CPU and GPU performance of large scale numerical simulations in Geophysics

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 that 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 implementation is based on available open source software packages - Deal.II, Trilinos, PARALUTION and AGMG. These packages provide toolboxes with state-of-the-art implementations of iterative solution methods and preconditioners for multicore computer platforms and GPU. We present performance results in terms of numerical and the 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, multicore, block-preconditioners, inner-outer iterations, CPU-GPU, performance

1 Introduction

Solving realistic, large scale applied problems with advanced numerical techniques 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 single method to be implemented and optimized on a given 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 in turn require the repeated solution of nonlinear and/or linear systems; inner-outer solution procedures, block preconditioners that utilize internal algebraic structures, solution

[⋆] 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

methods as algebraic multigrid that have a recursive nature. All this has to work efficiently on modern computer architectures. Another aspect to mention is that codes at this level of complexity can no longer be written from scratch but rather (combination of) ready toolboxes have to be used instead.

We consider as 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, the resulting scalability and performance on multicore CPU and GPU. Section 2 describes the simplified target problem and the mathematical model, used in the numerical experiments. In Section 3 we outline the solution method and the acceleration technique - a block lower-triangular preconditioner. Section 4 depicts the most important characteristics of the software packages used for the computer implementation of the numerical solution procedure. The experiments are described 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

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.

Mathematical model Although GIA model describes a very complex applied problem, here we deal with a model of modest mathematical difficulty. However, the test problem considered here appears as a building block in an extended simulation context, see [12] for more details.

The benchmark setting consists of a two-dimensional vertical cut of Earth's crust, assumed to be pre-stressed, homogeneous, visco-elastic and in a constant gravity field. The space domain is axisymmetric, 10000 km long and 4000 deep. We compute the deformations subject to the load of rectangular-shaped ice of sizes 1000 km times 2 km. The geometry of the problem is shown in Figure 1.

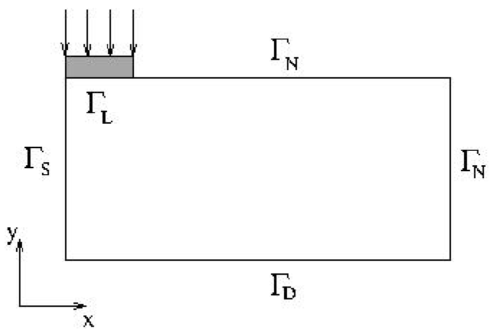


Fig. 1: The geometry of the problem

The momentum equation describing the quasi-static perturbations of such a material body is

$$-\underbrace{\nabla\cdot\sigma}_{(A)}-\underbrace{\nabla(\mathbf{u}\cdot\nabla p_0)}_{(B)}+\underbrace{(\nabla\cdot\mathbf{u})\nabla p_0}_{(C)}=\mathbf{f}\ \text{in}\ \Omega\subset\mathbb{R}^2\ , \tag{1}$$

where $\boldsymbol{\sigma}$ is the stress tensor, $\mathbf{u}=[u_i]_{i=1}^d$ is the displacement vector, p_0 is the so-called pre-stress and $\mathbf{f}$ is the body force. The boundary conditions are standard, $\sigma(\mathbf{u})\cdot\mathbf{n}=\ell$ on Γ_L, $\sigma(\mathbf{u})\cdot\mathbf{n}=\mathbf{0}$ on Γ_N, $\mathbf{u}=\mathbf{0}$ on Γ_D, $u_1=0$, $\partial_x u_2=0$ on Γ_S .

Term (A) describes the force due to spatial gradients in stress. Term (B) represents the so-called *advection of pre-stress* and has proven to be crucial for the successful modeling of the underlying physical processes [29]. Term (C) describes the buoyancy of the compressed material.

In addition to (1) we add appropriate constitutive relations, describing stress as a function of strain, displacements and time, namely, $\boldsymbol{\sigma}(\mathbf{x},t)=\boldsymbol{\sigma}_{\mathrm{E}}(\mathbf{x})-\boldsymbol{\sigma}_{\mathrm{I}}(\mathbf{x},t)$, where $\boldsymbol{\sigma}_{\mathrm{E}}(\mathbf{x})$ is the instantaneous stress due to elastic (reversible) response to load and $\boldsymbol{\sigma}_{\mathrm{I}}(\mathbf{x},t)$ is the contribution due to inelastic response. In the target context the stress evolution is described via a heredity equation of the form

$$\boldsymbol{\sigma}(\mathbf{x},t)=C(\mathbf{x},0)\boldsymbol{\varepsilon}_{\mathrm{E}}-\int_0^t\frac{\partial C(\mathbf{x},t-\tau)}{\partial\tau}\boldsymbol{\varepsilon}(\mathbf{x},\tau)\,d\tau\ .$$

Without describing the visco-elastic problem any further, we mention that we use the so-called Maxwell relaxation model, that simplifies the computation of the integral term. Here we consider the elastic part only, utilizing the standard relations between stress, strain and displacements, given by Hooke's law for a homogeneous, isotropic, linear, and purely elastic lithosphere, namely,

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

where the coefficients μ and λ are the Lamé coefficients. We note that μ and λ are related to the Young modulus E and the Poisson ratio ν as $\mu_{\mathrm{E}}=\frac{E}{2(1+\nu)}$ and $\lambda_{\mathrm{E}}=\frac{2\mu\nu}{1-2\nu}$.

In order to compensate for excluding self-gravitation effects, we need to model fully incompressible materials, i.e., for which $\nu = 0.5$. However, when $\nu \to 0.5$, λ becomes unbounded. Thus, Equation (2) becomes ill-posed, and the corresponding discrete analogue of Equation (1) becomes extremely ill-conditioned. This phenomenon is known as *volumetric locking*. See, for example, [10], 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 replacing the term $\lambda(\nabla \cdot \mathbf{u})I$ in (2), reformulate Equation (1) as a coupled system of PDEs, which yields

$$-\nabla \cdot (2\mu\varepsilon(\mathbf{u})) - \nabla(\mathbf{u} \cdot \nabla p_0) + (\nabla \cdot \mathbf{u})\nabla p_0 - \mu\nabla p = \mathbf{f} \text{ in } \Omega \tag{3a}$$

$$\mu\nabla \cdot \mathbf{u} - \frac{\mu^2}{\lambda}p = 0 \text{ in } \Omega \tag{3b}$$

with appropriate boundary conditions. Below we consider the solution of (3).

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 the Q2-Q1 stable finite element pair of spaces for the displacements and the pressure, in order to satisfy the LBB condition (see, e.g. [11] for more details).

Remark: The handling of the visco-elastic problem and the corresponding numerical procedure are described in detail in [22]. In brief, we obtain a matrix-vector form of the problem to be solved at time t_j find the displacements $\mathbf{u}_j$ and the pressure $\mathbf{p}_j$ by solving the linear system

$$\mathcal{A}_j \begin{bmatrix} \mathbf{u}_j \\ \mathbf{p}_j \end{bmatrix} = \begin{bmatrix} \mathbf{r}_j \\ \mathbf{q}_j \end{bmatrix}, \ \mathcal{A}_j = \mathcal{A} - \frac{\Delta t_j}{2}\mathcal{A}_0, \ \mathcal{A} = \begin{bmatrix} M & B^T \\ B & -C \end{bmatrix}, \ \mathcal{A}_0 = \begin{bmatrix} M_0 & B_0^T \\ B_0 & -C_0 \end{bmatrix}.$$

The detailed forms of $\mathbf{r}_j$ and $\mathbf{q}_j$ and the matrix blocks are given in [22].

3 Numerical solution method and preconditioning

To summarize, at each time t_j we have to solve a linear system with the matrix $\mathcal{A}_j$. We assume that Δt_j is small enough and from now on we investigate the solution of one representative system of equations of the type,

$$\mathcal{A}\mathbf{x} = \begin{bmatrix} M & B^T \\ B & -C \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}, \tag{4}$$

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. Thus, $\mathcal{A}$ is of saddle point form.

The algebraic problem in (4) is solved with an iterative solution method, preconditioned by the block lower-triangular matrix $\mathcal{D} = \begin{bmatrix} \widetilde{M} & 0 \\ B & -\widetilde{S} \end{bmatrix}$, where the block $\widetilde{M}$ approximates M, and the block $\widetilde{S}$, approximates the negative Schur complement of $\mathcal{A}$, $S = C + BM^{-1}B^T$. The block-triangular matrix $\mathcal{D}$ is one of the possible choices of a preconditioner for $\mathcal{A}$ and we refer to [9, 4] for an extensive survey on solution techniques for saddle point problems.

A matrix of the above block lower-triangular form is among the most often used preconditioners for matrices of saddle point form as in (4). For $\widetilde{M} = M$ ans $\widetilde{S} = S$, the matrix $\mathcal{D}$ 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 [28]) will converge in two-three iterations, the computational cost is, however, prohibitive.

Approximating M and S by $\widetilde{M}$ and $\widetilde{S}$ respectively causes an increase in the number of iterations of the preconditioned iterative method. Eigenvalue analysis reveals that $\widetilde{M}$ must approximate M very well. In order to control the quality of this approximation, we use an inner solver for M with some suitable preconditioner. The usage of an inner solver is denoted in the sequel by $[M]$. On the other hand, approximating S by $\widetilde{S}$ has less profound effect, compared with that of $\widetilde{M}$. This is confirmed by the rigorous eigenvalue analysis provided in [25] (see in particular Corollary 4.5). For the purpose of this study we compute $\widetilde{S}$ using the so-called *element-by-element* (EBE) approach, see for instance, [16, 5, 22, 21] and the references therein. The EBE technique is very attractive as it is easily patallelizable and produces a good quality sparse approximation of the, in general, dense Schur complement.

The system $\mathcal{A}$ is solved by the flexible variant of GMRES, FGMRES, cf. [28, 27], referred to as the *outer solver*. The choice of the outer method is due to the fact that M is nonsymmetric and that the preconditioner is variable, due to the inner solvers involved. The preconditioner is thus

$$\mathcal{D} = \begin{bmatrix} [M] & 0 \\ B & -[\widetilde{S}] \end{bmatrix}. \tag{5}$$

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

We construct the AMG preconditioner for $[M]$ in two ways. The first option is for the block M as a whole, denoted by $[M_0]$. Alternatively, ordering the degrees of freedom (DOFs) of the displacements first in x-direction and then in y-direction reveals a block two-by-two structure of the matrix M itself. Then the block diagonal of M is a good choice to construct AMG. For an explanation, see for instance, [3]. The resulting preconditioned inner solver is denoted by $[M_1]$.

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-the-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 (4), preconditioned by (5), using $[M_0]$ for the solution of systems with M. The code is developed using *C++* except AGMG that is available in FORTRAN. The systems $\mathcal{A}$, $[M_0]$ and $[\widetilde{S}]$ are solved with tolerance of 10^{-7} and 10^{-1} respectively. We note that solving the same system in 3D requires minor adjustment in the preconditioner and handling the boundary values. The rest is automatically taken care of through a template parameter stating the dimension of the problem.

Differential Equations Analysis Library (Deal.II) is used as the main finite element software toolbox. It is a general purpose object-oriented finite element library suitable for solving systems of partial differential equations. Deal.II is publicly available under an Open Source license. For more details on Deal.II, see [7].

Deal.II expands its functionality by providing interfaces to other packages such as Trilinos, PETSc [6], METIS [15] and others. Each package is added and used via a wrapper class in Deal.II. Data movement between Deal.II and other packages can be avoided by using the proper data structure provided by the Deal.II wrappers. For our purpose, we use the Deal.II wrapper for Trilinos.

As precondtioners for the inner solvers for M and $\widetilde{S}$ we use an AMG preconditioner, provided by Trilinos, AGMG and PARALUTION, see below.

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 [14]. Deal.II configures and uses the packages from Trilinos through Deal.II wrappers. In this study we use *Epetra* for sparse matrix and vector storage, *Teuchos* to provide parameters of solver and preconditioner, *ML* for multigrid preconditioning, *AZTEC* for the iterative solver (GMRES). Note that all the aforementioned packages are accessed through Deal.II wrappers.

The AMG from Trilinos is configured using Chebyshev smoother with two pre- and post-smoothing steps, uncoupled aggregation with threshold of 0.02 and one multigrid cycle. We refer to [7] for a detailed description of the settings.

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 goal of this project is to provide a portable library for iterative sparse methods on state-of-the-art hardware. The library contains a solid collection of various methods for solving sparse linear systems. All solvers and preconditioners are based on matrix and vector objects.

PARALUTION separates its objects from actual hardware specification. The objects are initially allocated on the host (CPU). Then every object can be moved to a selected accelerator by a simple move-to-accelerator function. The execution is based on run-time type information (RTTI) which allows the user to select where and how to perform the operations at run time. This is in contrast to template-based libraries that need this information at compilation time.

The philosophy of the library is to abstract the hardware-specific functions and routines from the actual program which describes the algorithm. This abstraction layer of the hardware specific routines is the core of PARALUTION's design, it is built to explore fine-grained level of parallelism suited for multi/many-core devices. In this way PARALUTION differs from most of the available parallel sparse libraries which are mainly based on domain decomposition techniques. Thus, the design of the iterative solvers and the preconditioners are very different. Another cornerstone of PARALUTION is the native support of accelerators - the memory allocation, transfers and specific hardware functions are handled internally in the library. The library provides OpenMP (Host, Xeon Phi/MIC), OpenCL (NVIDIA, AMD GPUs), CUDA (NVIDIA GPUs) backends. For more details we refer to [18].

A plug-in to Deal.II is provided which is not a direct wrapper as for Trilinos, but exports and imports data from Deal.II to PARALUTION. To solve the linear problem, we use the preconditioner $[M_0]$ and own implementation of AMG. The AMG is set to have the coarsest grid size as 2000 with smooth aggregation as the coarsening method, coupling strength is set to 0.001, the smoother is of multi-colored Gauss-Seidel type with relaxation parameter set to 1.3, [17]. Additionally, one pre-smoothing step, two post- smoothing steps and one multigrid cycle for preconditioning are performed. The AMG has to be constructed entirely on the CPU, while the execution can be performed on the CPU or on the GPU without any code modification.

AGMG implements an aggregation-based algebraic multigrid method [19]. 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, which implement the method described in [23], with further improvements from [20, 24].

AGMG's parallel performance is tested on up to 370000 cores. However, currently parallel implementation is available only with MPI and, therefore, in this study we compare it's serial performance with that of Trilinos AMG.

In this study AGMG uses double pairwise aggregation for the coarsening (with quality control as in [20, 24]), performed separately on the two components of the displacement vector. Furthermore, AGMG performs one forward and one backward Gauss-Seidel sweep for pre- and post-smoothing respectively and also a K-cycle [26], i.e., two Krylov accelerated iterations at each intermediate level. The main iterative solver in AGMG is the Generalized Conjugate Residual method, [13].

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-

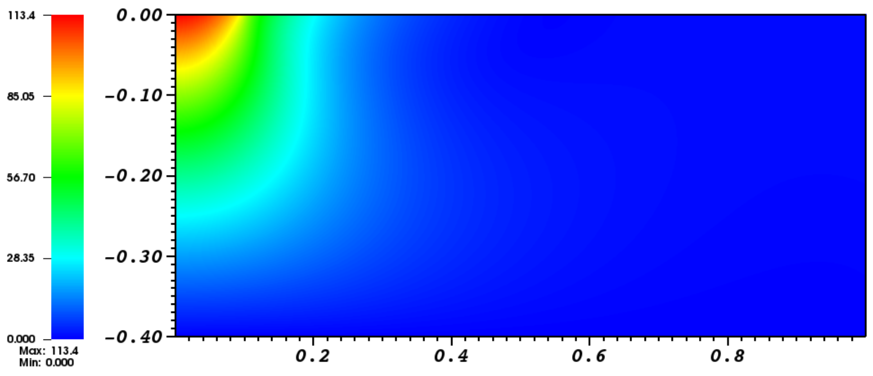


Fig. 2: GIA simulation - vertical displacements on the surface

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 [30]. 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 [8] 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

In Figure 2 we illustrate the GIA simulations. We show the vertical displacements in the domain, caused by a rectangular ice load as described in the test problem.

The performance results for ABAQUS, Deal.II, Trilinos and AGMG are obtained using (C1). The solver from PARALUTION (ver 0.6.1) is tested on CPU using both resources and on the GPU using (C2). Parallelism is exploited by the

Table 1: Comparison between Deal.II, PARALUTION and ABAQUS on (C1)

No. of	Deal.II + Trilinos			PARALUTION		ABAQUS		
Threads	DOFs	Setup	Solve (2/3)	Setup	Solve (2/3)	DOFs	Setup	Solve
1		3.43	69.9 (46.6)	14.5	51.6 (34.4)		7.44	59
4	1 479 043	3.04	47.3 (31.5)	9.76	20.9 (13.9)	986 626	7.49	33
8		3.00	42.9 (28.6)	8.76	22.4 (14.9)		7.51	28
1		15.4	317 (211)	59.4	220 (147)		29.72	269
4	5 907 203	14.2	210 (140)	40.4	92.7 (61.8)	3 939 330	29.93	145
8		14.1	297 (198)	36.2	92.5 (61.7)		29.94	122

built-in functionality of the packages to use OpenMP. The maximum number of threads, which is the number of cores without hyper-threading is set to twelve on (C1) and four on (C2).

From (3) we observe that while enabling to solve the models with fully incompressible materials, we obtain a system of equation that is about 30% larger than what is solved with ABAQUS. The problem sizes are shown in Table 1.

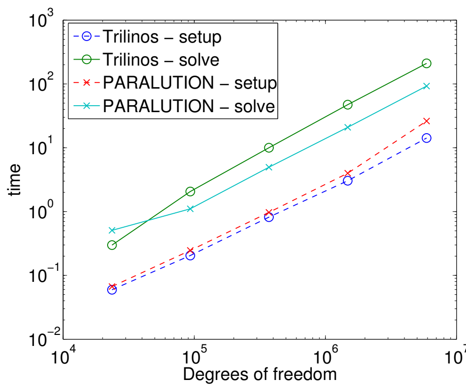
It is evident from Table 1 that both solution techniques scale nearly exactly linearly with the problem size. However, for the two- dimensional problem at hand, the direct solver from ABAQUS is somewhat faster than the preconditioned iterative solver, implemented in Trilinos. More detailed information, including iteration counts, is given in [12], confirming experimentally the optimality of the iterative solver.

Figure 3a shows the used time of the iterative solver using Trilinos and PARALUTION on four threads. While PARALUTION is faster than Trilinos in computing the solution, it takes more CPU time in the setup phase. We note that both Trilinos and PARALUTION do not scale to more than four threads. For more details see [12]. We also note that PARALUTION performs faster than the direct solver from ABAQUS.

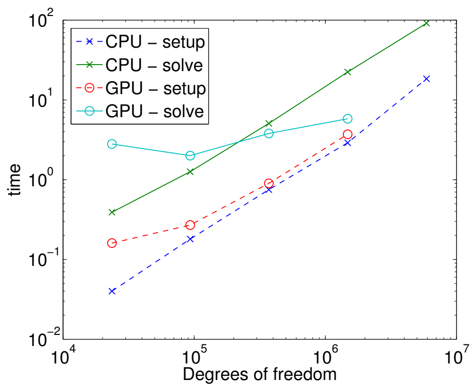
From experiments, not included here, performed using *valgrind*, cf. [2], one can see that the lack of scaling is due to the matrix- vector multiplications in AMG. This issue is not solvable within the shared memory programming model and using OpenMP for further developing the model and/or extending the computational domain will be a bottle neck. We expect to see better scaling by changing the programming model to MPI.

As mentioned above PARALUTION is tested on CPU using both (C1) and (C2). The results of these experiments are used as reference to compare the results of PARALUTION on the GPU with the results of ABAQUS and Trilinos. The results are presented in Figure 3b. 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 outperforms the CPU with up to four times speedup. Solving the largest problem size on the GPU leads to insufficient memory error.

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.



(a) Comparison with Trilinos



(b) CPU vs GPU

Fig. 3: Performance comparisons: PARALUTION

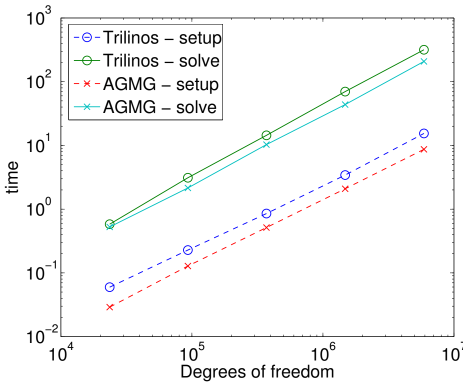


Fig. 4: Comparison between Trilinos-AMG and AGMG

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.

To optimize the solution process, we consider changing 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 Figure 4 show that AGMG has better computational efficiency. 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.

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. In the current setting PARALUTION proves to show the best performance results.

However, 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. Valgrind, `http://www.valgrind.org`.
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. 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.
7. W. Bangerth, G. Kanschat, and R. Hartmann. `deal.II` differential equations analysis library, `http://www.dealii.org`.
8. 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.
9. M. Benzi, G. H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137, 5 2005.
10. D. Braess. *Finite elements.* Cambridge University Press, Cambridge, third edition, 2007. Theory, fast solvers, and applications in elasticity theory.
11. 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.

12. A. Dorostkar, D. Lukarski, B. Lund, M. Neytcheva, Y. Notay, and P. Schmidt. Parallel performance study of block-preconditioned iterative methods on multicore computer systems. Technical Report 2014-007, Department of Information Technology, Uppsala University, Mar. 2014.
13. S. C. Eisenstat, H. C. Elman, and M. H. Schultz. Variational iterative methods for nonsymmetric systems of linear equations. 20:345–357, 1983.
14. M. A. Heroux and J. M. Willenbring. Trilinos Users Guide `http://trilinos.sandia.gov`. Technical Report SAND2003-2952, Sandia National Lab., 2003.
15. G. Karypis and V. Kumar. MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0. `http://www.cs.umn.edu/~metis`, 2009.
16. J. Kraus. Additive Schur complement approximation and application to multilevel preconditioning. *SIAM J. Sci. Comput.*, 34(6):A2872–A2895, 2012.
17. 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.
18. D. Lurkarski. Paralution project, `http://www.paralution.com`.
19. Yvan Notay. AGMG software and documentation, `http://homepages.ulb.ac.be/~ynotay/AGMG`.
20. A. Napov and Y. Notay. An algebraic multigrid method with guaranteed convergence rate. *SIAM J. Sci. Comput.*, 34(2):A1079–A1109, 2012.
21. M. Neytcheva. On element-by-element Schur complement approximations. *Linear Algebra Appl.*, 434(11):2308–2324, 2011.
22. 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.
23. Y. Notay. An aggregation-based algebraic multigrid method. *Electron. Trans. Numer. Anal.*, 37:123–146, 2010.
24. Y. Notay. Aggregation-based algebraic multigrid for convection-diffusion equations. *SIAM J. Sci. Comput.*, 34(4):A2288–A2316, 2012.
25. Y. Notay. A new analysis of block preconditioners for saddle point problems. *SIAM J. Matrix Anal. Appl.*, 35:143–173, 2014.
26. Y. Notay and P. S. Vassilevski. Recursive Krylov-based multigrid cycles. *Numer. Lin. Alg. Appl.*, 15:473–487, 2008.
27. Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14(2):461–469, 1993.
28. Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 7(3):856–869, 1986.
29. P. Wu. Viscoelastic versus viscous deformation and the advection of pre-stress. *Geophysical Journal International*, 108(1):136–142, 1992.
30. 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.