Parallel performance study of block-preconditioned iterative methods on multicore computer systems

Ali Dorostkar[*], Dimitar Lukarski[*], Björn Lund[†], Maya Neytcheva[*], Yvan Notay[‡], Peter Schmidt[†]

Abstract

In this work we benchmark the performance of a preconditioned iterative method, used in large scale computer simulations of a geophysical application, namely, the elastic Glacial Isostatic Adjustment model. The model is discretized using the finite element method. It gives raise to algebraic systems of equations with matrices that are large, sparse, nonsymmetric, indefinite and with a saddle point structure. The efficiency of solving systems of the latter type is crucial as it is to be embedded in a time-evolution procedure, where systems with matrices of similar type have to be solved repeatedly many times.

The computer code for the simulations is implemented using available open source software packages - Deal.ii, Trilinos, PARALUTION and AGMG. These packages provide toolboxes with state-of-art implementations of iterative solution methods and preconditioners for multicore computer platforms and GPU. We present performance results in terms of numerical and computational efficiency, number of iterations and execution time, and compare the timing results against a sparse direct solver from a commercial finite element package, that is often used by applied scientists in their simulations.

Keywords: glacial isostatic adjustment, iterative methods, block-preconditioners, inner-outer iterations, multicore, GPU, performance

1 Introduction

Solving realistic, large scale applied problems with the most modern numerical methods can be seen as a multidimensional optimization problem, with many levels of complexity that have to be simultaneously taken into account. We do not have anymore just one

[*]Department of Information Technology, Uppsala University, Sweden, ali.dorostkar, dimitar.lukarski, maya.neytcheva@it.uu.se

[†]Department of Earth Sciences, Uppsala University, Sweden, bjorn.lund, peter.schmidt@geo.uu.se

[‡]Numerical Analysis Group Service de Métrologie Nucléaire, Université Libre de Bruxelles, Belgium, ynotay@ulb.ac.be

single method to be implemented and optimized on some computer platform. The code that enables such large scale computer simulations usually requires a whole collection of algorithms, such as unstructured, adaptive or moving meshes; time-dependent processes that require the repeated solution of nonlinear and/or linear systems; inner-outer solution procedures, block preconditioners that utilize internal algebraic structures, methods as algebraic multigrid that have a recursive nature. All this has to work efficiently on the modern computer architectures. Another aspect to mention is that codes at this level of complexity can no longer be written from scratch.

In this work we consider an example a large scale problem from Geophysics. We implement it using several publicly available high quality libraries and compare the performance of the underlying advanced multi-level numerical solver, to investigate the resulting scalability and performance on multicore CPU and GPU.

The paper is organized as follows. Section 2 describes the target problem and the mathematical model in its full complexity as well as the simplified version of it, used in the numerical experiments. In Section 3 we outline the numerical solution method and the chosen acceleration technique - a block lower-triangular preconditioner. We also discuss the matrix approximations that have a crucial role in preserving the numerical efficiency and also for obtaining high quality solution, together with the related computational cost. Section 4 depicts the most important characteristics of the software packages used for the computer implementation of the numerical solution procedure. The results of the computer experiments are shown in Section 5. We illustrate both the numerical and computational efficiency of the numerical simulations, as well as the scalability and the parallel performance on multicore and GPU platforms. Conclusions are found in Section 6.

2 Description of the problem – discretization and algebraic formulation

As already stated, we aim to study the performance of a highly complex numerical solution procedure. It involves a block-preconditioned inner-outer iteration solver, designed for large scale nonsymmetric saddle point systems and is implemented on modern multicore and GPU architectures.

The applied problem, that gives raise to the large scale linear systems of algebraic equations to be solved, is the so-called glacial isostatic adjustment (GIA) model. It comprises the response of the solid Earth to redistribution of mass due to alternating glaciation and deglaciation periods. The processes that cause subsidence or uplift of the Earth surface are active today. To fully understand the interplay between the different processes, and, for example, be able to predict how coast lines will be affected and how glaciers and ice sheets will retreat, these have to be coupled also to recent global warming trend and melting of the current ice sheets and glaciers world wide. The long-term aim is to couple GIA modeling with other large scale models, such as Climate and Sea-level changes, Ice modeling etc. In this work, however, we consider only GIA models.

2.1 Mathematical model

Although in the numerical experiments we deal with a GIA model of modest mathematical difficulty, we describe the applied problem in its full complexity, showing that the part, handled here, appears as a building block in the extended simulation context.

The mathematical description of GIA processes employs a generalized visco-elastic non-linear rheology that accounts for viscous flow as the main mechanism of stress relaxation. Models, describing the response of the solid Earth to load, are based on the concept of isostacy, where the weight of the load is compensated for by adjustment in, and buoyancy of, the viscous mantle. The underlying physics is governed by the following coupled partial differential equations (PDEs):

$$\underbrace{\nabla \cdot \boldsymbol{\sigma}}_{(A)} - \underbrace{\nabla(\rho_0 \mathbf{u} \cdot \nabla \Phi_0)}_{(B)} - \underbrace{\rho_1 \nabla \Phi_0}_{(C)} - \underbrace{\rho_0 \nabla \Phi_1}_{(D)} = \mathbf{0} \quad \text{in } \Omega \subset \mathbb{R}^d, d = 2, 3 \tag{1a}$$

$$\nabla \cdot (\nabla \Phi_1) - 4\pi G \rho_1 = 0, \tag{1b}$$

$$\rho_1 + \rho_0 \nabla \cdot \mathbf{u} + \mathbf{u} \cdot \frac{\partial \rho_0}{\partial \mathbf{r}} = 0. \tag{1c}$$

Equation (1a) describes the equilibrium state (conservation of momentum) of a hydrostatically pre-stressed self-gravitating viscoelastic spherical body, where $\boldsymbol{\sigma}, \mathbf{u} = [u_i]_{i=1}^d, \Phi, G$ are the stress tensor, the displacement vector, the gravitational potential and Newton's gravitational constant, respectively; $\rho_0(r)$ is the initial density distribution, $\mathbf{r}$ is a vector along the radial direction and r is the radial distance from the centre of the Earth. Subscript '1' denotes the corresponding perturbed state of Φ and ρ.

Term (A) describes the force due to spatial gradients in stress. Term (B) represents the so-called *advection of pre-stress* and describes how the initial hydrostatic background stress is carried by the moving material. Incorporating term (B) has proven to be crucial for the successful modelling of the underlying physical processes [36]. Term (C) is the result of internal buoyancy and term (D) includes self-gravitation effects. The density ρ_1 is determined by the linearized continuity equation (1c).

In addition, equation (1a) has to be equipped with appropriate constitutive relations. The general form of these relations, describing stress as a function of strain, respectively, displacements, is expressed as follows,

$$\boldsymbol{\sigma}(\mathbf{x}, t) = \boldsymbol{\sigma}_E(\mathbf{x}) - \boldsymbol{\sigma}_I(\mathbf{x}, t), \tag{2}$$

where $\boldsymbol{\sigma}_E(\mathbf{x}) = C(\mathbf{x}, 0)\boldsymbol{\varepsilon}_E$ is the instantaneous stress due to elastic (reversible) response to load and $\boldsymbol{\sigma}_I(\mathbf{x}, t) = C(\mathbf{x}, t)\boldsymbol{\varepsilon}_I$ is the contribution due to inelastic response. Here $\boldsymbol{\varepsilon}$ is the strain tensor, $C(\mathbf{x}, t)$ is the constitutive or stress relaxation tensor, which entries are functions of the linear elasticity material (Lamé) coefficients at $t = 0$ and their viscoelastic analogues at $t \neq 0$. Here, the total strain is $\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}_E + \boldsymbol{\varepsilon}_I$ and infinitesimal strain is assumed, $\boldsymbol{\varepsilon} = \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)$.

Due to stress-strain relations, the form of $\boldsymbol{\sigma}_I$ depends on the inelastic strain-rate $\dot{\boldsymbol{\varepsilon}}$, that contributes to the total stain-rate $\dot{\boldsymbol{\varepsilon}}$ (here the dot denotes time derivative). In general

$\dot{\boldsymbol{\varepsilon}}_I$ represents some relaxation process, and can be described in a general form as

$$\dot{\boldsymbol{\varepsilon}}_I = \dot{\gamma} R = f(\boldsymbol{\sigma}, \gamma) R, \tag{3}$$

where γ is the amplitude of inelastic strain and R is a unitary symmetric tensor, representing local directions of inelastic strain rate. The general form of the rheology $\dot{\gamma} = f(\boldsymbol{\sigma}, \gamma)$ allows for expressing various relaxation mechanisms – viscoelastic relaxation, fault creep and poroelasticity, see, e.g., [12]. Then, the stress evolution is described via a heredity equation of the form

$$\boldsymbol{\sigma}(\mathbf{x}, t) = C(\mathbf{x}, 0)\boldsymbol{\varepsilon}_E - \int_0^t \dot{\gamma} C(\mathbf{x}, t) R \, d\tau. \tag{4}$$

In this way, the mathematical model (1a),(1c), (4) becomes a coupled system of integro-differential equations of high complexity. The main difficulties when performing computer simulations of the GIA model lie in the coupled system of equations for $\mathbf{u}$, ρ_1 and Φ_1, the very long time period (hundreds of thousands of years), the very large spatial domain, material inhomogeneity of the crust and the upper mantel, and the necessity to consider 3D spherical Earth models in order to obtain a physically correct behaviour of the model ([34]).

We consider here relation (2) with the following particular choice of $\boldsymbol{\sigma}_I$,

$$\boldsymbol{\sigma}(\mathbf{x}, t)_I = \int_0^t \frac{\partial C(\mathbf{x}, t-\tau)}{\partial \tau} \boldsymbol{\varepsilon}(\mathbf{x}, \tau) \, d\tau, \tag{5}$$

referred to as Hooke's law with memory.

Utilizing the standard relations between stress, strain and displacements, given by Hooke's law for a homogeneous, isotropic, linear, and purely elastic lithosphere, namely,

$$\sigma(\mathbf{u}) = 2\mu\varepsilon(\mathbf{u}) + \lambda(\nabla \cdot \mathbf{u})I \tag{6}$$

we assume that

$$\begin{aligned} \boldsymbol{\sigma}_E(\mathbf{x}) &= 2\mu_E \boldsymbol{\varepsilon}(\mathbf{u}(\mathbf{x}, 0)) + \lambda_E \nabla \cdot \mathbf{u}(\mathbf{x}, 0) I \\ \boldsymbol{\sigma}_I(\mathbf{x}, t) &= 2\partial_t \mu(\mathbf{x}, t) \boldsymbol{\varepsilon}(\mathbf{u}(\mathbf{x}, t)) + \partial_t \lambda(\mathbf{x}, t) \nabla \cdot \mathbf{u}(\mathbf{x}, t) I. \end{aligned} \tag{7}$$

The presence of the memory term (5) couples the displacements in (1a) during the whole time interval of interest. This requires special care during the modelling as well as in the discretization and the numerical solution of the arising algebraic systems. To simplify the numerics, we assume in addition, that the stress relaxation functions obey the so-called Maxwell model, i.e., the time-dependence in the stress field is due to time-dependent material coefficients only and is of the form

$$\begin{aligned} \mu(t, \tau) &= \mu_E e^{-\alpha_0 (t-\tau)} = \mu_E \chi(\alpha_0, t, \tau) \\ \lambda(t, \tau) &= \lambda_E e^{-\alpha_0 (t-\tau)} = \lambda_E \chi(\alpha_0, t, \tau). \end{aligned} \tag{8}$$

In (8), the coefficients μ_E, λ_E and $\mu(t,\tau)$, $\lambda(t,\tau)$ are Lamé coefficients in the elastic case and the viscoelastic case correspondingly, η is the viscosity parameter, and $\alpha_0 = \mu_E/\eta$ is the inverse of the so-called Maxwell time. We note that μ_E and λ_E are related to he Young modulus E and the Poisson ratio ν as

$$\mu = E/2/(1+\nu), \quad \lambda = 2\mu\nu/(1-2\nu). \tag{9}$$

In this study we deal with a simplified formulation of the problem. As a first reduction of the complexity, we neglect the self-gravitation effects (term (D) in (1a) and equations (1b) and (1c)) and use a two dimensional model, where the Earth is a flat viscoelastic homogeneous material body, that can be modelled as compressible or incompressible. Namely, we consider the momentum equation for quasi-static perturbations of a homogeneous, elastic continuum in a constant gravity field

$$-\nabla\cdot\sigma - \nabla(\mathbf{u}\cdot\nabla p_0) + (\nabla\cdot\mathbf{u})\nabla p_0 = \mathbf{f} \text{ in } \Omega \subset \mathbb{R}^2 \tag{10a}$$

with boundary conditions

$$\sigma(\mathbf{u})\cdot\mathbf{n} = \ell \text{ on } \Gamma_L \qquad \sigma(\mathbf{u})\cdot\mathbf{n} = \mathbf{0} \text{ on } \Gamma_N \tag{10b}$$

$$\mathbf{u} = \mathbf{0} \text{ on } \Gamma_D, \qquad u_1 = 0,\ \partial_x u_2 = 0 \text{ on } \Gamma_S. \tag{10c}$$

The geometry of the problem is shown in Figure 1.

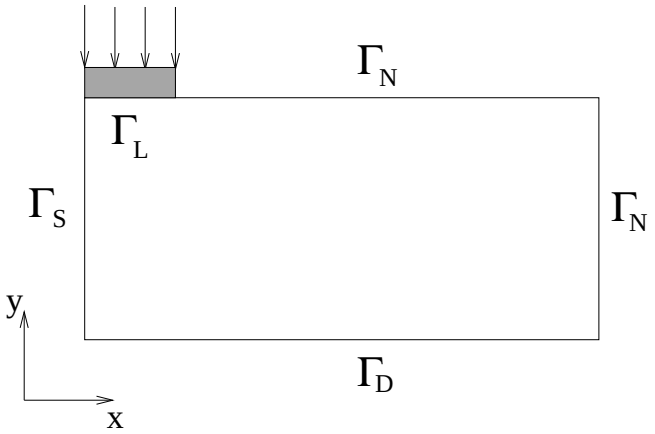

Figure 1: The geometry of the problem

In Equation (10a), p_0 is the so-called *pre-stress*, $\mathbf{f}$ is a body force, and ℓ is a surface load. The third term on the left hand side of Equation (10a) describes the *buoyancy* of the compressed material, and it vanishes for purely incompressible material since $\nabla\cdot\mathbf{u} = 0$.

In order to compensate for excluding self-gravitation effects, we need to model fully incompressible materials, i.e., for which $\nu = 0.5$. Note, however, that for fully incompressible linear elastic solid the problem is not well-posed, since λ becomes unbounded. It is seen from the definition of the Lamé coefficient λ in (9) that when ν approaches 0.5, λ becomes

unbounded. Thus, when $\nu \to 0.5$, the problem in Equation (6) becomes ill-posed, and the corresponding discrete analogue of Equation (10a) becomes extremely ill-conditioned. This is the mathematical formulation of the phenomenon known as *volumetric locking*, which may lead to erroneous results when solving the discretized Equation (10a) in the nearly incompressible limit. See, for example, [14], for further details on the locking effect.

A known remedy to the locking problem is to introduce the so-called *kinematic pressure* $p = \frac{\lambda}{\mu}\nabla \cdot \mathbf{u}$, and reformulate Equation (10a) as a coupled system of PDEs, which yields

$$-\nabla \cdot (2\mu\varepsilon(\mathbf{u})) - \nabla(\mathbf{u} \cdot \nabla p_0) + (\nabla \cdot \mathbf{u})\nabla p_0 - \mu\nabla p = \mathbf{f} \text{ in } \Omega \tag{11a}$$

$$\mu\nabla \cdot \mathbf{u} - \frac{\mu^2}{\lambda}p = 0 \text{ in } \Omega \tag{11b}$$

with boundary conditions

$$\begin{aligned} [2\mu\varepsilon(\mathbf{u}) + \mu p I] \cdot \mathbf{n} &= \ell \text{ on } \Gamma_L \\ [2\mu\varepsilon(\mathbf{u}) + \mu p I] \cdot \mathbf{n} &= \mathbf{0} \text{ on } \Gamma_N \\ \mathbf{u} &= \mathbf{0} \text{ on } \Gamma_D \\ u_1 = 0, \partial_x u_2 &= 0 \text{ on } \Gamma_S. \end{aligned}$$

In the sequel we consider the solution of (11), together with (4), (5), (7) and (8).

2.2 Space discretization and algebraic formulation

We next perform a finite element space discretization of Ω, namely, consider a discretized domain Ω_h and some finite dimensional subspaces $V_h \subset V$ and $P_h \subset P$. To this end, we use mixed finite elements and a stable finite element pair of spaces for the displacements and the pressure, in order to satisfy the LBB condition (see, for instance [15] for more details).

The handling of the visco-elastic problem and the corresponding numerical procedure are described in detail in [24]. In brief, we obtain a matrix-vector form of the problem:

at time t_j, find the displacements $\mathbf{u}_j$ and the pressure $\mathbf{p}_j$ by solving the linear system

$$\mathcal{A}_j \begin{bmatrix} \mathbf{u}_j \\ \mathbf{p}_j \end{bmatrix} = \begin{bmatrix} \mathbf{r}_j^{(1)} \\ \mathbf{r}_j^{(2)} \end{bmatrix}, \tag{12a}$$

$$\text{where } \mathcal{A}_j = \mathcal{A} - \frac{\Delta t_j}{2}\mathcal{A}_0,\ \mathcal{A} = \begin{bmatrix} M & B^T \\ B & -C \end{bmatrix} \text{ and } \mathcal{A}_0 = \begin{bmatrix} M_0 & B_0^T \\ B_0 & -C_0 \end{bmatrix}. \tag{12b}$$

The detailed forms of $\mathbf{r}_j^{(1)}$ and $\mathbf{r}_j^{(2)}$ and the matrix blocks are given in [24]. The matrices M_0, B_0, C_0 correspond to certain bilinear forms $\widetilde{a}$, $\widetilde{b}$, $\widetilde{c}$, evaluated at $\tau = t$ and therefore do not explicitly depend on t. The blocks in $\mathcal{A}$ do depend however on α_0, see (8), which in its turn depends on the problem coefficients, and the latter might be discontinuous in space. The problem has been earlier studied in [24, 9, 10], where independence of the convergence of the preconditioned iterative method of possible discontinuous problem coefficient is shown.

To summarize, at each time t_j we have to solve a linear system with the matrix $\mathcal{A}_j$ given in (12).

3 Numerical solution method and preconditioning

We assume that Δt_j is small enough and from now on we investigate the solution of one representative system of equations of type (12b),

$$\mathcal{A}\mathbf{x} = \begin{bmatrix} M & B^T \\ B & -C \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}, \tag{13}$$

where $M \in \mathbb{R}^{N_u \times N_u}$ is non-symmetric, sparse and in general indefinite, and $C^{N_p \times N_p}$ is positive semi-definite.

The algebraic problem in (13) is solved with an iterative solution method, preconditioned by the block-lower triangular matrix

$$\mathcal{D} = \begin{bmatrix} \widetilde{M} & 0 \\ B & -\widetilde{S} \end{bmatrix}, \tag{14}$$

where the first diagonal block $\widetilde{M}$ approximates M, and the second diagonal block, $\widetilde{S}$, approximates the negative Schur complement of $\mathcal{A}$, $S = C + BM^{-1}B^T$. The block-triangular matrix $\mathcal{D}$ is only one possible choice of a preconditioner for $\mathcal{A}$ and we refer to [13, 4] for an extensive survey over existing preconditioning and solution techniques for saddle point problems.

A matrix of block-lower triangular form, as in (14), is among the most often used preconditioners for matrices of saddle point form as in (13). The matrix

$$\mathcal{D}_I = \begin{bmatrix} M & 0 \\ B & -S \end{bmatrix} \tag{15}$$

is referred to as the *ideal preconditioner*, as it clusters the spectrum of $\mathcal{D}_I^{-1}\mathcal{A}$ in just two points, -1 and 1, ensuring that, for instance, GMRES (see [31]) will converge in just two iterations.

Approximating M and S by $\widetilde{M}$ and $\widetilde{S}$ causes an increase in the number of iterations of the preconditioned iterative method. We include a simple eigenvalue analysis to show the importance of using good approximations of both M and S. Consider first

$$\mathcal{D}_{AE} = \begin{bmatrix} \widetilde{M} & 0 \\ B & -S \end{bmatrix} \tag{16}$$

and the generalized eigenvalue problem $\mathcal{A}\mathbf{v} = \lambda\mathcal{D}_{AE}\mathbf{v}$. To simplify the derivations, we analyse

$$(\mathcal{A} - \mathcal{D}_{AE})\mathbf{v} = (\lambda - 1)\mathcal{D}_{AE}\mathbf{v}.$$

In detail,

$$\begin{bmatrix} M - \widetilde{M} & B^T \\ 0 & S - C \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{bmatrix} = (\lambda - 1) \begin{bmatrix} \widetilde{M} & 0 \\ B & -S \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{bmatrix}.$$

Equivalently,

$$\begin{bmatrix} \widetilde{M} & 0 \\ B & -S \end{bmatrix}^{-1} \begin{bmatrix} M-\widetilde{M} & B^T \\ 0 & S-C \end{bmatrix} = \begin{bmatrix} \widetilde{M}^{-1} & 0 \\ S^{-1}B\widetilde{M}^{-1} & -S^{-1} \end{bmatrix} \begin{bmatrix} M-\widetilde{M} & B^T \\ 0 & S-C \end{bmatrix}$$

$$= \begin{bmatrix} \widetilde{M}^{-1}M - I_1 & \widetilde{M}^{-1}B^T \\ S^{-1}B(\widetilde{M}^{-1}M - I_1) & -S^{-1}B(\widetilde{M}^{-1}M - I_1)M^{-1}B^T \end{bmatrix}.$$

Here I_1 and I_2 are identity matrices of corresponding order.

We see, that even if we use the exact Schur complements in the preconditioner, the errors due to approximating the first block (M) spread out in the whole preconditioned matrix. Thus, $\widetilde{M}$ must approximate M very well. In order to control the quality of this approximation and the error $\widetilde{M}^{-1}M - I_1$, in our experiments we use an inner solver for M with some suitable preconditioner. The usage of an inner solver is denoted in the sequel by $[M]$.

Consider next the effect of approximating S by $\widetilde{S}$, provided that $\widetilde{M} \equiv M$, i.e., let

$$\mathcal{D}_{EA} = \begin{bmatrix} M & 0 \\ B & -\widetilde{S} \end{bmatrix}. \tag{17}$$

Analogous derivation shows that

$$\mathcal{D}_{EA}^{-1}\mathcal{A} = \begin{bmatrix} M^{-1} & 0 \\ \widetilde{S}^{-1}BM^{-1} & -\widetilde{S}^{-1} \end{bmatrix} \begin{bmatrix} 0 & B^T \\ 0 & \widetilde{S}-C \end{bmatrix} = \begin{bmatrix} 0 & M^{-1}B^T \\ 0 & \widetilde{S}^{-1}S - I_2 \end{bmatrix},$$

i.e., the quality of $\widetilde{S}$ as an approximation of S has less profound effect, compared with that of $\widetilde{M}$. This is confirmed by the rigorous eigenvalue analysis developed in [28] (see in particular Corollary 4.5).

The preconditioner in (14) originates from the exact factorization of $\mathcal{A}$ as

$$\mathcal{A} = \begin{bmatrix} M & 0 \\ B & -S \end{bmatrix} \begin{bmatrix} I_1 & M^{-1}B^T \\ 0 & I_2 \end{bmatrix}$$

For the purpose of this study we compute $\widetilde{S}$ using the so-called *element-by-element* approach, see for instance, [19, 5, 24, 23] and the references therein. For completeness we briefly describe here the construction of $\widetilde{S}$. We notice first, that for any finite element pair of spaces, chosen to approximate $\mathbf{u}$ and p, the system matrix $\mathcal{A}$ can be assembled from element matrices that are also of saddle point form. Namely,

$$\mathcal{A} = \sum_{i=1}^{m} R_i^T \mathcal{A}_i^{(e)} R_i, \quad \text{where} \quad \mathcal{A}_i^{(e)} = \begin{bmatrix} M_i^{(e)} & (B_i^{(e)})^T \\ B_i^{(e)} & -C_i^{(e)} \end{bmatrix}, \tag{18}$$

R_i are Boolean matrices that define local-to-global mapping of the degrees of freedom and m is the number of the finite elements in the discretization mesh.

In this particular case $\mathbf{u}$ is discretized using piece-wise quadratic and p - using piece-wise linear basis functions on quadrilateral mesh.

Based on (18), we then compute element-wise small-sized Schur complements $S_i^{(e)} = C_i^{(e)} + B_i^{(e)} \left(M_i^{(e)} + h^2 I^{(e)} \right)^{-1} (B_i^{(e)})^T, \quad i = 1, 2, \cdots, m,$ where h is the characteristic size of the spatial mesh. The diagonal perturbation of $M_i^{(e)}$ is needed since these matrices are singular. Then, $\widetilde{S}$ is obtained via finite element assembly of the local matrices $S_i^{(e)}$, $\widetilde{S} = \sum_{i=1}^{m} \widetilde{R}_i^T S_i^{(e)} \widetilde{R}_i$.

We next describe the solution procedure in more detail. The system $\mathcal{A}$ are solved by GMRES or its flexible variant FGMRES, cf. [30], referred to as the *outer solver*. The choice of the outer method is due to the fact that M is nonsymmetric. The preconditioner is

$$\mathcal{D} = \begin{bmatrix} [M] & 0 \\ B & -[\widetilde{S}] \end{bmatrix}. \tag{19}$$

Systems with M and $\widetilde{S}$ are solved by inner iterations, again using GMRES and an algebraic Multigrid (AMG) method as a preconditioner.

We have some further options regarding $[M]$. Most straightforwardly we can construct an AMG preconditioner for the whole block M. The corresponding inner solver in the experiments is denoted by $[M]_0$.

Alternatively, we can use a block-diagonal preconditioner for M. An ordering of the degrees of freedom of the displacements first in x-direction and then in y-direction reveals a block two-by-two structure of the matrix M itself. For the purely elastic problem (with no advection terms), the diagonal blocks correspond to (mildly) anisotropic discrete Laplacians. From Korn's inequality we obtain that the block-diagonal part of M is spectrally equivalent to the whole block. For an explanation, see for instance, [3]. The block-diagonal part of M can still be used in the presence of advection, as long as it is not dominating. The resulting block-diagonally preconditioned inner solver in the experiments is denoted by $[M]_1$. Again, the diagonal blocks are solved by an AMG-preconditioned iterative method.

We note, that since $\mathcal{A}$ can be factored exactly as

$$\mathcal{A} = \begin{bmatrix} -\Sigma & B^T \\ 0 & -C \end{bmatrix} \begin{bmatrix} I_1 & 0 \\ -C^{-1}B & I_2 \end{bmatrix}$$

with $\Sigma = M + B^T C^{-1} B$, we can analogously precondition $\mathcal{A}$ by a matrix of the form

$$\mathcal{F} = \begin{bmatrix} F_1 & B^T \\ 0 & F_2 \end{bmatrix},$$

where F_1 approximates Σ and F_2 approximates C. The study of the properties of the element-by-element approximation of Σ is planned for th future research.

4 Implementation details

As already pointed out, computer simulations of realistic applied problems usually result in very complex coupled models. Implementing a fully functional and flexible code for such models needs excessive coding and a substantial amount of time. Over the past decades, many libraries have been developed to ease the development process for scientific computing applications. These libraries include state-of-art numerical solution methods that are robust and reliable due to many years of development and rigorous testing. Using such libraries helps the researchers to concentrate on the problem itself and not on implementation technicalities.

We describe next the software used to implement the solution procedure for (13), preconditioned by (19), using $[M]_0$ for the solution of systems with M. The implementation is developed mostly in *C++*, however, it links some additional solvers that are available in FORTRAN.

4.1 Deal.II

As a main finite element software toolbox we use Deal.ii. The name Deal.ii stands for Differential Equations Analysis Library. It is a finite element library aimed at solving systems of partial differential equations. Deal.ii is publicly available under an Open Source license.

In this study, Deal.ii is used as a tool for mesh generation, handling of the degrees of freedom(DOFs), assembling system matrices and computing the element-by-element Schur complement approximation. For more details on Deal.ii, see [8].

Due to the level of abstraction provided in Deal.ii one can, with minimal amount of effort, change problem specifications such as problem dimension, types of solvers and much more.

Deal.ii expands its functionality by providing interfaces to other packages such as Trilinos, PETSc [7], METIS [18] and others. Each package is introduced into Deal.ii as a wrapper and is utilized indirectly through Deal.ii methods. That is, rather than calling routines from the packages directly, Deal.ii routines are used which in turn setup and call the proper method. By using the proper data structure provided by the wrappers, one can avoid unnecessary data movement between Deal.ii and other packages.

A major ingredient for the preconditioning step in the proposed numerical method is the solution of systems with M and $\widetilde{S}$. For this purpose we use inner solvers with AMG preconditioning. As Deal.ii does not provide its own AMG, we use that of Trilinos and AGMG.

To have a cost free transition between Deal.ii methods and Trilinos solvers, we use the Trilinos data structure for sparse matrices and vectors, that is available in the Deal.ii wrapper.

4.2 Trilinos

Trilinos in its whole has a vast collection of algorithms within an object oriented framework aimed at scientific computing problems. More details about Trilinos can be found in [17]. In this study we only use some packages, related to sparse matrix storage and solution of linear system of equations, namely,

- Epetra for sparse matrix and vector storage,
- Teuchos for passing parameters to solver and preconditioner,
- ML for multigrid preconditioning,
- AZTEC for the iterative solver (GMRES).

Note that all these packages are not used directly, but through Deal.ii wrappers.

Deal.ii configures the algebraic Multigrid (AMG) preconditioner from Trilinos using the following default settings:

- Chebyshev smoother with two pre- and post-smoothing steps
- Uncoupled aggregation with threshold of 0.02
- One Multigrid cycle

We refer to [8] for a detailed description of the settings.

4.3 AGMG

AGMG implements an aggregation-based algebraic multigrid method [25]. It provides tools for constructing the preconditioner and to solve linear systems of equations, and is expected to be efficient for large systems arising from the discretization of scalar second order elliptic PDEs. The method is however purely algebraic. The software package provides subroutines, written in FORTRAN 90, which implement the method described in [26], with further improvements from [22, 27].

The parallel performance is tested on up to 370000 cores. However, parallel implementation only with MPI is available and, therefore, in this study we compare only the serial performance of this method.

The settings are the following. They all correspond to the AGMG defaults, except the first one (the default being to treat all unknowns uniformly).

- The coarsening is by double pairwise aggregation (with quality control as in [22, 27]), performed separately on the two components of the displacement vector.
- One forward Gauss-Seidel sweep for pre-smoothing and one backward Gauss-Seidel sweep for post-smoothing.
- K-cycle [29], i.e., two Krylov accelerated iterations at each intermediate level.
- The main iterative solver is the Generalized Conjugate Residual (GCR) method, [16].

4.4 PARALUTION

PARALUTION is a sparse linear algebra library with focus on exploring fine-grained parallelism, targeting modern processors and accelerators including multi/many-core CPU and GPU platforms. The main goal of the PARALUTION project is to provide a portable library containing iterative methods and preconditioners for linear systems with sparse matrices, to be run on state of the art hardware. Details can be found in [21]. The library provides build-in plug-in to Deal.ii. The plug-in is not a direct wrapper as for Trilinos, but exports and imports data from Deal.ii to PARALUTION. For solving the linear problem, we use the preconditioners $[M]_0$ and $[M]_1$, and AMG with the following settings:

- Coarse grid size - 2000
- Coupling strength - 0.001
- Coarsening type - smoothed aggregation
- Multi-colored Gauss-Seidel smoother with relaxation parameter set to 1.3, [20]
- One pre-smoothing step and two post-smoothing steps
- One multigrid cycle for preconditioning
- Smoother matrix format - as in ELLPACK (ELL)
- Operator matrix format - compressed sparse row (CSR)

The AMG is constructed entirely on the CPU, while the execution can be performed on the CPU or on the GPU without any code modification.

4.5 ABAQUS

ABAQUS is a general-purpose finite element analysis program, most suited for numerical modelling of structural response. It handles various stress problems, both with static and dynamic response. The program is designed to ease the solution of complex problems, and has a simple input language, with comprehensive data checking, as well as a wide range of preprocessing and post-processing options. However, enhanced numerical simulations of GIA problems are not straightforwardly performed with ABAQUS since important terms in the continuous model, such as prestress advection, cannot be added directly, leading to the necessity to modify the model in order to be able to use the package. These questions are described in detail in [38]. Further, ABAQUS cannot handle purely incompressible materials - ν cannot be set to 0.5 but to some closer value, such as 0.4999, for instance.

Nevertheless, ABAQUS offers highly optimized numerical solution methods. In particular, the available sparse direct solver in 2D shows nearly optimal computational complexity, see the results in [9] and the performance figures in Section 5. The direct solver can be executed in parallel and its scalability is also presented. We use here ABAQUS 6.12.

The iterative methods, included in ABAQUS can be tested only on 3D problems. For further details we refer to ABAQUS' user manual [1].

5 Performance analysis

In this section we present the results of the numerical experiments with different software packages. The computations are performed on the following computer resources:

(C1) CPU: Intel(R) Xeon(R) 1.6GHz 12 cores

(C2) CPU: Intel(R) Core(TM) i5-3550 CPU 3.30GHz 4 cores
GPU: NVIDIA K40, 12G, 2880 cores

The performance results for ABAQUS and Deal.ii/Trilinos/AGMG are obtained using the computer resource (C1). The CPU version of the solver from PARALUTION (ver 0.6.1) is tested on both (C1) and (C2), and the GPU implementation of it is executed on (C2). Parallelism is exploited by the built-in functionality of the packages to use OpenMP. The maximum number of threads used on each device is set to the number of cores available, thus, on (C1) executions are carried on one, four, eight and twelve threads, while on (C2) only four threads are used (without Hyperthreading).

>From (11) we observe that while enabling to solution of models with fully incompressible materials, we obtain a system of equation that is about 30% larger that that, solved with ABAQUS. The problem sizes are shown in Table 1.

It is evident from Table 1 that both solution techniques scale nearly exactly linearly with the problem size. However, for the two-dimensional problem at hand, the direct solver from ABAQUS is somewhat faster than the preconditioned iterative solver, implemented in Trilinos. The iterations presented in the table show the number of outer iterations and in brackets, the average number of inner iteration to solve with M and with the approximate Schur complement matrix $\widetilde{S}$. The optimal numerical efficiency of the iterative solver exhibits itself in the number of iterations, that are constant for the different number of threads and are nearly constant with increasing the problem size.

No. of	Deal.II + Trilinos				ABAQUS		
Threads	DOF	Iterations	Setup	Solve (2/3)	DOFs	Setup	Solve
1		21(6, 6)	3.43	69.9 (46.6)		7.44	59
4	1 479 043	21(6, 6)	3.04	47.3 (31.5)	986 626	7.49	33
8		21(6, 6)	3.00	42.9 (28.6)		7.51	28
1		18(8, 6)	15.4	317 (211)		29.72	269
4	5 907 203	18(8, 6)	14.2	210 (140)	3 939 330	29.93	145
8		18(8, 6)	14.1	297 (198)		29.94	122

Table 1: Comparison between Deal.ii and ABAQUS on (C1)

Table 2 shows a comparison between the performance of the iterative solver using Trilinos and PARALUTION. We see that PARALUTION solves the system faster than Trilinos but it uses more CPU time for the setup phase. The implementations both Trilinos and PARALUTION do not scale to more than four threads, which can be attributed to memory

bandwidth utilization (all solvers are bandwidth bounded). We also note, that the solution time for PARALUTION has a better scaling factor compared to Trilinos. Another observation is that PARALUTION is performing better than the direct solver from ABAQUS.

Threads	DOFS	Trilinos			PARALUTION		
		Iterations	Setup	Solve	Iterations	Setup	Solve
1	23 603	18(4, 4)	0.0597	0.581	24(5, 5)	0.0955	0.9116
4		18(4, 4)	0.0596	0.298	24(5, 5)	0.0669	0.5075
8		18(4, 4)	0.136	0.249	24(5, 5)	0.0607	0.5248
12		18(4, 4)	0.134	0.251	24(5, 5)	0.0757	0.5850
1	93 283	20(4, 4)	0.228	3.11	25(5, 5)	0.4059	2.9241
4		20(4, 4)	0.205	2.06	25(5, 5)	0.2459	1.1031
8		20(4, 4)	0.280	1.93	25(5, 5)	0.2259	0.9312
12		20(4, 4)	0.331	1.9	25(5, 5)	0.2154	0.8725
1	370 883	21(5, 5)	0.854	14.4	25(5, 5)	1.6689	12.2548
4		21(5, 5)	0.819	10	25(5, 5)	0.9709	4.9483
8		21(5, 5)	0.947	9.41	25(5, 5)	0.8673	5.1510
12		21(5, 5)	1.33	8.39	25(5, 5)	0.8166	4.5229
1	1 479 043	21(6, 6)	3.43	69.9	26(5, 5)	6.8383	51.6089
4		21(6, 6)	3.04	47.3	26(5, 5)	3.9909	20.9045
8		21(6, 6)	3.00	42.9	26(5, 5)	3.5134	22.4387
12		21(6, 6)	5.37	41.5	26(5, 5)	3.3536	18.5971
1	5 907 203	18(8, 6)	15.4	317	27(5, 5)	31.385	220.1920
4		18(8, 6)	14.2	210	27(5, 5)	16.2374	92.7028
8		18(8, 6)	14.1	297	27(5, 5)	13.9868	92.4935
12		18(8, 6)	22.1	180	27(5, 5)	13.1867	84.2809

Table 2: Comparison between Trilinos and PARALUTION on (C1), CPU only

As there is no GPU accelerator available on (C1), the results of the PARALUTION solver are reproduced on (C2) and act as a reference to compare the solution time on the GPU with that of ABAQUS and Trilinos. PARALUTION provides a seamless transition from running on CPU to running on an accelerator (GPU) and because it is already available on (C2), it is our CPU solver of choice. We use four threads on the CPU since the hardware provides only four cores. The results are presented in Table 3. We see that for smaller problem sizes the GPU solver from PARALUTION is slower than the CPU solver. As the problem size grows the GPU solver performs better and we obtain a speedup of up to four times. Solving the largest problem size is not possible on the GPU due to insufficient memory.

We note that the discretization of the problem, corresponding to 1 479 043 degrees of freedom, on the surface of the computational domain agrees with the placement of the surface sensors that gather data from geophysical experiments. This size fits on the GPU and the performance is fastest. Trying to solve problems with larger computational domain in 2D or 3D problems might fail due to insufficient memory on the currently available GPUs.

DOF	Outer iter.	Time on CPU (s)		Time on GPU (s)	
		Setup	Solve	Setup	Solve
23 603	24	0.04	0.39	0.16	2.8
93 283	24	0.18	1.26	0.27	2.0
370 883	24	0.75	5.07	0.9	3.8
1 479 043	25	2.9	22.4	3.7	5.8
5 907 203	25	18.4	91.5	out of memory	

Table 3: Comparison between CPU and GPU on (C2) using PARALUTION

Next, we take a more detailed look at the run time, spent on executing various parts of the solution procedure. To underline the need for fast preconditioner, on Table 4 we present the time distribution for the linear system for the Trilinos solver. It clearly shows that the solution with M takes the largest amount of time and the question how to reduce this cost naturally arises. The percentage spend on the preconditioner compared to the total solution time is similar for all other tested packages.

DOF	Solution with M	Solution with $\widetilde{S}$	Total solution time
23 603	0.625 (90%)	0.0286 (10%)	0.687
93 283	3.49 (93%)	0.0983 (7%)	3.72
370 883	15.3 (93%)	0.441 (7%)	16.3
1 479 043	77.2 (94%)	2.22 (6%)	81.8
5 907 203	350 (94%)	10.3 (6%)	370

Table 4: Solution time distribution using Trilinos-AMG

To optimize the solution process, we consider several options. As a first option, one could try to avoid the inner solver with M, replacing it either with one action of the AMG preconditioner or replace it with its block-diagonal part ($[M]_1$). Numerical simulations, not included in the report show that, as indicated by the analysis in Section 3, low accuracy solution with the top-left pivot block destroy the quality of the preconditioner, which leads to more outer iterations and requires a smaller stopping tolerance to compensate for the larger condition number of the preconditioned system. Therefore this approach is not recommended.

As a second option we can change the AMG implementation to investigate the potential scalability of the AMG implementation itself. To this end, we replace Trilinos-AMG by AGMG. The results in Table 5 show that AGMG is both more numerically and computationally efficient. The inner solver requires less iterations and those do not grow with size, the convergence of the inner solver is fast and allows for a smaller stopping tolerance. That means that for the same computational cost we solve more accurately with M, which, in turn, decreases the outer iterations and the overall simulation time. Since, we use the

AGMG only serially (there is no OpenMP implementation yet) we compare the timing with the Trilinos-AMG only using one thread (see Table 2).

	Trilinos			AGMG		
DOFs	Itr.	Setup	Solve	Itr.	Setup	Solve
23 603	18(4, 4)	0.0597	0.581	16(5, 4)	0.029	0.525
93 283	20(4, 4)	0.228	3.11	16(5, 4)	0.128	2.15
370 883	21(5, 5)	0.854	14.4	17(6, 5)	0.515	10.3
1 479 043	21(6, 6)	3.43	69.9	17(6, 5)	2.09	43.7
5 907 203	18(8, 6)	15.4	317	19(7, 6)	8.69	208

Table 5: Comparison between Trilinos-AMG and AGMG on (C1), CPU, serial mode

6 Conclusion

In this work we present a snapshot of the performance of a large scale computer simulation in terms of numerical efficiency, execution time and scalability on multicore platforms as well as on GPU.

First, we show that large scale coupled problems can be successfully implemented using publicly available numerical linear algebra software. Compared with highly specialized and optimized commercial software, the open source libraries, included in this study, allow to enhance the mathematical model and make it more realistic, adding features that are not straightforwardly incorporated when using commercial software.

Furthermore, we show that GPU devices can be used in complex numerical simulations with various combination of solvers and preconditioners. When the problem fits into the memory of the GPU, the PARALUTION-GPU implementation performs noticeably faster than all other tested CPU implementations.

Open source numerical libraries successfully compete with highly efficient commercial packages in terms of overall simulation time and show better price-performance ratio.

Due to the fact that all methods are memory bounded, none of the tested OpenMP-based CPU implementations scale linearly. This makes it necessary to extend the performance tests using MPI, which is a subject of future work.

Acknowledgments

This work has been supported by the Linnaeus center of excellence UPMARC, Uppsala Programming for Multicore Architectures Research Center.

References

[1] Abaqus FEA http://www.3ds.com/.

[2] O. Axelsson. *Iterative solution methods.* Cambridge University Press, Cambridge, 1994.

[3] O. Axelsson. On iterative solvers in structural mechanics; separate displacement orderings and mixed variable methods. *Math. Comput. Simulation*, 50(1-4):11–30, 1999. Modelling '98 (Prague).

[4] O. Axelsson. Milestones in the development of iterative solution methods. *J. Electr. Comput. Eng.*, pages Art. ID 972794, 33, 2010.

[5] O. Axelsson, R. Blaheta, and M. Neytcheva. Preconditioning of boundary value problems using elementwise schur complements. *SIAM J. Matrix Anal. Appl.*, 31(2):767–789, July 2009.

[6] I. Babuška. The finite element method with Lagrangian multipliers. *Numer. Math.*, 20:179–192, 1972/73.

[7] S. Balay, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, and H. Zhang. PETSc Web page. http://www.mcs.anl.gov/petsc, 2014.

[8] W. Bangerth, G. Kanschat, and R. Hartmann. `deal.II` differential equations analysis library, http://www.dealii.org.

[9] E. Bängtsson and B. Lund. A comparison between two solution techniques to solve the equations of glacially induced deformation of an elastic earth. *International Journal for Numerical Methods in Engineering*, 75(4):479–502, 2008.

[10] E. Bängtsson and M. Neytcheva. Numerical simulations of glacial rebound using preconditioned iterative solution methods. *Appl. Math.*, 50(3):183–201, 2005.

[11] E. Bängtsson and M. Neytcheva. An agglomerate multilevel preconditioner for linear isostasy saddle point problems. In *Large-scale scientific computing*, volume 3743 of *Lecture Notes in Comput. Sci.*, pages 113–120. Springer, Berlin, 2006.

[12] S. Barbot and Y. Fialko. A unified continuum representation of post-seismic relaxation mechanisms: semi-analytic models of afterslip, poroelastic rebound and viscoelastic flow. *Geophysical Journal International*, 182(3):1124–1140, 2010.

[13] M. Benzi, G. H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137, 5 2005.

[14] D. Braess. *Finite elements.* Cambridge University Press, Cambridge, third edition, 2007. Theory, fast solvers, and applications in elasticity theory.

[15] F. Brezzi. On the existence, uniqueness and approximation of saddle-point problems arising from Lagrangian multipliers. *Rev. Française Automat. Informat. Recherche Opérationnelle Sér. Rouge*, 8(R-2):129–151, 1974.

[16] S. C. Eisenstat, H. C. Elman, and M. H. Schultz. Variational iterative methods for nonsymmetric systems of linear equations. 20:345–357, 1983.

[17] M. A. Heroux and J. M. Willenbring. Trilinos Users Guide http://trilinos.sandia.gov. Technical Report SAND2003-2952, Sandia National Laboratories, 2003.

[18] G. Karypis and V. Kumar. MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0. http://www.cs.umn.edu/ metis, 2009.

[19] J. Kraus. Additive Schur complement approximation and application to multilevel preconditioning. *SIAM J. Sci. Comput.*, 34(6):A2872–A2895, 2012.

[20] D. Lukarski. *Parallel Sparse Linear Algebra for Multi-core and Many-core Platforms – Parallel Solvers and Preconditioners*. PhD thesis, Karlsruhe Institute of Technology, Jan. 2012.

[21] D. Lurkarski. Paralution project, http://www.paralution.com.

[22] A. Napov and Y. Notay. An algebraic multigrid method with guaranteed convergence rate. *SIAM J. Sci. Comput.*, 34(2):A1079–A1109, 2012.

[23] M. Neytcheva. On element-by-element Schur complement approximations. *Linear Algebra Appl.*, 434(11):2308–2324, 2011.

[24] M. Neytcheva and E. Bängtsson. Preconditioning of nonsymmetric saddle point systems as arising in modelling of viscoelastic problems. *Electronic Transactions on Numerical Analysis*, 29:193–211, 2008.

[25] Y. Notay. AGMG software and documentation, http://homepages.ulb.ac.be/ ynotay/AGMG.

[26] Y. Notay. An aggregation-based algebraic multigrid method. *Electron. Trans. Numer. Anal.*, 37:123–146, 2010.

[27] Y. Notay. Aggregation-based algebraic multigrid for convection-diffusion equations. *SIAM J. Sci. Comput.*, 34(4):A2288–A2316, 2012.

[28] Y. Notay. A new analysis of block preconditioners for saddle point problems. *SIAM J. Matrix Anal. Appl.*, 35:143–173, 2014.

[29] Y. Notay and P. S. Vassilevski. Recursive Krylov-based multigrid cycles. *Numer. Lin. Alg. Appl.*, 15:473–487, 2008.

[30] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14(2):461–469, 1993.

[31] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7(3):856–869, 1986.

[32] P. Schmidt, B. Lund, T. Árnadóttir, and H. Schmeling. Glacial isostatic adjustment constrains dehydration stiffening beneath iceland. *Earth and Planetary Science Letters*, 359–360(0):152 – 161, 2012.

[33] P. Schmidt, B. Lund, and C. Hieronymus. Implementation of the glacial rebound prestress advection correction in general-purpose finite element analysis software: Springs versus foundations. *Computers and Geosciences*, 40(0):97 – 106, 2012.

[34] G. Spada, V. R. Barletta, V. Klemann, R. E. M. Riva, Z. Martinec, P. Gasperini, B. Lund, D. Wolf, L. L. A. Vermeersen, and M. A. King. A benchmark study for glacial isostatic adjustment codes. *Geophysical Journal International*, 185(1):106–132, 2011.

[35] H. Steffen and P. Wu. Glacial isostatic adjustment in fennoscandia—a review of data and modeling. *Journal of Geodynamics*, 52(3–4):169 – 204, 2011.

[36] P. Wu. Viscoelastic versus viscous deformation and the advection of pre-stress. *Geophysical Journal International*, 108(1):136–142, 1992.

[37] P. Wu. Modelling postglacial sea levels with power-law rheology and a realistic ice model in the absence of ambient tectonic stress. *Geophysical Journal International*, 139(3):691–702, 1999.

[38] P. Wu. Using commercial finite element packages for the study of earth deformations, sea levels and the state of stress. *Geophysical Journal International*, 158(2):401–408, 2004.