Computational Geosciences manuscript No.
(will be inserted by the editor)

A parallel block preconditioner for large scale poroelasticity with highly heterogeneous material parameters

Joachim Berdal Haga · Harald Osnes · Hans Petter Langtangen

Received: date / Accepted: date

Abstract Large-scale simulations of coupled flow in deformable porous media require iterative methods for solving the systems of linear algebraic equations. Construction of efficient iterative methods is particularly challenging in problems with large jumps in material properties, which is often the case in realistic geological applications, such as basin evolution at regional scales. The success of iterative methods for such problems depends strongly on finding effective preconditioners with good parallel scaling properties, which is the topic of the present paper.

We present a parallel preconditioner for Biot's equations of coupled elasticity and fluid flow in porous media. The preconditioner is based on an approximation of the exact inverse of the two-by-two block system arising from a finite element discretisation. The approximation relies on a highly scalable approximation of the global Schur complement of the coefficient matrix, combined with generally available state-of-the-art multilevel preconditioners for the individual blocks. This preconditioner is shown to be robust on problems with highly heterogeneous material parameters. We investigate the weak and strong parallel scaling of this preconditioner on up to 512 processors, and demonstrate its ability on a realistic basin-scale problem in poroelasticity with over 8 million tetrahedral elements.

J. B. Haga
Computational Geosciences, Simula Research Laboratory
PO Box 134, NO-1325 Lysaker, Norway
Tel.: +47 67 82 82 00
Fax: +47 67 82 82 01
E-mail: jobh@simula.no

H. Osnes
Department of Mathematics, University of Oslo
PO Box 1053 Blindern, NO-0316 Oslo, Norway

H. P. Langtangen
Center for Biomedical Computing, Simula Research Laboratory
PO Box 134, NO-1325 Lysaker, Norway

Keywords parallel computing · block preconditioning · poroelasticity · low-permeable media · finite elements

Mathematics Subject Classification (2000) 35Q72 · 35Q80 · 65F10 · 65Y05 · 65Y20 · 74F10 · 74L10 · 74S05

1 Introduction

Iterative methods have proven to be the most scalable approach for parallel solvers for algebraic systems of equations, such as those arising from discretisations of partial differential equations (PDEs). Nonetheless, the efficiency of iterative solvers is highly problem-dependent and sensitive to the parameters of the system. Biot's equations [4], describing the coupled poroelastic response of fluid-filled materials, have been shown to be a difficult problem for such solvers due to the extreme jumps that some of the material parameters exhibit in realistic problems. As a result, direct solvers are often employed in such situations. Direct solvers, however, suffer from suboptimal scaling in time [9] and in space [14]. Thus, for truly large-scale problems, such as realistic basin-scale models, efficient and robust iterative methods must be found.

In [18], the present authors demonstrated the efficacy of a preconditioner based on the exact block decomposition in the serial case, using the Schur complement of the 2×2 coefficient matrix. For the individual blocks, an algebraic multigrid (AMG) preconditioner is used. The AMG preconditioner has been shown to have good parallel scaling properties for up to thousands of processors [2, 21]. Given scalable preconditioners for the individual blocks, block preconditioners that work on the unmodified blocks of the coefficient matrix are relatively straightforward to parallelise. The Schur complement, however, requires special care. Elman *et al.* [10, 11] studied the parallel scaling of block preconditioners based on the Schur complement for the Navier-

Stokes problem. Simpler Schur complement block preconditioners have been employed successfully for Biot's equation [25, 27]. However, it remains to investigate the parallel efficiency of the more advanced block preconditioners from [18], particularly targeting large jumps in material parameters. This is exactly the topic of the present paper.

We perform extensive numerical investigations on model problems in two and three dimensions on a computer cluster using up to 512 processors to verify parallel scalability. Additionally, we perform tests on a realistic basin model exported from a commercial basin simulator. This model is too large to be solved by direct methods, and has so far proven intractable to standard iterative methods due to the strong contrasts in the material parameters (in particular, the permeability).

This paper is organised as follows. In Section 2 the governing equations of the poroelastic problem are presented, followed by a brief overview of their weak form and the approximation this leads to in the finite element method. An outline of block preconditioners is found in Section 3, along with the algorithms for construction of the distributed Schur complement approximation. Section 4 shows how the mathematical model is implemented in software, and details how the parallelism is achieved, while Section 5 reports the results of the numerical investigations including parallel scaling results. Finally, we give some concluding remarks in Section 6.

2 Mathematical model

The equations describing poroelastic flow and deformation are derived from the principles of conservation of fluid mass and the balance of forces on the porous matrix. The linear poroelastic equations can, in the small-strains regime, be expressed as

$$S\dot{p} - \nabla\cdot\Lambda\nabla p + \alpha\nabla\cdot\dot{u} = q, \tag{1}$$
$$\nabla(\lambda+\mu)\nabla\cdot u + \nabla\cdot\mu\nabla u - \alpha\nabla p = r. \tag{2}$$

Here, we subsume body forces such as gravitational forces into the right-hand side source terms q and r. The primary variables are p for the fluid pressure and u for the displacement of the porous medium. Furthermore, S and Λ are the fluid storage coefficient and the flow mobility respectively, α is the Biot-Willis fluid/solid coupling coefficient, and λ and μ are the Lamé elastic parameters.

The fluid (Darcy) velocity is often of particular interest in poroelastic calculations. It can be written

$$v_{\mathrm{D}} = -\Lambda\nabla p, \tag{3}$$

and represents the net macroscopic flux. For the displacement equation, the main secondary quantity of interest is the effective stress tensor,

$$\tilde{\sigma} = (\alpha p + p_{\mathrm{s}})I + \mu(\nabla u + (\nabla u)^{\mathrm{T}}), \tag{4}$$

which is written here using the solid pressure

$$p_{\mathrm{s}} = -\lambda\nabla\cdot u. \tag{5}$$

The model is closed by the addition of boundary and initial conditions,

$$p|_{\Gamma_p} = p_\Gamma, \quad u|_{\Gamma_u} = u_\Gamma, \quad p(t_0) = p_0, \quad u(t_0) = u_0 \tag{6}$$

where Γ_p and Γ_u are the parts of the boundary fluid pressure and solid displacement are specified, and t_0 is the initial time.

2.1 Weak time-discrete form.

We employ a first-order backward finite difference method in time, which leads to the time-discrete form of Eq. 1

$$Sp - \Delta t\nabla\cdot\Lambda\nabla p + \nabla\cdot u = q\Delta t + S\hat{p} + \nabla\cdot\hat{u}. \tag{7}$$

Hatted variables ($\hat{p}$, $\hat{u}$) indicate values from the previous time step, while unmarked variables are taken to be at the current time step.

Next, we rewrite Eq. 2 and 7 in weak form, using integration by parts to eliminate second derivatives. We define the following (bi-)linear forms on the domain Ω with boundary Γ,

$$\begin{aligned} a^{\mathrm{p}}(\phi^{\mathrm{p}},p) &= -\int_\Omega S\phi^{\mathrm{p}}p + \Delta t\nabla\phi^{\mathrm{p}}\cdot\Lambda\nabla p\,\mathrm{d}\Omega,\\ l^{\mathrm{p}}(\phi^{\mathrm{p}}) &= -\int_\Omega (q\Delta t + S\hat{p} + \nabla\cdot\hat{u})\phi^{\mathrm{p}}\,\mathrm{d}\Omega\\ &\qquad + \int_\Gamma \phi^{\mathrm{p}} f_n\Delta t\,\mathrm{d}\Gamma, \end{aligned} \tag{8}$$

and

$$\begin{aligned} a^{\mathrm{u}}(\phi^{\mathrm{u}},u) &= \int_\Omega [(\lambda+\mu)(\nabla\cdot\phi^{\mathrm{u}})(\nabla\cdot u)\\ &\qquad + \mu\nabla\phi^{\mathrm{u}} : \nabla u]\,\mathrm{d}\Omega,\\ b(\phi^{\mathrm{u}},p) &= -\int_\Omega \alpha p\nabla\cdot\phi^{\mathrm{u}}\,\mathrm{d}\Omega,\\ l^{\mathrm{u}}(\phi^{\mathrm{u}}) &= -\int_\Omega \phi^{\mathrm{u}}\cdot r\,\mathrm{d}\Omega + \int_\Gamma \phi^{\mathrm{u}}\cdot t_n\,\mathrm{d}\Gamma. \end{aligned} \tag{9}$$

The problem then becomes: Find $p \in V_{\mathrm{p}}$ and $u \in V_{\mathrm{u}}$ that satisfy the following relations:

$$a^{\mathrm{p}}(\phi^{\mathrm{p}},p) + b(u,\phi^{\mathrm{p}}) = l^{\mathrm{p}}(\phi^{\mathrm{p}}) \qquad \forall\phi^{\mathrm{p}} \in V_{\mathrm{p}}, \tag{10}$$
$$a^{\mathrm{u}}(\phi^{\mathrm{u}},u) + b(\phi^{\mathrm{u}},p) = l^{\mathrm{u}}(\phi^{\mathrm{u}}) \qquad \forall\phi^{\mathrm{u}} \in V_{\mathrm{u}}. \tag{11}$$

The normal flux $f_n = v_{\mathrm{D}}\cdot n$ and the normal stress t_n on the boundary Γ appear in these equations as natural boundary conditions. The natural spaces for the continuous problem are $V_{\mathrm{p}} = H^1$ for the pressure and $V_{\mathrm{u}} = H^1$ for the displacement.

The discrete approximation follows from solving the equations for the weak form in finite-dimensional subspaces of the continuous spaces: Given finite element basis functions $\phi_i^{\mathrm{u}} \in V_{\mathrm{u}h} \subset V_{\mathrm{u}}$ spanning the discrete displacement space, and basis functions $\phi_i^{\mathrm{p}} \in V_{\mathrm{p}_h} \subset V_{\mathrm{p}}$ spanning the discrete fluid pressure space, the unknown functions are approximated as $u \approx \sum_i \mathrm{u}_i \phi_i^{\mathrm{u}}$ and $p \approx \sum_i \mathrm{p}_i \phi_i^{\mathrm{p}}$. The task is to find the vectors u and p that makes these approximations as good as possible (in some sense); this is done by the finite element method.

2.2 The algebraic system

The algebraic system that results from discretising Eqs. 10–11 is on the form

$$\mathscr{A}\mathrm{x} = \mathrm{b}, \tag{12}$$

where $\mathscr{A}$ is the coefficient matrix derived from the left-hand sides of Eqs. 10 and 11, b is the load vector arising from the right-hand sides, and x is the unknown solution vector. As this is a coupled system of two equations, the coefficient matrix can be viewed as a 2×2 block matrix. The signs of the equations have been chosen so as to make this a symmetric indefinite problem, which we write blockwise as

$$\mathscr{A} = \begin{bmatrix} \mathrm{A} & \mathrm{B} \\ \mathrm{B}^{\mathsf{T}} & \mathrm{C} \end{bmatrix}, \quad \mathrm{x} = \begin{bmatrix} \mathrm{u} \\ \mathrm{p} \end{bmatrix}, \quad \mathrm{b} = \begin{bmatrix} \mathrm{l}^{\mathrm{u}} \\ \mathrm{l}^{\mathrm{p}} \end{bmatrix}, \tag{13}$$

with A symmetric positive definite and C symmetric negative definite. Using the finite element basis functions ϕ_i^{u} and ϕ_i^{p} introduced above, the entries of each block are

$$A_{ij} = a^{\mathrm{u}}(\phi_i^{\mathrm{u}}, \phi_j^{\mathrm{u}}), \tag{14}$$
$$B_{ij} = b(\phi_i^{\mathrm{u}}, \phi_j^{\mathrm{p}}), \tag{15}$$
$$C_{ij} = a^{\mathrm{p}}(\phi_i^{\mathrm{p}}, \phi_j^{\mathrm{p}}). \tag{16}$$

The load vector is defined in a similar way, with $\mathrm{l}_i^{\mathrm{u}} = l^{\mathrm{u}}(\phi_i^{\mathrm{u}})$ and $\mathrm{l}_i^{\mathrm{p}} = l^{\mathrm{p}}(\phi_i^{\mathrm{p}})$.

The solution of algebraic systems of equations resulting from finite element discretisations, like Eq. 12, generally shows poor convergence properties when using iterative solvers. To overcome this, suitable preconditioning is crucial.

3 Block preconditioning methods

We seek a preconditioner that exploits the block structure of Eq. 13. The simplest example is perhaps the block Jacobi preconditioner,

$$\mathscr{P}_{\mathrm{J}}^{-1} = \begin{bmatrix} \mathrm{A}^{-1} & 0 \\ 0 & \mathrm{C}^{-1} \end{bmatrix}. \tag{17}$$

The single-block inverses are too expensive to compute exactly, and will be approximated by single-block preconditioners. In the following, we mark such approximations with a tilde: $\tilde{\mathrm{A}}^{-1}$ and $\tilde{\mathrm{C}}^{-1}$. By further defining the lower-triangular coupling matrix as

```
v  ← $\tilde{\mathrm{A}}^{-1}\mathrm{w}$
q′ ← $\mathrm{B}^{\mathsf{T}}\mathrm{v} - \mathrm{r}$
q  ← $\tilde{\mathrm{S}}^{-1}\mathrm{q}'$
if symmetric then
    v′ ← $\mathrm{B}\mathrm{q}$
    v  ← $\mathrm{v} - \tilde{\mathrm{A}}^{-1}\mathrm{v}'$
end
```

Algorithm 1 Application of the block Gauß-Seidel preconditioners to a block vector: $[\mathrm{v}\ \mathrm{q}]^{\mathsf{T}} \leftarrow \mathscr{P}_{\mathrm{g(S)GS}}^{-1}[\mathrm{w}\ \mathrm{r}]^{\mathsf{T}}$

$$\mathscr{G} = \begin{bmatrix} \mathrm{I} & 0 \\ -\mathrm{B}^{\mathsf{T}}\tilde{\mathrm{A}}^{-1} & \mathrm{I} \end{bmatrix}, \tag{18}$$

we can express the block Gauß-Seidel preconditioner as the matrix product $\mathscr{P}_{\mathrm{GS}}^{-1} = \mathscr{P}_{\mathrm{J}}^{-1}\mathscr{G}$, and its symmetric variation as $\mathscr{P}_{\mathrm{SGS}}^{-1} = \mathscr{G}^{\mathsf{T}}\mathscr{P}_{\mathrm{J}}^{-1}\mathscr{G}$.

The Schur complement of the block coefficient matrix $\mathscr{A}$ is defined as $\mathrm{S} = \mathrm{B}^{\mathsf{T}}\mathrm{A}^{-1}\mathrm{B} - \mathrm{C}$. It is symmetric and positive definite. Following [25] we can write the Generalised Jacobi preconditioner as[1]

$$\mathscr{P}_{\mathrm{gJ}}^{-1} = \begin{bmatrix} \tilde{\mathrm{A}}^{-1} & 0 \\ 0 & -\tilde{\mathrm{S}}^{-1} \end{bmatrix}. \tag{19}$$

The corresponding Generalised Symmetric Gauß-Seidel preconditioner, which we define by analogy with regular Gauß-Seidel as

$$\mathscr{P}_{\mathrm{gSGS}}^{-1} = \mathscr{G}^{\mathsf{T}}\mathscr{P}_{\mathrm{gJ}}^{-1}\mathscr{G}, \tag{20}$$

is in fact an exact inverse of $\mathscr{A}$, if the single-block inverses are exact.

An inexact version of Eq. 20, along with its nonsymmetric cousin $\mathscr{P}_{\mathrm{gGS}}^{-1}$, were shown in [18] to be very robust preconditioners for Biot's equations on a problem with extreme contrasts in the material parameters. Algorithm 1 shows the necessary steps to implement this preconditioner. Each assignment requires one global single-block operation, i.e., the processor-local operation followed by an update of the foreign nodes. The application of the $(1,1)$ preconditioner $\tilde{\mathrm{A}}^{-1}$ to a vector is normally by far the most expensive step of this algorithm, and the symmetric variant is therefore about twice as expensive as nonsymmetric generalised Gauß-Seidel. However, the symmetric variant provides the opportunity to use symmetric solvers. Such solvers are often more efficient and/or robust than nonsymmetric solvers, which may justify the increased cost. In the remainder of this paper, we focus on the symmetric variant.

[1] In the reference, a scalar multiplier α is used for the $(2,2)$ block; here, $\alpha = -1$.

3.1 The distributed Schur complement approximation

First a small note on terminology: The word "node" is traditionally used both in the parallel computing context and in the PDE context, with different meanings. In the following, we reserve *node* to mean a spatially located unknown in the finite element method, while *computational node* is used for a single computer in a cluster. To further clarify the computational hierarchy, *processor* is used interchangeably with *core* to mean a computing unit that runs a single *process*. One or more processors make up a *die*, and one or more dies make up a computational node. Thus, a typical computational node may have two quad-core dies with a total of eight processors.

We shall come back later to the subject of parallel partitioning, but to simplify the discussion we assume the following properties of the partitioning:

(i) Each node is owned exclusively by one processor. This node is then *interior* to the owning processor. The node may also be present on neighbouring processors, where it is a *foreign* node. We also use the term *border node* for those nodes which are interior, but share an element with (and hence couple to, in the coefficient matrix) a foreign node.
(ii) Every interior node has full cover on the owning processor, i.e., all elements that contain the node are present in the local finite element assembly.

While forming the exact Schur complement is infeasible, an approximation that was shown in [18] to perform well for high-contrasting material parameters is

$$\hat{\mathrm{S}} = \mathrm{Diag}(\mathrm{B}^\mathsf{T}(\mathrm{Diag}\,\mathrm{A})^{-1}\mathrm{B}) - \mathrm{C}, \tag{21}$$

where Diag is an operator that creates a matrix of equal dimension, containing only the diagonal elements.This approximation can be calculated in parallel with overhead equal to that of a single matrix-vector product. To understand how, we look briefly at the behaviour of a parallel matrix-matrix product.

In Fig. 1a, we have sketched the structure of a processor-local part of the global coefficient matrix. The salient part is this: All rows *and* columns involving interior nodes are globally correct and complete. Hence, the diagonal of the result of a local matrix-matrix product (shown in Fig. 1b) is correct for all entries associated with interior unknowns. Only the entries associated with foreign unknowns are incorrect. This is not a problem, since a matrix-vector product is always followed by an update of the foreign nodes. However, Eq. 21 involves a triple matrix product. To ensure that the diagonal of this triple-product is correct for all interior entries, we do need to have globally correct entries for the whole of Diag A; otherwise the product $(\mathrm{Diag}\,\mathrm{A})^{-1}\mathrm{B}$ will *not* have the structure of Fig. 1a (the interior columns of the foreign rows will be wrong). The complete algorithm to create the distributed Schur complement of Eq. 21 is presented in Algorithm 2. The only interprocess communication in this algorithm takes place in step 3, where the diagonal is updated.

```
1  parallel for each processor $P$ do
2      $\mathrm{a}^P \leftarrow \mathrm{diag}(\mathrm{A}^P)$          ▷ create column vector from diagonal
3      $\mathrm{a}^P \leftarrow \mathrm{update}(\mathrm{a}^P)$        ▷ fetch foreign nodes from neighbours
4      $\hat{\mathrm{S}}^P \leftarrow -\mathrm{C}^P$
5      for each interior row $i$ do
6          for each nonzero index $k$ in the matrix row $\mathrm{B}^P_i$ do
7              $\hat{S}^P_{ii} \leftarrow \hat{S}^P_{ii} + (B^P_{ik})^2 (a^P_k)^{-1}$
8          end
9      end
10 end
```

Algorithm 2 Construction of the distributed Schur complement approximation $\hat{\mathrm{S}} \leftarrow \mathrm{Diag}(\mathrm{B}^\mathsf{T}\,\mathrm{Diag}(\mathrm{A})^{-1}\mathrm{B}) - \mathrm{C}$

3.2 The single-block preconditioners

The block preconditioners in the previous section depend on the availability of efficient single-block preconditioners $\tilde{\mathrm{A}}^{-1}$ and $\tilde{\mathrm{S}}^{-1}$. We restrict our attention to preconditioners which are efficient on massively parallel computers. This rules out incomplete and approximate direct solvers such as the otherwise excellent ILU methods.

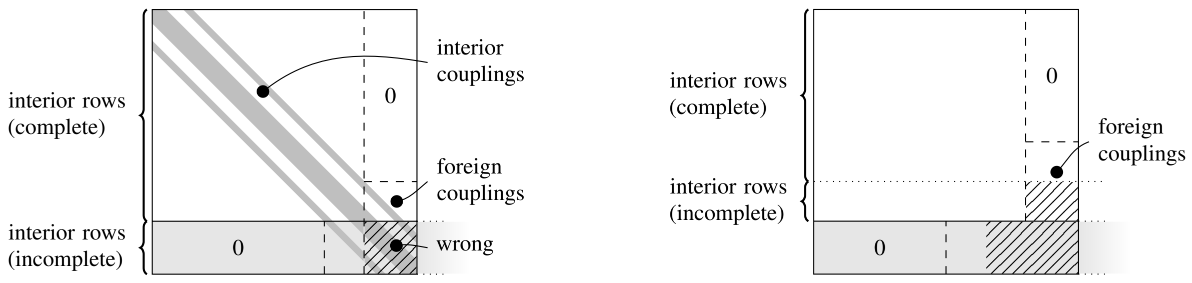
Adams [1] found AMG to behave very well on problems of elastic deformation, even in the presence of strong material discontinuities. In particular, the smoothed aggregation (SA) method [28, 5] was considered to be the overall superior AMG method for elasticity problems. The present authors likewise found SA to be a nearly optimal preconditioner for the discontinuous Poisson pressure problem (see [16]), and to perform well on the similarly structured Schur complement approximation found in Eq. 21 (see [18]).

In the light of these earlier results, and the fact that AMG has been shown to scale very well in parallel, to at least thousands of processors [29, 2, 21, 7], we have chosen to use SA to precondition both the decoupled displacement equation (A) and the Schur complement approximation ($\hat{\mathrm{S}}$).

4 Software framework

We have implemented the finite element discretisation and assembly, as well as the the block preconditioners and iterative solvers, using the Diffpack C++ framework [23, 8], with extensive modifications in key areas: parallel block systems, parallel partitioning, and mixed finite elements (serial and parallel).

A domain decomposition approach is used for the finite element assembly stage, where each processor works on a



(a) Processor-local matrix structure **(b)** Structure after local matrix-matrix product

Fig. 1 In a processor-local matrix-matrix product, the interior rows with nonzero foreign coupling terms are wrong; only the interior-row part of the diagonal can be trusted

subset of the global grid. In the linear algebra stage, message passing (using Message Passing Interface, MPI) is used to formulate globally consistent operations for matrix-vector products, vector inner products, and so on. The main tradeoff in this approach is in choosing how to partition the grid between processors. Our choice is mainly motivated by the ease of interfacing with external parallel libraries. Hence, we employ a model wherein each node is owned exclusively by one processor. If we further require that every such interior node is provided with full cover on the owning processor, we gain the desirable property that the matrix rows (and, incidentally, the matrix columns) associated with this node are complete.

The partitioning procedure proceeds in two stages:

(i) Balance the nodes between the processors, while minimising the number of intersected elements,
(ii) Add foreign nodes to each partition until full cover is provided for each interior node.

A hypergraph partitioner, with each hyperedge containing the nodes of one element, should be the ideal way to achieve (i). However, all partitioners use heuristics to achieve acceptable performance, and a graph or even a geometric partitioner may perform equally well on a given problem. We interface with the PHG hypergraph partitioner and a geometric partitioner from Zoltan [6], and with the METIS and ParMETIS [22] graph partitioners.

The single-block AMG preconditioners are from the ML package for Smoothed Aggregation [13], which is part of the Trilinos project [19]. The ML interface requires the input of complete local rows for the global coefficient matrix, a task which is greatly aided by the properties of the partitioning listed above.

In addition to the above, we have developed software to import finite element grids, fields and material parameters from Petromod [24], which is one of the leading basin simulation software packages in the oil and gas industry. This allows us to use realistic geometries, initial conditions and material parameters in our tests.

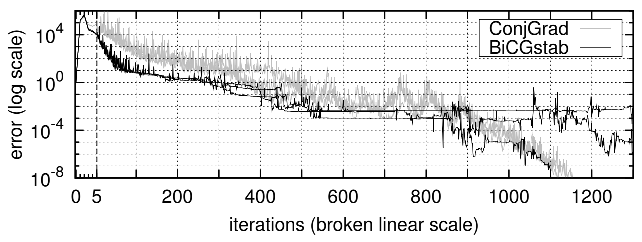
5 Numerical experiments

5.1 Convergence criterion

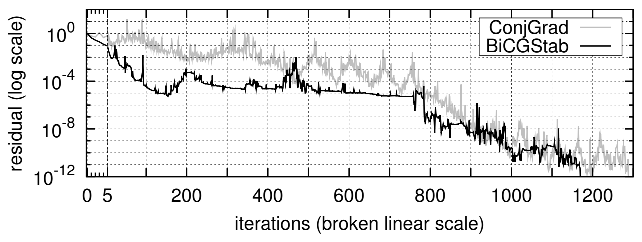
When using iterative methods for solving algebraic systems of equations, a suitable convergence criterion must be introduced. Different criteria are possible, but the "ideal" criterion which measures the error is generally not available unless the solution is known in advance. More commonly, a convergence criterion based on the residual $\mathrm{r}_k = \mathrm{b} - \mathscr{A}\mathrm{x}_k$ (in the k-th iteration) is used. However, such a criterion may be misleading when $\mathscr{A}$ is very ill-conditioned [16], such as with severe jumps in the material parameters. We are less interested in the solution itself than in the convergence properties of the solver, and thus we may exploit a convenient property of iterative solvers: the convergence is independent of the right-hand side b as long as the initial guess (and hence the initial residual) contains all eigenvectors of $\mathscr{A}$ [15, ch. 3.4].

Hence, we solve the modified problem $\mathscr{A}\mathrm{x} = 0$ with a randomised initial solution vector x_0, instead of the original $\mathscr{A}\mathrm{x} = \mathrm{b}$. With a zero right-hand side, the error is simply $\mathrm{e}_k = \mathrm{x}_k$. We also note that due to this testing procedure, the exact value of any boundary condition is irrelevant, since these values go into the b vector. The only relevant information in this case is *where* essential boundary conditions are used, since the presence of an essential boundary condition at a node is reflected by a modification to the associated row(s) and column(s) of $\mathscr{A}$.

In the description that follows of the numerical experiments, we use the term *error criterion* (with an associated tolerance ε, implying $\|\mathrm{e}_k\| \leq \varepsilon$) for the convergence criterion described above. However, in order to measure more narrowly the efficiency of the parallel implementation itself, it is sometimes advantageous to measure the time to complete a fixed number of iterations (convergence criterion $k = k_{\max}$); we shall refer to this as the *iteration criterion*.



(a) Error with three random initial vectors and zero right-hand side



(b) Residual with zero initial vector and nonzero right-hand side

Fig. 2 Iterations to reach a given reduction in error norm (a) or residual norm (b) on the realistic basin case (III)

5.2 Choice of iterative solver

The coefficient matrix $\mathscr{A}$ is symmetric indefinite. Since the preconditioner is symmetric, the preconditioned coefficient matrix $\mathscr{P}_{gSGS}^{-1}\mathscr{A}$ is also symmetric. With such a system of equations, one would normally prefer an iterative solver which can be used with indefinite systems. It is known that the Conjugate Gradient method can perform well even when there are a few negative eigenvalues [12]; however, the present problem has a large number of negative eigenvalues. The convergence of the Conjugate Gradient method does not generally follow from the theory on such problems. Although convergence is not guaranteed, we have nevertheless seen good convergence in practice with this method, as reported in [18]. Fig. 2 compares the Stabilised Bi-Conjugate Gradient (BiCGStab) method, which is designed for general problems, with the Conjugate Gradient (ConjGrad) method on the realistic basin model described below. In Fig. 2a, three experiments are shown for each of BiCGStab and Conjugate Gradients, using random initial vectors with error 10^0 (jumping to $\sim 10^5$ in the first iteration). The residuals, with a nonzero right-hand side, are shown in Fig. 2b for comparison. Notably, the residual does not exhibit the initial large jump that is seen in the error, and hence the residual is many orders of magnitute below the error.

This is our most difficult test case for the iterative solver. It appears that the Conjugate Gradient method performs just as well as the BiCGStab method, and furthermore that it has much smaller sensitivity to the (random) initial solution vector. Consequently, we use the Conjugate Gradient method in our experiments.

We should note, lest the results in Fig. 2a make our chosen preconditioner look bad, that this test problem is one which we previously have not been able to solve at all using standard iterative solvers and preconditioners. Thus, even a preconditioner which requires 500+ iterations is a significant step forward.

5.3 Scaling

Before looking further into the experimental data, it may be advantageous to have a rough idea what to expect from the results. We can identify five main causes of imperfect parallel scaling:

(i) increased local problem size due to duplicated nodes and imbalance,
(ii) point-to-point (neighbour) communication,
(iii) collective (global) communication,
(iv) increasing number of iterations in the iterative solver for a given accuracy, and
(v) slowdown due to congestion of shared resources (within or between computational nodes).

In general, (iv) depends on the chosen preconditioner/iterative solver combination, and can be controlled by using an iteration criterion instead of an error criterion, while (v) is hardware dependent and must usually be discovered through testing.

We investigate the parallel scalability in two different scaling paradigms. In the weak scaling paradigm, the number of nodes (or work) per processor is fixed. Causes (i) and (ii) should then approach constant overhead (after an initial ramp-up), while the cost of cause (iii) is of order $\log P$ on P processors [26].

We also investigate strong scaling. In this paradigm, the total problem size is fixed as the number of processors increases. Strong scaling has received less attention than weak scaling in the literature, but in practical applications the need to solve a large problem *as fast as possible* is perhaps more common than the need to solve a problem that is *as large as possible* in a given time. In this case, the absolute overhead due to causes (i) and (ii) decreases with increasing P. It does not, however, decrease as fast as the amount of useful work per processor. Hence, the relative overhead increases.

We define the *efficiency* as the ratio of the perfect-scaling runtime to the actual runtime, or, equivalently, the number of unknowns processed per unit aggregate time. In D spatial dimensions, the walltime T and efficiency E in the weak scaling paradigm, with N nodes on each of P processors,

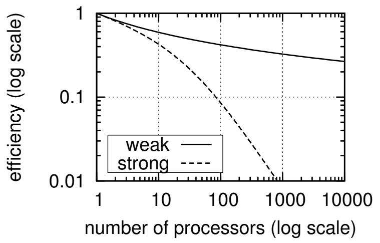


Fig. 3 Typical shape of the efficiency curves for strong and weak scaling.

can be modelled as

$$T(1) = cN, \tag{22}$$

$$T_\mathrm{w}(P) = T(1) + acN^{\frac{D-1}{D}} + bc\log P, \tag{23}$$

$$E_\mathrm{w}(P) = \frac{T(1)}{T_\mathrm{w}(P)} = \left[1 + aN^{\frac{-1}{D}} + bN^{-1}\log P\right]^{-1}, \tag{24}$$

where a, b and c are constant factors depending on the specifics of the problem and of the platform. In the strong scaling paradigm, with $N/P \geq 1$ nodes per processor, these can be modelled as

$$T_\mathrm{s}(P) = T(1)/P + ac(N/P)^{\frac{D-1}{D}} + bc\log P, \tag{25}$$

$$E_\mathrm{s}(P) = \frac{T(1)}{PT_\mathrm{s}(P)} = \left[1 + a(P/N)^{\frac{1}{D}} + b(P/N)\log P\right]^{-1}. \tag{26}$$

This assumes perfectly scalable hardware (no interconnect saturation, etc), and a fixed number of iterations of the iterative solver. The constant a comes from (i)–(ii) above, and b comes from (iii); the quantity $N^{\frac{D-1}{D}}$ is proportional to the number of nodes intersected by a slice through the domain.

Disregarding the exact value of the various constants, we expect to see efficiency curves of the general shapes shown in Fig. 3: A nearly flat, slightly upturned curve on the log-log plot in weak scaling, and a strongly downturned curve in strong scaling.

Comparing this with numerical tests on various hardware is instructive. Two such are shown in Figs. 4a and 4b for weak scaling, at a fixed number of iterations. On the Cray cluster,[2] Fig. 4a, the scaling appears roughly as in the simple model illustrated in Fig. 3, except a small ($\sim$ 10%) drop when utilising all four processors on a single computational node instead of one processor on each of four computational nodes. This drop must be caused by contention of a shared resource internally to a computational node, most likely exhaustion of the memory bandwidth. Compare this with a commodity cluster[3] (Fig. 4b) when utilising multiple cores per computational node: Four cores on a single computational node, 36% drop in efficiency; eight cores, 64% drop! Clearly, this hardware is not very efficient for such a data intensive workload.

[2] The *hexagon* Cray XT4 cluster located in Bergen, Norway [20].

[3] The *bigblue* computer cluster at Simula Research Laboratory [3].

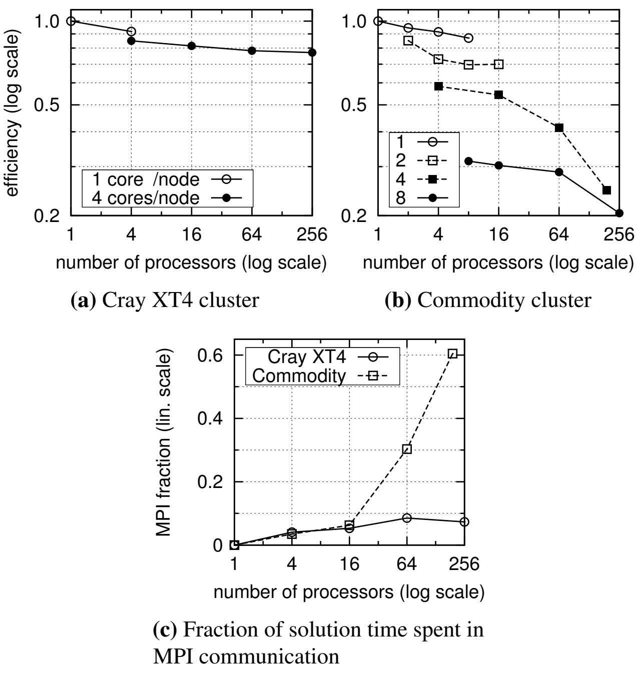


Fig. 4 Weak scaling efficiency on two different hardware platforms

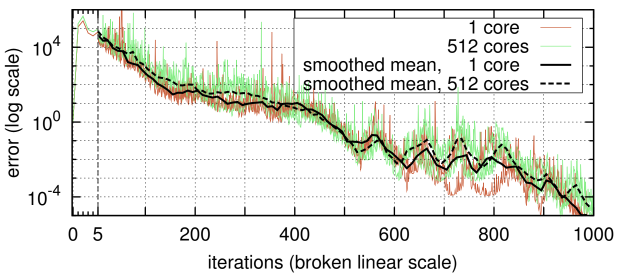
Furthermore, we see a rapid worsening of the efficiency on the commodity cluster when more than a few tens of processors are involved. This does not match our expectation from the weak scaling curve of Fig. 3, and we therefore suspect it is caused by congestion of the interconnect between computational nodes. Measuring the time spent in MPI communications shows this to be the major cause, as shown in Fig. 4c. On the commodity cluster (which uses a GHz Ethernet interconnect), most of the time is indeed spent doing MPI communication.

The point of this comparison is that the interpretation of parallel scaling experiments must consider the hardware they are performed on, since even a good algorithm may look bad on inadequate hardware. Since we want our algorithm to look good, we perform the remainder of our experiments on the Cray cluster.

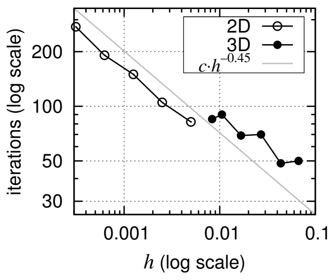
5.4 On the number of iterations for the iterative solver

In [18], the present authors estimated the number of iterations of the $\mathscr{P}_\mathrm{gSGS}$ preconditioner on the current problem as proportional to $h^{-0.4}$–$h^{-0.5}$, where h is the characteristic element size. It should be remarked that this estimation was performed using only a quite small two-dimensional test problem. There are two questions we need to answer.

(i) Is the number of iterations independent of the number of processors P in the strong scaling paradigm?

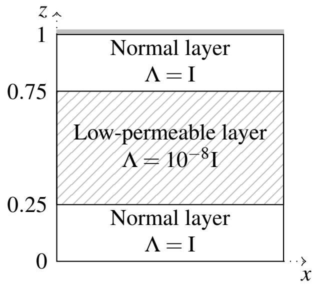


(a) Convergence of test case III on 1 and 512 processors

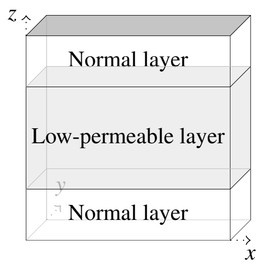


(b) Iterations of the solver for the error convergence criterion (case II)

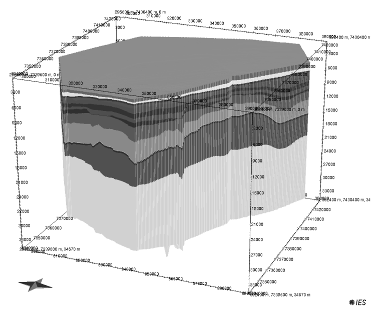
Fig. 5 The dependence of the convergence on the number of processors (a) and the characteristic grid cell size (b) of the problem.



(a) An embedded low-permeable layer in a 2D unit square (I–II)



(b) A low-permeable layer in a 3D unit cube (I–II)



(c) The Vøring basin model, with 16 layers and 1.7 million nodes (III)

Fig. 6 The domains of test cases I–III

(ii) Does the number of iterations keep growing at about the same rate as previously estimated in the weak scaling paradigm?

Question (i) only makes sense if some of the operations in the iterative solver are not independent of P. Generally, the Conjugate Gradient iteration is independent of P, as is the block preconditioner. However, the Smoothed Aggregation single-block preconditioners behave somewhat differently when P is large. In particular, the high-level aggregates do not cross processor boundaries [29]. To answer this question, we compared the convergence of the basin-scale model from Section 5.7 when it is run in sequential mode and in parallel using 512 processors. The results, shown in Fig. 5a, indicate that the differences in convergence are minimal. The smoothed mean of three experiments is shown for each case, along with the individual experiments.

The answer to question (ii) is found in Fig. 5b; $h^{-0.4}$–$h^{-0.5}$ remains a fair estimate of the order of the solver in the two-dimensional case. The three-dimensional case is less clear, owing to insufficient data, but a similar rate is indicated. The test case used to gather this data is described in Section 5.5, with Q^2Q^1 Taylor-Hood elements and a factor 10^{-6} reduction of the error in $L2$-norm as the convergence criterion.

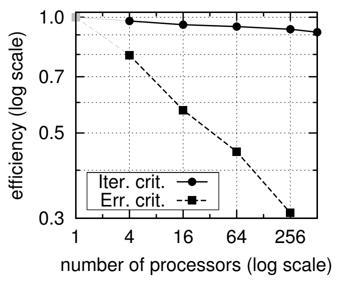
5.5 Test case I: Weak scaling

Layered media with severe jumps in material parameters constitute the normal case in basin modelling. To capture the essence of the numerical difficulties with such media, we have constructed a test problem with three layers as shown in Figs. 6a–6b. We have investigated these (and similar) model problems in earlier works. A low-permeable layer with vanishing fluid storage coefficient S creates an ill-defined problem for the decoupled pressure equation [16], which can nevertheless be solved (up to an arbitrary constant) by an AMG-preconditioned iterative solver. The coupling of the fluid pressure with displacement makes the problem well-defined, but when the permeability contrasts are sufficiently strong (the permeability ratio of adjacent layers is greater than approximately 10^4 with $S = 0$), novel preconditioners such as the one presented herein are required for convergence [18].

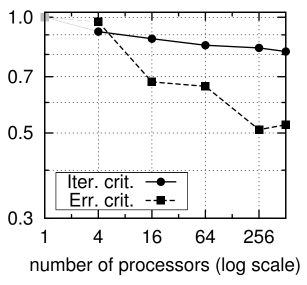
As explained in Section 5.1, we do not care about the actual boundary conditions, except to note where essential conditions are in use:

– The displacement equation has essential boundary conditions in the normal direction at the sides and the bottom,
– The fluid pressure equation has essential boundary conditions at the top.

Another difficulty is that of nonphysical oscillations in the fluid pressure, which may occur in models where low-

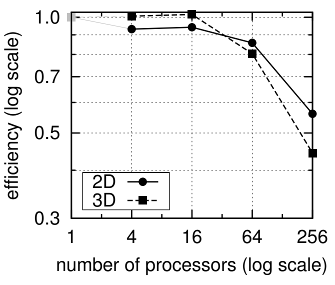


(a) 2D test (see Fig. 6a)

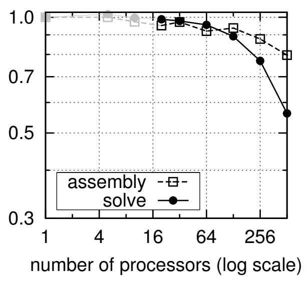


(b) 3D test (see Fig. 6b)

Fig. 7 Weak scaling in a two- and three-dimensional test case



(a) Test case II



(b) Test case III

Fig. 8 Parallel efficiency in strong scaling tests

permeable layers are present. Pursuant to the results in [17], we avoid this by using the Taylor-Hood quadrilateral or hexahedral element combination, with second order Lagrange elements for the displacement and first order Lagrange elements for the fluid pressure.

In this test case of weak scaling, each processor is responsible for about 200^2 elements in 2D, or about 16^3 elements in 3D; these are the largest problems that can fit comfortably in the available 1GB of memory per processor. Parallel partitioning is performed using the METIS graph partitioner [22].

The plots in Fig. 7 show the parallel efficiency of the iterative solver phase of a single time step of this model, i.e., of solving Eq. 12. When using an iteration convergence criterion, the parallel scalability is excellent, with only 10–20% lower efficiency at 512 processors. Since the ratio of foreign to interior nodes is much larger in the three-dimensional case, it is somewhat less efficient than the two-dimensional case. However, once a more practical error criterion is used, this is turned upside down: Since the condition number of the matrix (as a function of the problem size) deteriorates less rapidly in the three-dimensional case, the actual error-reduction efficiency is much better in 3D than in 2D. The 2D efficiency drops below 50% at around 32 processors, while the 3D efficiency still remains above 50% at 512 processors.

5.6 Test case II: Strong scaling

Test case II uses the same model geometry, parameters, elements, and partitioning as test case I. The only difference is that it is fixed in size: 400^2 elements in 2D and 26^3 elements in 3D. Again, the size is determined by memory considerations: These are the largest problems to fit in memory on a single 4GB computational node.

We assume that the number of iterations for a given reduction in error is nearly independent of the number of processors (as discussed in Section 5.4, and in particular Fig. 5a), and hence that the error and iteration criteria are nearly equivalent; an error criterion with $\varepsilon = 10^{-6}$ is used.

The scalability results are shown in Fig. 8a. As we may expect from the results of test case I, the 3D test drops off faster in efficiency, but both tests exhibit adequate scalability up to 256 processors.

We remark that in the strong scaling paradigm, the limits of scalability are determined to a large degree by the problem size. A large problem can be subdivided more times before the number of foreign nodes becomes significant. For example, with 256 processors the number of foreign nodes is larger than the number of interior nodes in the 3D test.

5.7 Test case III: Strong scaling on a basin-scale geometry

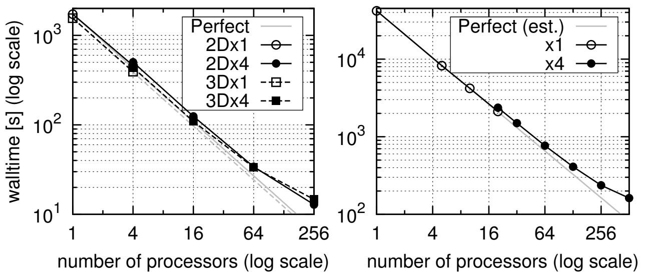
Our final test case is a realistic model of a sedimentary basin, derived from a real industry model. Shown in Fig. 6c, the model consists of 16 distinct layers of sediments, $8.4 \cdot 10^6$ tetrahedral elements, and $1.7 \cdot 10^6$ nodes. No attempt has been made to make the computational grid more friendly to finite element calculations, and thus the grid quality is low in some places — outer/inner radius ratios exceed 100 in many elements. The material parameters are also from the real model, and are listed in Table 1.

For this test, equal-order Lagrange tetrahedral elements P^1P^1 are used. We believe this to be acceptable, since the fluid storage coefficient S does not vanish anywhere (see discussion in Section 5.5 and [17]). Even if it were not acceptable, Taylor-Hood elements would simply be too expensive on this grid — the memory requirements would increase almost tenfold, to well over a hundred gigabytes. One possibility would be to use a mixed element with extra internal degrees of freedom, such as the MINI element, and to eliminate the internal degrees of freedom at the element level by static condensation. Such a procedure would reduce the size of the system to that of the P^1P^1 combination used here.

The efficiency results are shown in Fig. 8b, with the associated runtime (for both the assembly and the solution phase) in Fig. 9b. A peculiarity with these graphs is that the single-processor runtime is only estimated, because the memory requirements for this test case precludes running it on fewer than five computational nodes. This estimate, which is used

Table 1 Material parameters for test case III.

Layer no.	$S[\mathrm{Pa}^{-1}]$	$\Lambda_x, \Lambda_y[\mathrm{m}^2\mathrm{Pa}^{-1}\mathrm{s}^{-1}]$	$\Lambda_z[\mathrm{m}^2\mathrm{Pa}^{-1}\mathrm{s}^{-1}]$	$\nu[\cdot]$	$G[\mathrm{Pa}]$
1	$1\cdot10^{-10} - 2\cdot10^{-10}$	$3\cdot10^{-1} - 7\cdot10^{0}$	$8\cdot10^{1} - 2\cdot10^{3}$	0.35	$5\cdot10^{8}$
2	$1\cdot10^{-10} - 2\cdot10^{-10}$	$6\cdot10^{1} - 3\cdot10^{2}$	$2\cdot10^{4} - 1\cdot10^{5}$	0.35	$5\cdot10^{8}$
3	$1\cdot10^{-10} - 2\cdot10^{-10}$	$3\cdot10^{0} - 2\cdot10^{1}$	$1\cdot10^{2} - 6\cdot10^{2}$	0.35	$5\cdot10^{8}$
4	$1\cdot10^{-10} - 2\cdot10^{-10}$	$2\cdot10^{-2} - 1\cdot10^{-1}$	$8\cdot10^{0} - 3\cdot10^{1}$	0.35	$5\cdot10^{8}$
5	$1\cdot10^{-10}$	$5\cdot10^{-3} - 7\cdot10^{-2}$	$1\cdot10^{0} - 2\cdot10^{1}$	0.35	$5\cdot10^{8}$
6	$1\cdot10^{-10}$	$2\cdot10^{-6} - 5\cdot10^{-2}$	$5\cdot10^{-4} - 2\cdot10^{1}$	0.35	$5\cdot10^{8}$
7	$1\cdot10^{-10}$	$1\cdot10^{-2} - 3\cdot10^{-2}$	$3\cdot10^{0} - 5\cdot10^{0}$	0.35	$5\cdot10^{8}$
8–9	$1\cdot10^{-10}$	$2\cdot10^{-6} - 1\cdot10^{-4}$	$5\cdot10^{-4} - 4\cdot10^{-2}$	0.35	$5\cdot10^{8}$
10	$1\cdot10^{-10}$	$2\cdot10^{-6} - 4\cdot10^{-4}$	$5\cdot10^{-4} - 1\cdot10^{-1}$	0.35	$5\cdot10^{8}$
11	$1\cdot10^{-10}$	$2\cdot10^{-3} - 5\cdot10^{-2}$	$2\cdot10^{-1} - 6\cdot10^{0}$	0.35	$5\cdot10^{8}$
12	$2\cdot10^{-10}$	$5\cdot10^{-2} - 8\cdot10^{0}$	$5\cdot10^{0} - 8\cdot10^{2}$	0.25	$8\cdot10^{8}$
13	$1\cdot10^{-10}$	$2\cdot10^{-3} - 6\cdot10^{-3}$	$4\cdot10^{-1} - 1\cdot10^{0}$	0.35	$5\cdot10^{8}$
14	$6\cdot10^{-11}$	$5\cdot10^{-14}$	$5\cdot10^{-14}$	0.40	$1\cdot10^{9}$
15	$2\cdot10^{-10}$	$3\cdot10^{-1} - 3\cdot10^{2}$	$7\cdot10^{0} - 6\cdot10^{3}$	0.20	$9\cdot10^{8}$
16	$1\cdot10^{-10}$	$2\cdot10^{-2} - 3\cdot10^{-2}$	$3\cdot10^{0} - 6\cdot10^{0}$	0.35	$5\cdot10^{8}$



(a) Test case II **(b)** Test case III

Fig. 9 Actual and perfect-scaling runtime for one time step (assembly and solve), using one (x1) and four (x4) cores per computational node

both to determine the multiplicative factor $T(1)$ in the efficiency and to determine the "perfect scaling" line in Fig. 9b, is made by simply subtracting the MPI communication overhead from the five-processor aggregate runtime.

6 Concluding remarks

We have implemented and tested a parallel block preconditioner for the finite element discretisation of a fully coupled 3D problem of fluid flow in elastic porous media. The parallel preconditioner targets especially the challenges of real-world geological problems: unstructured computational grids, and heterogeneous material parameters with severe jumps between geological layers. As the numerical results in previous sections show, we achieve strong scaling results for a realistic large-scale basin model that are quite acceptable on up to five hundred processors, thereby making simulations on this scale practical. The performance of this parallel block preconditioner is robust with respect to heterogeneities and severe grid distortion.

The smaller strong scaling case (test case II) shows an earlier drop-off in efficiency. This may be expected from the fact that a smaller problem has a higher ratio of foreign nodes to interior nodes, which increases relative communication overhead as well as local overhead.

The results for weak scaling can be interpreted in two different ways. On one hand, the parallel scalability for a fixed number of iterations is very good, and should easily scale into thousands of processors (limited mostly by the per-processor problem size, as alluded to in the strong-scaling case). On the other hand, the preconditioner is not optimal, in that its performance degrades with problem size (see Fig. 5b). This degradation is rather small, but it still overwhelms the parallel overhead, and thus the weak scalability (particularly in 2D) is less good when using an error criterion for convergence. Further research into improving the size-dependence of the preconditioner may be warranted.

As our main result, we demonstrate that good parallel scaling is achievable on a complex problem in coupled geomechanics, using a standard iterative solver and state-of-the-art general single-block preconditioners, combined in a novel fashion.

Acknowledgements The authors would like to thank professor Xing Cai at Simula, who implemented the initial parallel simulator, and who has been very forthcoming with ideas and assistance for using and extending the Diffpack parallel framework. We also thank Statoil ASA for their support, both financial and in access to their geological models and expertise. We are grateful to the Norwegian Metacenter for Computational Science (NOTUR) for allowing us the use of the *hexagon* computational cluster. This work is supported by a Center of Excellence grant from the Norwegian Research Council to Center for Biomedical Computing at Simula Research Laboratory.

References

1. Adams, M.: Evaluation of three unstructured multigrid methods on 3D finite element problems in solid mechanics. Int. J. Numer. Methods Eng. **55**, 519–534 (2002). DOI 10.1002/nme.506
2. Adams, M.F., Bayraktar, H.H., Keaveny, T.M., Papadopoulos, P.: Ultrascalable implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom. In: Proceedings of the ACM/IEEE Conference on Supercomputing (SC2004), p. 34. IEEE Computer Society (2004)
3. The Simula computer cluster *bigblue*. URL `http://simula.no/research/sc/cbc/events/2008/081105-slides/bigblue-intro.pdf`
4. Biot, M.A.: General theory of three-dimensional consolidation. J. Appl. Phys. **12**(2), 155–164 (1941). DOI 10.1063/1.1712886
5. Brezina, M., Falgout, R., MacLachlan, S., Manteuffel, T., McCormick, S., Ruge, J.: Adaptive smoothed aggregation (αSA). SIAM J. Sci. Comput. **25**(6), 1896–1920 (2004). DOI 10.1137/S1064827502418598
6. Catalyurek, U.V., Boman, E.G., Devine, K.D., Bozdag, D., Heaphy, R.T., Riesen, L.A.: Hypergraph-based dynamic load balancing for adaptive scientific computations. In: Proc. of 21st International Parallel and Distributed Processing Symposium (IPDPS'07). IEEE (2007)
7. Chow, E., Falgout, R.D., Hu, J.J., Tuminaro, R., Yang, U.M.: A survey of parallelization techniques for multigrid solvers. In: M.A. Heroux, P. Raghavan, H.D. Simon (eds.) Parallel Processing for Scientific Computing, pp. 179–202. SIAM (2006)
8. URL `http://www.diffpack.com/`. Library for numerical solution of PDEs from inuTech GmbH
9. Doi, S., Washio, T.: Ordering strategies and related techniques to overcome the trade-off between parallelism and convergence in incomplete factorizations. Parallel Comput. **25**, 1995–2014 (1999). DOI 10.1016/S0167-8191(99)00064-2
10. Elman, H.C., Howle, V.E., Shadid, J.N., Shuttleworth, R., Tuminaro, R.S.: A taxonomy and comparison of parallel block multi-level preconditioners for the incompressible Navier-Stokes equations. J. Comput. Phys. **227**(3), 1790–1808 (2007). DOI 10.1016/j.jcp.2007.09.026
11. Elman, H.C., Howle, V.E., Shadid, J.N., Tuminaro, R.S.: A parallel block multi-level preconditioner for the 3D incompressible Navier-Stokes equations. J. Comput. Phys. **187**(2), 504–523 (2003). DOI 10.1016/S0021-9991(03)00121-9
12. Fletcher, R.: Conjugate gradient methods for indefinite systems. In: G. Watson (ed.) Numerical Analysis, *Lecture Notes in Mathematics*, vol. 506, pp. 73–89. Springer (1976). DOI 10.1007/BFb0080116
13. Gee, M.W., Siefert, C.M., Hu, J.J., Tuminaro, R.S., Sala, M.G.: ML 5.0 smoothed aggregation user's guide. Tech. Rep. SAND2006-2649, Sandia National Laboratories (2006). URL `http://software.sandia.gov/trilinos/packages/ml/`
14. George, A., Ng, E.: On the complexity of sparse *QR* and *LU* factorization of finite-element matrices. SIAM J. Sci. Stat. Comput. **9**, 849 (1988). DOI 10.1137/0909057
15. Hackbush, W.: Iterative Solution of Large Sparse Systems of Equations. Springer-Verlag (1995)
16. Haga, J.B., Langtangen, H.P., Nielsen, B.F., Osnes, H.: On the performance of an algebraic multigrid preconditioner for the pressure equation with highly discontinuous media. In: B. Skallerud, H.I. Andersson (eds.) Proceedings of MekIT'09, pp. 191–204. NTNU, Tapir (2009). URL `http://simula.no/research/sc/publications/Simula.SC.568`
17. Haga, J.B., Langtangen, H.P., Osnes, H.: On the causes of pressure oscillations in low-permeable and low-compressible porous media (2011). URL `http://simula.no/publications/Simula.simula.18`. Submitted to *International Journal for Numerical and Analytical Methods in Geomechanics*
18. Haga, J.B., Osnes, H., Langtangen, H.P.: Efficient block preconditioners for the coupled equations of pressure and deformation in highly discontinuous media. Int. J. Numer. Anal. Methods Geomech. (2010). URL `http://simula.no/research/sc/publications/Simula.SC.660`. Accepted for publication.
19. Heroux, M.A., Bartlett, R.A., Howle, V.E., Hoekstra, R.J., Hu, J.J., Kolda, T.G., Lehoucq, R.B., Long, K.R., Pawlowski, R.P., Phipps, E.T., Salinger, A.G., Thornquist, H.K., Tuminaro, R.S., Willenbring, J.M., Williams, A., Stanley, K.S.: An overview of the Trilinos project. ACM Trans. Math. Softw. **31**(3), 397–423 (2005). DOI 10.1145/1089014.1089021
20. The NOTUR computer cluster *hexagon*. URL `http://www.notur.no/hardware/hexagon`
21. Joubert, W., Cullum, J.: Scalable algebraic multigrid on 3500 processors. Electron. Trans. Numer. Anal. **23**, 105–128 (2006)
22. Karypis, G., Schloegel, K., Kumar, V.: ParMETIS parallel graph partitioning and sparse matrix ordering library, version 3.1. University of Minnesota, Minneapolis (2003). URL `http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview`
23. Langtangen, H.P.: Computational Partial Differential Equations: Numerical Methods and Diffpack Programming, 2nd edn. Springer (2003)

24. URL `http://www.petromod.com/`. Petroleum systems modelling software from Schlumberger Aachen Technology Center
25. Phoon, K.K., Toh, K.C., Chan, S.H., Lee, F.H.: An efficient diagonal preconditioner for finite element solution of Biot's consolidation equations. Int. J. Numer. Methods Eng. **55**, 377–400 (2002). DOI 10.1002/nme.500
26. Thakur, R., Gropp, W.: Improving the performance of collective operations in MPICH. Recent Adv. Parallel Virtual Mach. Message Passing Interface pp. 257–267 (2003)
27. Toh, K.C., Phoon, K.K., Chan, S.H.: Block preconditioners for symmetric indefinite linear systems. Int. J. Numer. Methods Eng. **60**, 1361–1381 (2004). DOI 10.1002/nme.982
28. Tuminaro, R.S., Tong, C.: Parallel smoothed aggregation multigrid: Aggregation strategies on massively parallel machines. In: Proceedings of the 2000 ACM/IEEE conference on Supercomputing. IEEE Computer Society (2000). DOI 10.1109/SC.2000.10008
29. Yang, U.M.: Parallel algebraic multigrid methods—high performance preconditioners. In: A.M. Bruaset, A. Tveito (eds.) Numerical Solution of Partial Differential Equations on Parallel Computers, pp. 209–236. Springer (2006). DOI 10.1007/3-540-31619-1_6