire Arctic (4×10^5 mesh nodes). This makes `neXtSIM` capable of handling resolutions similar to, or higher than other state-of-the art sea-ice models, and the model can thus be used for both realistic short-term forecast applications at regional scales and for climate simulations in coupled geophysical systems.

Acknowledgements

This work was supported by the neXtWIM project (Norwegian Research Council grant no 244001). The authors acknowledge the grant nn2993k of the Norwegian supercomputing infrastructure NOTUR and the user support on the system Hexagon.

References

[1] W.D. Hibler, A dynamic thermodynamic sea ice model, J. Phys. Oceanogr. 9 (4) (1979) 815–846.
[2] E.C. Hunke, J.K. Dukowicz, An elastic-viscous-plastic model for sea ice dynamics, J. Phys. Oceanogr. 27 (9) (1997) 1849–1867.
[3] J.-F. Lemieux, D.A. Knoll, M. Losch, C. Girard, A second-order accurate in time IMplicit–EXplicit (IMEX) integration scheme for sea ice dynamics, J. Comput. Phys. 263 (2014) 375–392.
[4] M. Losch, D. Menemenlis, J.-M. Campin, P. Heimbach, C. Hill, On the formulation of sea-ice models. Part 1: effects of different solver implementations and parameterizations, Ocean Model. 33 (1–2) (2010) 129–144.
[5] L. Girard, S. Bouillon, J. Weiss, D. Amitrano, T. Fichefet, V. Legat, A new modeling framework for sea-ice mechanics based on elasto-brittle rheology, Ann. Glaciol. 52 (57) (2011) 123–132.
[6] S. Bouillon, P. Rampal, Presentation of the dynamical core of neXtSIM, a new sea ice model, Ocean Model. 91 (2015) 23–37, http://dx.doi.org/10.1016/j.ocemod.2015.04.005.
[7] P. Rampal, S. Bouillon, E. Ólason, M. Morlighem, neXtSIM: a new Lagrangian sea ice model, Cryosphere 10 (3) (2016) 1055–1073, http://dx.doi.org/10.5194/tc-10-1055-2016.
[8] P. Sakov, F. Counillon, L. Bertino, K.A. Lisæter, P.R. Oke, A. Korablev, TOPAZ4: an ocean-sea ice data assimilation system for the North Atlantic and Arctic, Ocean Science 8 (4) (2012) 633–656.
[9] D.A. Hebert, R.A. Allard, E.J. Metzger, P.G. Posey, R.H. Preller, A.J. Wallcraft, M.W. Phelps, O.M. Smedstad, Short-term sea ice forecasting: an assessment of ice concentration and ice drift forecasts using the U.S. Navy's Arctic Cap Nowcast/Forecast System, J. Geophys. Res., Oceans 120 (12) (2015) 8327–8345.
[10] G.C. Smith, F. Roy, M. Reszka, D. Surcel Colan, Z. He, D. Deacu, J.-M. Belanger, S. Skachko, Y. Liu, F. Dupont, J.-F. Lemieux, C. Beaudoin, B. Tranchant, M. Drévillon, G. Garric, C.-E. Testut, J.-M. Lellouche, P. Pellerin, H. Ritchie, Y. Lu, F. Davidson, M. Buehner, A. Caya, M. Lajoie, Sea ice forecast verification in the Canadian Global Ice Ocean Prediction System, Q. J. R. Meteorol. Soc. 140 (2015) 881–894, http://dx.doi.org/10.1002/qj.2555.
[11] M. Leppäranta, The Drift of Sea Ice, Springer, Helsinki, 2005.
[12] R.A. Walters, E.M. Lane, E. Hanert, Useful time-stepping methods for the Coriolis term in a shallow water model, Ocean Model. 28 (1–3) (2009) 66–74, http://dx.doi.org/10.1016/j.ocemod.2008.10.004.
[13] A. Ern, J. Guermond, Theory and Practice of Finite Elements, Appl. Math. Sci., vol. 159, Springer, 2004, http://books.google.fr/books?id=CCjm79FbJbcC.
[14] T.J.R. Hughes, The Finite Element Method: Linear Static and Dynamic Finite Element Analysis, Dover Civil and Mechanical Engineering, Dover Publications, 2012.
[15] V. Kumar, A. Grama, A. Gupta, G. Karypis, Introduction to Parallel Computing: Design and Analysis of Algorithms, vol. 400, Benjamin/Cummings, Redwood City, CA, 1994.
[16] G.H. Golub, J.M. Ortega, Scientific Computing: An Introduction with Parallel Computing, Elsevier, 2014.
[17] F. Hecht, Bamg: bidimensional anisotropic mesh generator, Source code, http://www.ann.jussieu.fr/~hecht/ftp/bamg.
[18] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra Mpi, The compete reference, Comput. Math. Appl. 31 (11) (1996) 140.
[19] S. Koranne, Boost c++ libraries, in: Handbook of Open Source Tools, Springer, 2011, pp. 127–143.

[20] C. Geuzaine, J.-F. Remacle Gmsh, A 3-d finite element mesh generator with built-in pre-and post-processing facilities, Int. J. Numer. Methods Eng. 79 (11) (2009) 1309–1331.
[21] G. Karypis, V. Kumar, Metis, A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN, 1998.
[22] S. Balay, K. Buschelman, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, L.C. McInnes, B.F. Smith, H. Zhang, PETSc Users Manual, Tech. Rep. ANL-95/11 – Revision 2.1.5, Argonne National Laboratory, 2004.
[23] W. Huang, Mathematical principles of anisotropic mesh adaptation, Commun. Comput. Phys. 1 (2) (2006) 276–310.
[24] M. Piggott, C. Pain, G. Gorman, P. Power, A. Goddard, h, r, and hr adaptivity with applications in numerical ocean modelling, in: The Second International Workshop on Unstructured Mesh Numerical Modelling of Coastal, Shelf and Ocean Flows, Ocean Model. 10 (1–2) (2005) 95–113, http://dx.doi.org/10.1016/j.ocemod.2004.07.007.
[25] H. Borouchaki, P.L. George, F. Hecht, P. Laug, E. Saltel, Delaunay mesh generation governed by metric specifications. Part I. Algorithms, Finite Elem. Anal. Des. 25 (1) (1997) 61–83.
[26] E. Larour, H. Seroussi, M. Morlighem, E. Rignot, Continental scale, high order, high spatial resolution, ice sheet modeling using the ice sheet system model (issm), J. Geophys. Res., Earth Surf. 117 (F1) (2012), http://dx.doi.org/10.1029/2011JF002140.
[27] A. Samaké, Large Scale Nonconforming Domain Decomposition Methods, Phd thesis, Université de Grenoble, Dec. 2014, https://tel.archives-ouvertes.fr/tel-01092968.
[28] T. Coupez, H. Digonnet, R. Ducloux, Parallel meshing and remeshing, Appl. Math. Model. 25 (2) (2000) 153–175, http://dx.doi.org/10.1016/S0307-904X(00)00045-7.