Computational Geosciences (2024) 28:241–251
https://doi.org/10.1007/s10596-023-10215-4

ORIGINAL PAPER

Nonlinear domain-decomposition preconditioning for robust and efficient field-scale simulation of subsurface flow

Olav Møyner[1] · Atgeirr F. Rasmussen[1] · Øystein Klemetsdal[1] · Halvor M. Nilsen[1] · Arthur Moncorgé[2] · Knut-Andreas Lie[1]

Received: 14 November 2022 / Accepted: 20 April 2023 / Published online: 27 July 2023

Abstract
We discuss a nonlinear domain-decomposition preconditioning method for fully implicit simulations of multicomponent porous media flow based on the additive Schwarz preconditioned exact Newton method (ASPEN). The method efficiently accelerates nonlinear convergence by resolving unbalanced nonlinearities in a local stage and long-range interactions in a global stage. ASPEN can improve robustness and significantly reduce the number of global iterations compared with standard Newton, but extra work introduced in the local steps makes each global iteration more expensive. We discuss implementation aspects for the local and global stages. We show how the global-stage Jacobian can be transformed to the same form as the fully implicit system, so that one can use standard linear preconditioners and solvers. We compare the computational performance of ASPEN to standard Newton on a series of test cases, ranging from conceptual cases with simplified geometry or flow physics to cases representative of real assets. Our overall conclusion is that ASPEN is outperformed by Newton when this method works well and converges in a few iterations. On the other hand, ASPEN avoids time-step cuts and has significantly lower runtimes in time steps where Newton struggles. A good approach to computational speedup is therefore to adaptively switch between Newton and ASPEN throughout a simulation. A few examples of switching strategies are outlined.

Keywords Fully implicit methods · Newton's method · Additive Schwarz Preconditioned Exact Newton (ASPEN)

1 Introduction

Reservoir simulation models from commercial applications are usually stiff in the sense that the flow equations contain terms that can lead to rapid variations in the solution. Most reservoir simulators therefore rely on implicit formulations, which give rise to large algebraic systems of nonlinear equations to be solved for each time step. The standard approach is to use Newton's method. At its best, the second-order Newton method will converge within a few iterations, but in practice, the Newton updates must be dampened and the method applied in combination with strategies for cutting the time step when iterations fail or converge too slowly. This will typically happen because natural initial guesses are too far from the solution or as a result of unbalanced nonlinearities in space and time. Imbalanced nonlinearities can have many different causes, including strongly coupled flow equations with mixed elliptic–hyperbolic subcharacter and time constants that vary largely throughout the domain (and in time); coupling of well and reservoir equations; rapid transients induced by (discontinuous) changes in well controls; strong heterogeneities in petrophysical properties amplified by unstructured, non-orthogonal grids with high aspect ratios; non-smooth and hysteretic rock-fluid properties; etc. This not only leads to badly conditioned linear

✉ Olav Møyner
olav.moyner@sintef.no

Atgeirr F. Rasmussen
atgeirr.rasmussen@sintef.no

Øystein Klemetsdal
oystein.klemetsdal@sintef.no

Halvor M. Nilsen
halvormoll.nilsen@sintef.no

Arthur Moncorgé
arthur.moncorge@totalenergies.com

Knut-Andreas Lie
knut-andreas.lie@sintef.no

[1] SINTEF Digital, Mathematics & Cybernetics, P.O. Box 124 Blindern, N-0314 Oslo, Norway

[2] TotalEnergies, Reservoir Simulation Department, BB3110, Avenue Larribau, 64018 Pau, France

systems that are expensive to solve with standard iterative solvers, but a lot of computational effort is often wasted on iterations that are discarded as a result of time-step chops, in particular when one tries to run the simulator with longer time steps in prediction mode.

To more efficiently handle problems with unbalanced nonlinearities, Cai and Keyes [1] proposed to use domain decomposition as a nonlinear preconditioner, giving the so-called additive Schwarz preconditioned exact Newton method (ASPIN); a multiplicative version was developed later [2]. In either case, the key idea is to replace some of the global Newton iterations, which require the solution of large and often ill-conditioned linear (Jacobian) systems that are expensive to solve and do not parallelize well, by iterations localized to smaller subdomains that give smaller and better conditioned linear systems. Likewise, solving in subdomains will introduce local control on the iteration process and contribute to take some of the "stiffness" out of the nonlinear algebraic problems. Variants of nonlinear domain-decomposition preconditioning have been applied for immiscible two-phase porous media flow [3–5], single-phase Forchheimer flow [6], and to improve the convergence of sequential fully implicit schemes [7, 8], for which other acceleration strategies have also been studied [9–11]. In particular, Dolean et al. [6] were the first to point out that the exact Jacobian is easy and inexpensive to compute after the subdomain solves, and may well be used instead of the inexact one suggested in the original ASPIN formulation.

Recently, we extended ASPEN to three-phase compositional flow, showed how to reformulate the associated global Jacobian matrix so that one can use well-known, efficient iterative linear solvers, and demonstrated significant reduction in the number of nonlinear iterations for both fully implicit and sequentially implicit formulations across a wide variety of cases [12]. In another paper [13], we showed how the ASPEN formulation can be used to develop an adaptive sequentially fully implicit solver that uses a sequential fully implicit solution formulation in subdomains with weaker coupling between flow and transport and a fully implicit formulation in subdomains with strong coupling.

The disadvantage of ASPEN (and similar methods) is that each iteration is more expensive than a single iteration with Newton. Not only do we need to solve subdomain problems but we also need a global step. In principle, this step is comparable in complexity to a standard Newton update, but is usually larger in magnitude and will hence require more linear iterations to converge. To be more efficient than standard Newton, ASPEN thus needs to provide a sufficient reduction in the number of nonlinear iterations to overcome the increased cost of each iteration. The purpose of this paper is to study the balance between reduced iteration count and increased iteration cost in an optimized implementation of ASPEN. We show how the cost of the global step can be reduced by recasting it as a correction to the local updates. We also discuss how one can improve efficiency and robustness of the nonlinear solver by adaptively switching between Newton and ASPEN steps throughout the simulation and/or iteration process.

2 Nonlinear domain-decomposition preconditioning

In the following, we consider a standard fully implicit, finite-volume discretization of the flow equations for a multiphase, multicomponent system

$$\frac{M_i^{n+1} - M_i^n}{\Delta t^n} + \mathsf{div}(V_i^{n+1}) - Q_i^{n+1} = 0, \quad i = 1, \ldots, m. \tag{1}$$

Here, the vectors M_i and Q_i contain the conserved quantity of component i and the corresponding source terms in all the grid cells, and V_i holds the flow rates for component i across each cell interface. Superscript n refers to the timestep and div is a discrete analogue of the standard divergence operator; see, Lie [14, Section 4.4] for details. To advance the solution, we thus have to solve an algebraic system of nonlinear equations for each time step, which we write compactly as $R(u; x) = 0$, where u is the set of *primary* unknown states associated with a spatial position x in our computational domain Ω from which M_i, V_i and Q_i can be computed. For a basic two-phase model, these primary unknowns will typically be chosen to be a fluid pressure p and a fluid saturation S, so that $u = (p, S)$.

Domain decomposition (DD) methods are based on the old divide-and-conquer idea and solve problems by first splitting them into smaller problems solved on local subdomains with fixed boundary conditions and then iterating to coordinate the solutions across the subdomains. To present the nonlinear domain decomposition idea, it is sufficient to consider two non-overlapping[1] subdomains Ω_1 and Ω_2 defined such that $\Omega_1 \cup \Omega_2 = \Omega$, over which we decompose u so that $u_i(x) = u(x)$ for $x \in \Omega_i$. We apply the same decomposition to the residual so that $R_i(u_1, u_2) = R(u; x \in \Omega_i)$. To be clear, R_1 here denotes the set of nonlinear algebraic equations for the unknowns u_1 in Ω_1, which will also involve values u_2 from inside Ω_2 since the discrete divergence operator div is a stencil of finite width that also operates on neighboring cells across the inter-domain boundary. The definition of R_2 is symmetric. Now, let $\mathcal{S}_i(u)$ denote the formal solution operators that solve each of these residual equations, which we with a slight abuse of notation can write in fixed-point form

[1] Notice that in this case, the one-level restricted ASPEN (RASPEN) method of Dolean et al. [6] coincides with ASPEN.

as $u_i = \mathcal{S}_i(u_1, u_2)$. We then write the additive nonlinear domain decomposition method as

$$R_1\big(\mathcal{S}_1(u_1, u_2), u_2\big) = 0, \quad R_2\big(u_1, \mathcal{S}_2(u_1, u_2)\big) = 0. \tag{2}$$

If we collect the two local solution operators into one operator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, the equivalent nonlinear equation suitable for fixed-point iterations reads

$$R(u) = 0 \iff u = \mathcal{S}(u) \iff F(u) \equiv u - \mathcal{S}(u) = 0. \tag{3}$$

Applying Newton's method to solve this equation gives

$$u^{k+1} = u^k + \Delta u, \quad -\frac{\partial F}{\partial u}\Delta u = F(u^k), \quad \text{where}$$
$$\frac{\partial F}{\partial u} = I - \frac{\partial \mathcal{S}}{\partial u}. \tag{4}$$

Because F is defined in terms of the solution operator $\mathcal{S}$, computing its Jacobian requires some consideration. Going back to the definition of the residual equations and remembering to compute partial derivatives with respect to all variables, it follows that

$$\begin{aligned}\frac{\partial R_1}{\partial u} &= \frac{\partial R_1}{\partial u_1}\frac{\partial \mathcal{S}_1}{\partial u} + \frac{\partial R_1}{\partial u_2}\frac{\partial u_2}{\partial u} = 0 \\ \Rightarrow \frac{\partial \mathcal{S}_1}{\partial u} &= -\left(\frac{\partial R_1}{\partial u_1}\right)^{-1}\frac{\partial R_1}{\partial u_2}\frac{\partial u_2}{\partial u},\end{aligned} \tag{5}$$

evaluated in $(u_1, u_2) = (\mathcal{S}_1(u), u_2)$. For an additive method, we likewise have that

$$\begin{aligned}\frac{\partial R_2}{\partial u} &= \frac{\partial R_2}{\partial u_1}\frac{\partial u_1}{\partial u} + \frac{\partial R_2}{\partial u_2}\frac{\partial \mathcal{S}_2}{\partial u} = 0 \\ \Rightarrow \frac{\partial \mathcal{S}_2}{\partial u} &= -\left(\frac{\partial R_2}{\partial u_2}\right)^{-1}\frac{\partial R_2}{\partial u_1}\frac{\partial u_1}{\partial u},\end{aligned} \tag{6}$$

evaluated in $(u_1, u_2) = (u_1, \mathcal{S}_2(u))$. The method generalizes naturally to N subdomains.

The matrix $\partial F/\partial u$ is generally dense, expensive to assemble, and challenging to solve efficiently with a standard iterative linear solver. However, if we look at its block structure, it is easy to see that the matrix can be factored as follows [12]:

$$\begin{aligned}\frac{\partial F}{\partial u} &= \begin{bmatrix} I_1 & \left(\frac{\partial R_1}{\partial u_1}\right)^{-1}\frac{\partial R_1}{\partial u_2} \\ \left(\frac{\partial R_2}{\partial u_2}\right)^{-1}\frac{\partial R_2}{\partial u_1} & I_2 \end{bmatrix} \\ &= \begin{bmatrix} \left(\frac{\partial R_1}{\partial u_1}\right)^{-1} & 0 \\ 0 & \left(\frac{\partial R_2}{\partial u_2}\right)^{-1} \end{bmatrix}\begin{bmatrix} \frac{\partial R_1}{\partial u_1} & \frac{\partial R_1}{\partial u_2} \\ \frac{\partial R_2}{\partial u_1} & \frac{\partial R_2}{\partial u_2} \end{bmatrix} = D^{-1}\frac{\partial R}{\partial u}.\end{aligned} \tag{7}$$

The block matrix D is sparse, and if we insert this factorization into the second equation of (4) and premultiply with D, we end up with the following linearized problem for the Newton increment Δu

$$-\frac{\partial R}{\partial u}\Delta u = DF(u). \tag{8}$$

This system has the same form as the Jacobian of our original algebraic system, except for slight differences in the off-diagonal blocks at the boundaries between subdomains and a different right-hand side. The system can therefore be solved with the same linear solvers as one would use for the usual Newton iteration of fully implicit systems. The ASPEN method can be applied to domain decomposition in both physical and state space; all you need is a reliable (and efficient) method to solve the residual equation in each subdomain and then ASPEN gives systematic global corrections. However, extra care is necessary to ensure correct treatment for flow models that involve variable switching like in black-oil models, and more research remains here.

2.1 Remarks on the implementation

To make a highly efficient implementation of ASPEN, it is important to reuse as many computations from the local stages as possible. We can summarize the overall algorithm as follows:

1. Solve for $u_i^{k+1/2}$ in each subdomain (nonlinearly).
2. Compute the local ASPEN residual $D_i F_i = r_{ai} = J_i(u_i^k - u_i^{k+1/2})$, where $J_i = \partial R_i/\partial u_i$ is the Jacobian in subdomain i.
3. Compute boundary fluxes with partial derivatives between subdomains ($\partial R_j/\partial u_i$).
4. Assemble the global Jacobian J_a from local Jacobians and boundary fluxes.
5. Assemble the global ASPEN residual r_a from local ASPEN residuals.
6. Solve the global system $-J_a\Delta u = r_a$ to obtain updates such that $u^{k+1} = u^k + \Delta u$.

Written in this form, Δu is the increment relative to the start of the subdomain solves. Applying standard dampening techniques (e.g., that restrict saturation or pressure increments) to this update will generally be too restrictive. Instead, we only dampen the part of the increment that has not passed through similar "quality controls" in the subdomain iterations. We can then define

$$\begin{aligned}\delta u &= \Delta u - \big(u^{k+1/2} - u^k\big) = \big(u^{k+1} - u^k\big) - \big(u^{k+1/2} - u^k\big) \\ &= u^{k+1} - u^{k+1/2}.\end{aligned}$$

Applying dampening only to δu should avoid too aggressive "chopping" of the part of the update that has already proved

to be safe during successful subdomain solves, which will hopefully lead to improved performance. So, to summarize, we implement the algorithm exactly as just outlined, except that the global solve in Step 6 is replaced by

$$-J_a \Delta u = -J_a\big(\delta u + (u^{k+1/2} - u^k)\big) = r_a, \tag{9}$$

which is equivalent to

$$-J_a \delta u = r_a + J_a\big(u^{k+1/2} - u^k\big). \tag{10}$$

Notice that J_a becomes the fully implicit Jacobian as the iteration converges, and that the corrected right-hand side residual can be used to check convergence in the normal way; i.e., compute mass-balance errors (MAT BAL in ECLIPSE notation), maximum normalized residuals (CNV in ECLIPSE notation), etc. This does not significantly change the convergence behavior of the schemes in terms of number of required iterations, but removes the need for an extra assembly of the Newton residual in each iteration to check convergence. This additional assembly has significant cost, and removing it improves computational performance substantially. Also, we no longer need as strict linear tolerance for the ASPEN solve as we would use in a straightforward implementation that solves for the full update.

For comparison, we will also consider a simple ad hoc method that just concatenates local subdomain iterations and a standard Newton step without making any attempt to construct an overall fixed-point iteration as in (4). In lack of a better name, we refer to this method as NLDD. This method is less code-invasive than ASPEN and can thus be seen as a first step in implementing ASPEN. We note in the passing that NLDD is similar to the inexact Newton nonlinear elimination (INB-NE) method [15], with the difference that instead of solving locally only for a selected subset of the variables (determined "bad" in a suitable measure), NLDD solves for all variables in the local stage.

3 Numerical examples

In previous work [12, 13], we have demonstrated the ability of ASPEN to reduce the number of nonlinear iterations and improve the general robustness of the nonlinear solve compared to Newton. For this, we have used the automatic differentiation (AD-OO) simulator framework of MRST [14]. In [12], we also presented a simplified complexity analysis which indicates that under ideal parallelization, the extra cost incurred from the local solves is small compared to the cost of the global correction stage. Use of memory-optimized backends for automatic differentiation and high-performing linear solvers written in complied languages has brought the computational power of MRST to the level of compiled simulators [16]. Implementations in native MATLAB will nonetheless inevitably have a certain computational overhead that will bias actual runtimes observed when comparing nonlinear solvers. We could therefore not back up claims regarding ASPEN's computational efficiency with credible observations of reduced runtime.

To do this, we will herein instead rely on a highly optimized computational backend (Jutul) written in Julia, a dynamic, open-source programming language for scientific computing that aims to achieve the same performance as Fortran or C++ with a high-level syntax similar to Python or MATLAB. Jutul relies on MRST for preprocessing simulation cases for immiscible, black-oil, and equation-of-state compositional flow. Thanks to a highly efficient prototype implementation with static, hard-coded stencils, the performance of Jutul's automatic differentiation library is a factor 3–5 times faster than in OPM Flow [17] and AD-GPRS [18, 19], which in turn gives significantly faster assembly. Linear systems are solved with a standard Krylov method together with a constrained-pressure residual (CPR) preconditioner [20, 21] that uses AMG and block-ILU(0) for the first and second stages, respectively. All wells are treated as multisegment wells. Jutul is open source software[2], but our ASPEN implementation is not yet public.

In the following, we present a total of five test cases. The first test case is a conceptual example included to illustrate typical convergence of the different nonlinear solvers, whereas the following four test cases compare computational performance of ASPEN and Newton in terms of both iteration counts and actual runtimes observed. We will also discuss a few strategies to hybridize the two methods for optimal performance.

3.1 Case 0: Buckley–Leverett displacement

We start our examples with a small conceptual case that corresponds to Example 1 in [12]. A two-phase one-dimensional Buckley–Leverett displacement is discretized into 1000 fine cells and five coarse subdomains. Using 200 time steps gives an approximate CFL number of 5. We remove the limits on pressure and saturation changes that are standard in industry-grade simulators so that the only limit in place is the physical constraint that saturations should be within the unit interval. Disabling update limits is generally not feasible for more complex models. However, in this particular, simple case, it allows us to test an unmodified Newton solver against ASPEN, and test whether our modified ASPEN formulation in (9) has significant impact on the convergence behavior. Figure 1 reports the cumulative iteration counts together with the error reduction in the L_2 norm as function of iteration steps for the first two time steps of the simulation. Com-

[2] https://github.com/sintefmath/JutulDarcy.jl

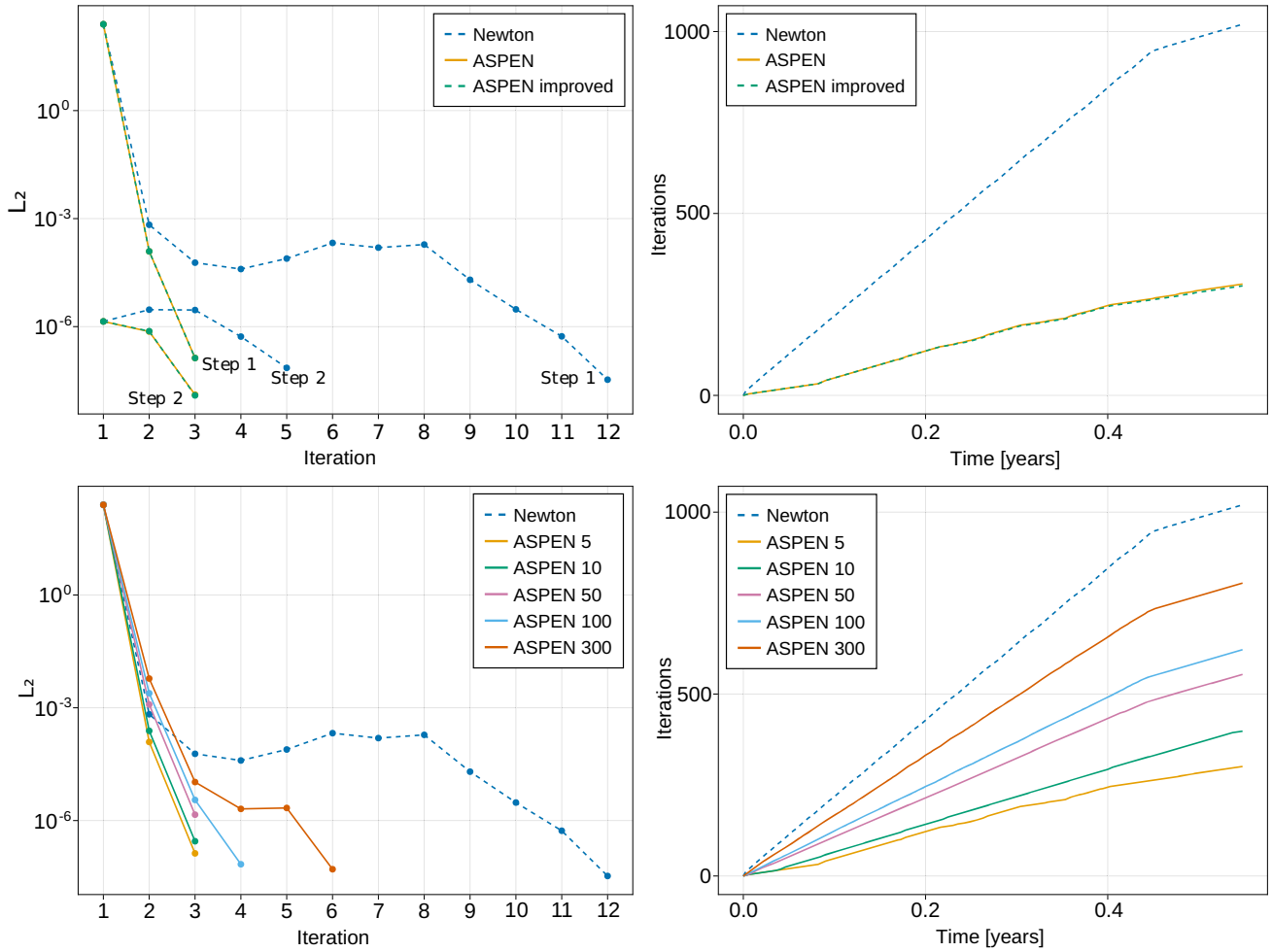


Fig. 1 The upper row shows error reduction for the first two steps (upper left) and cumulative iteration counts for the full simulation (upper right) for a simple Buckley–Leverett displacement with standard Newton, ASPEN, and the alternate ASPEN formulation (denoted "ASPEN-improved") as nonlinear solvers; the local stages use five uniformly sized subdomains. The first step is the initiation of the simulation and requires 12 iterations to complete for Newton. The second step uses less iterations and is representative of the remainder of the time steps. We have plotted only the non-converged residuals. The same experiment is repeated in the second row for varying number of coarse blocks. For legibility, we have only plotted the error reduction for the first iteration

paring computational performance measured in runtime is not especially meaningful for this very small example, as all versions run in approximately a second with a direct linear solver. We see the expected reduction in nonlinear iteration count relative to Newton when ASPEN is used. The standard and the improved ASPEN formulations show nearly identical convergence. This is also confirmed in other cases. In the following, we thus only consider our improved version, which has lower runtime because of fewer residual evaluations.

The lower row in Fig. 1 compares error reduction and cumulative iteration count for various sizes of the local subdomains. Here, we clearly see that the larger the subdomains, the better the convergence. On the other hand, larger subdomains usually means higher computational costs for the local stage, and in practice there is a trade-off between improved convergence and increased computational cost. In the following, we will as a simple rule of thumb use partitions with between 500 and 1000 cells per subdomain, which on our hardware gives a good balance for the cost of the local solves relative to the reduction in the global nonlinear iterations.

3.2 Case 1: fractured reservoir

Our first performance test case is a continuation of a test case reported earlier by Klemetsdal et al. [12, Example 2], which considers a horizontal reservoir cross-section consisting of five bands with distinctly different petrophysical properties, intersected by a pattern of thirteen fracture corridors. The domain is described by a fully unstructured PEBI grid with 7,932 cells that adapt to the fracture corridors, represented as

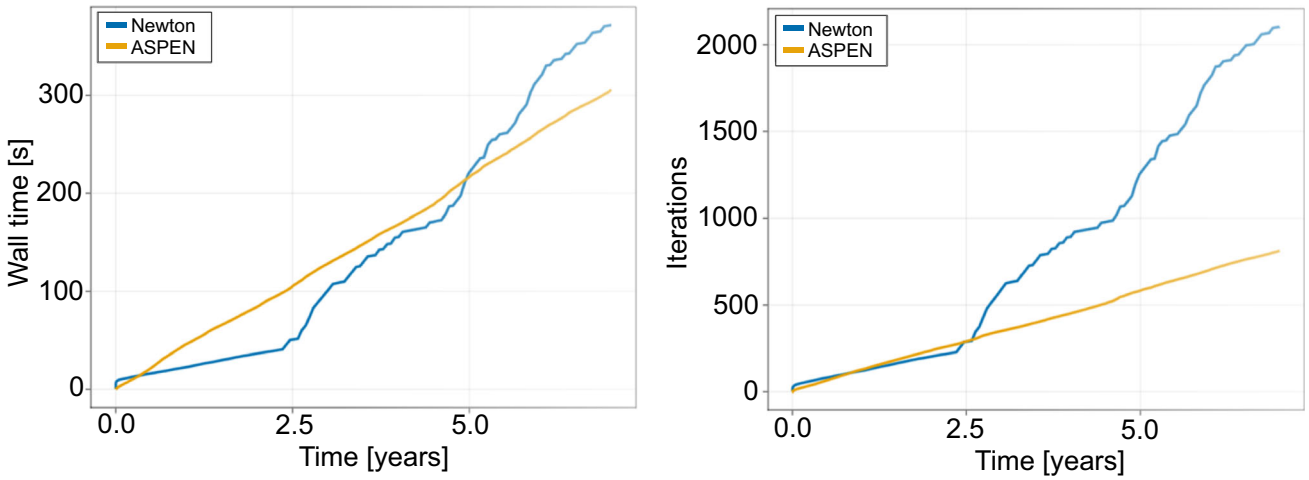


Fig. 2 Computational performance test for the compositional, fractured case with a three-component fluid model, simulated by a fully implicit method with either standard dampened Newton ("fully implicit" in the legend) or ASPEN as nonlinear solver. The left plot shows cumulative runtime and the right plot the cumulative number of global iterations used per time step

volumetric cells with high permeability. The reservoir is initially found at 75 bar and 150 degrees C and is filled by a mixture of methane (30%), carbon dioxide (10%), and n-decane (60%). Hydrocarbons are produced by injecting 0.25 pore volumes of supercritical carbon dioxide, containing 10% methane.

During the first injection phase, Newton and ASPEN converge steadily, using on average 3.7 and 2.0 nonlinear iterations per timestep, respectively. After the injected CO_2 has broken through at 900 days, Newton fails to converge and must cut the timestep in half multiple times, wasting many iterations. ASPEN, on the other hand, continues to converge steadily. Averaged over all timesteps, Newton uses 18.6 nonlinear iterations, whereas ASPEN requires only 2.2. However, each iteration step with ASPEN is more expensive than with Newton because of the local assembly and solves in the domain-decomposition preconditioner and the extra work required to evaluate the modified right-hand side and the Jacobian of the global step. A reduction in iteration numbers does therefore not necessarily translate to an equally big saving in computational cost.

Figure 2 reports cumulative runtimes and cumulative iteration count for the dampened Newton method and ASPEN. Altogether, Newton required 2103 iterations, out of which 1280 were wasted, and consumed a total runtime of 6.2 minutes. In comparison, ASPEN did not suffer from any time-step cuts and hence did not waste any iterations, requiring a total of 814 global iterations and a total runtime of 5.1 minutes. With Newton, the majority of the runtime is consumed in the linear assembly, whereas ASPEN uses most of its time in the subdomain solves.

The test case is very challenging for Newton because large pressure changes that result when the injected gas breaks through in the producer, which causes Newton's convergence to deteriorate significantly. The setup is thus not necessarily representative of what one would see in a well-tuned model. However, it clearly demonstrates the advantage of using ASPEN as a very robust solver for difficult time steps.

Given this insight, an ad-hoc remedy would be to use Newton up to breakthrough and ASPEN thereafter. Figure 3 shows that this strategy is highly efficient measured in terms of iteration count. In fact, the total iteration count is lower than when ASPEN is applied to all time steps because Newton uses slightly fewer iterations than ASPEN up to breakthrough and because ASPEN, in our implementation, is configured to always solve one global step. In comparison, using ASPEN only in the first iteration results in fewer iterations but slightly higher runtime up to breakthrough. After breakthrough, the scheme fails to converge several time steps and altogether wastes 340 iterations, clocking in at 4.2 minutes and 972 iterations. The iteration count is higher than for ASPEN and for the hybrid strategy with Newton up to breakthrough, but the runtime is nonetheless slightly lower because the added cost of using the more expensive ASPEN steps in 2/3 of the simulation cancels the computational gain from reduced iterations.

3.3 Case 2: olympus waterflooding

The Olympus reservoir from the ISAAP Optimization Challenge [22] consists of 50 model realizations inspired by a North Sea reservoir. The reservoir covers an approximate area of 9×3 km^2, is bounded on one side by a major fault, and contains six internal faults. An impermeable shale divides the 50 m thick reservoir into two zones: an upper zone consisting of high-permeability fluvial channel sands

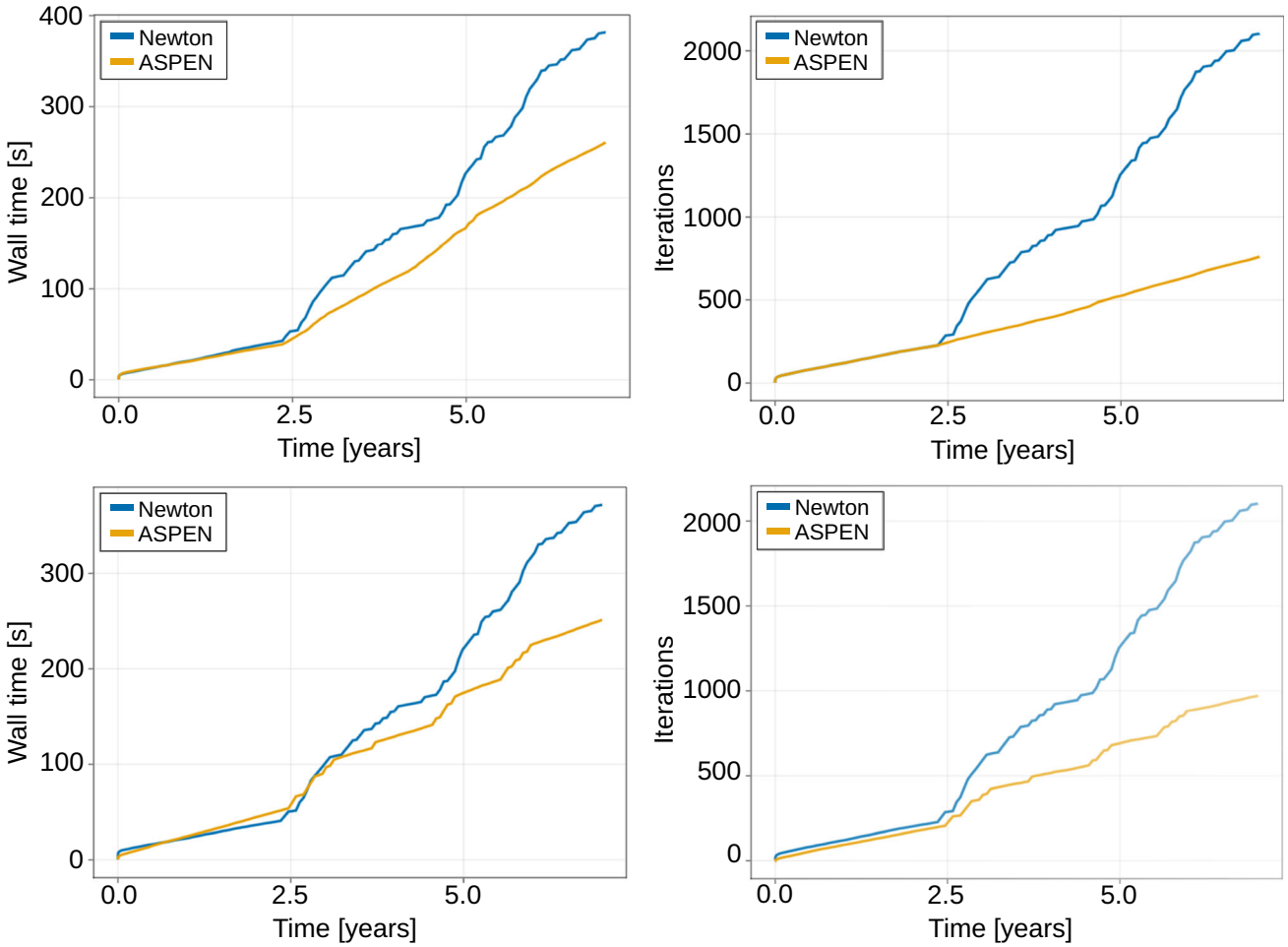


Fig. 3 Cumulative runtime and iteration count for two different hybrid strategies applied to the three-component, fractured test case: the top plots use Newton up to breakthrough and ASPEN thereafter, the lower plots use ASPEN in the first iterations and Newton thereafter. (In the legends: "fully implicit" refers to standard Newton, whereas "ASPEN" are the hybrid solvers.)

(permeability 1 darcy, porosity 0.35) embedded in floodplain shales, and lower zone consisting of alternating layers of coarse, medium, and fine sands with a predetermined dip similar to a clinoformal stratigraphic sequence. The geology is modelled on a $118 \times 181 \times 16$ corner-point grid with 192 749 active cells Fig. 4. The two-phase oil/water model assumes compressible fluids with densities of 850 and 1020 kg/m^3 and viscosities of 2.59 and 0.395 cp at standard conditions, respectively. Herein, we simplify the model by using the same relative permeability curves in all fluid regions and by assuming that the reservoir initially is filled with pure oil.

A standard dampened Newton method requires 5.8 nonlinear iterations per time step on average and wastes no iterations in time-step cuts (see Table 1), which we consider as acceptable computational performance. For ASPEN with 67 subdomains (including one for each well), the total number of global iterations is reduced by 43% and the automatic time-step selection also gives one less time step. This is not sufficient to counteract the added cost of the local solves on a single core. If the subdomains are solved concurrently on four cores, the overall runtime is the same as for Newton. With NLDD, the local and global iterations do not work in union in some time steps and the method ends up wasting eighty global iterations.

To make the case more challenging for the Newton solver, we reduce the compressibility of the oil. This causes a dramatic increase in the global iterations for Newton, which now ends up wasting many iterations. ASPEN also needs more iterations but does not waste any iterations and ends up with a 46% reduction in runtime compared with Newton. NLDD still wastes iterations, but less than in the more compressible case, and all over gives a 38% reduction in runtime compared with Newton.

Table 1 Runtime statistics for the Olympus model simulated with Jutul; iterations wasted on time-step cuts are given in parenthesis.

Method	Single core			Four cores			Modified	
	Glob.its	Time	Steps	Glob.its	Time	Steps	Glob.its	Time
Newton	139 (0)	133 sec.	24	139 (0)	102 sec.	24	426 (160)	222 sec.
NLDD	147 (80)	222 sec.	29	147 (80)	136 sec.	29	126 (20)	138 sec.
ASPEN	79 (0)	144 sec.	23	79 (0)	102 sec.	23	111 (0)	120 sec.

(With OPM Flow, the same simulation with Newton as the nonlinear solver consumed 240 seconds.) Simulations for the modified model with reduced oil compressibility were performed using a single core on a different computer

3.4 Case 3: Real asset model

Our next example is a real asset model consisting of a $58 \times 145 \times 40$ corner-point grid with 141,756 active cells, which together with a three-phase, seven-component fluid model results in almost one million unknowns. A pattern of 22 wells inject and extract fluids over two decades, discretized by timesteps ranging from 1 to 34 days. Injectors operate at fixed total rates, and producers at fixed oil or liquid rates. Here, we consider a slightly simplified version that uses a simple separator instead of a complete separator tree, disregards end-point scaling, and only considers the first 100 out of a total of 340 report steps.

Simulations are run with standard Newton, NLDD, and ASPEN on a single core using the same linear solver and somewhat relaxed convergence criteria chosen to enable the fully implicit solver to use time steps of reasonable length within the prescribed report steps. Newton resolves the 100 report steps in 333 substeps with an average length of 4.17 days and a median length of 3.07 days. The length of individual time steps range from 0.04 days to 28 days. ASPEN and NLDD can consistently take longer time steps, and NLDD has a mean time-step length of 6.7 days, with median of 4 days. ASPEN has an average of 6.8 days and median of 6.0 days. The distribution of these time steps in Fig. 5 clearly show how NLDD and ASPEN have a wider span of time-step lengths compared with Newton.

In the plot of accumulated runtime and global iterations in Fig. 6 the trend is as observed in other examples; NLDD and ASPEN use significantly fewer iterations, but each iteration is more costly in terms of runtime because of the local subdomain solves. In this case, the reduction in iteration count clearly out-weights the increased step cost. The Newton base case runs in 5.2 hours, whereas NLDD and ASPEN run in 2.3 and 1.66 hours, resulting in a speedup of 2.3 and 3.1, respectively. ASPEN reuses results from the local subdomain solves, including flash, for the global assembly. NLDD does not, but this would be an obvious candidate for improvement that would bring the global assembly costs for ASPEN and NLDD close together.

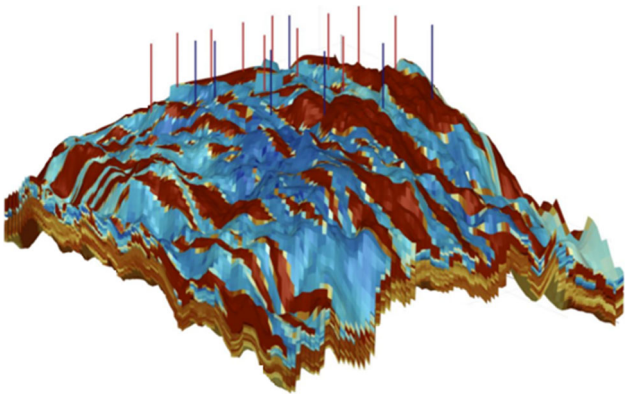


Fig. 4 The Olympus field model [22] has 192 749 active and 18 wells. Colors signify the logarithm of the lateral permeability

3.5 Case 4: Sleipner CO_2 storage model

The final example is a CO_2 injection case based on the 2019 Sleipner benchmark model.[3] The specific permeability and porosity distribution is taken from the version distributed by the Open Porous Media initiative (OPM).[4] The model is described by a $64 \times 118 \times 263$ corner-point grid having a total of 1 986 176 active cells. The model is layered both in permeability and grid resolution, with large flow compartments of good sands with high permeability (1-3 Darcy), separated by finely gridded layers of nearly impermeable shales with permeabilities as low as 1 micro Darcy.

One megatonne of CO_2 is injected annually over a time horizon of 14.25 years from a well perforated in the lower part of the formation. The CO_2 plume formed from the injection migrates quickly upwards until it reaches a shale, at which point the plume splits into a horizontally-migrating part that remains within the flow compartment, and a small volume of CO_2 that penetrates the shale. The upward migration through the shale is limited by the brine–gas capillary pressure. Once the shale has been passed, the mobile CO_2 continues to move upward, and the same process is repeated inside this new flow

[3] https://co2datashare.org/dataset/sleipner-2019-benchmark-model

[4] https://github.com/OPM/opm-data

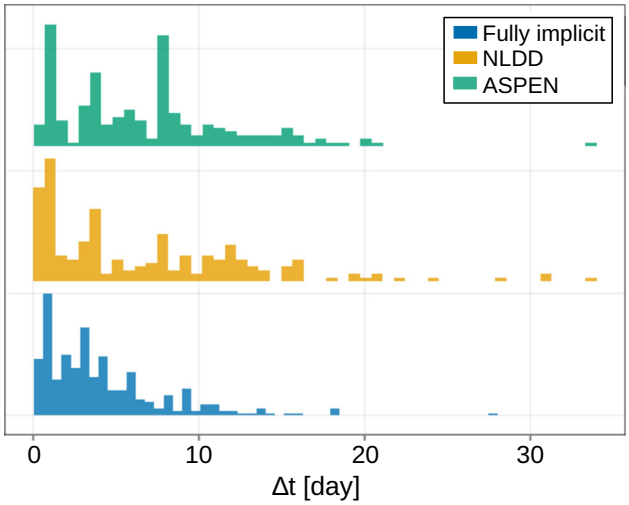


Fig. 5 Time-step lengths within the prescribed reports steps for the real asset model with Newton ("fully implicit"), NLDD, and ASPEN as nonlinear solvers

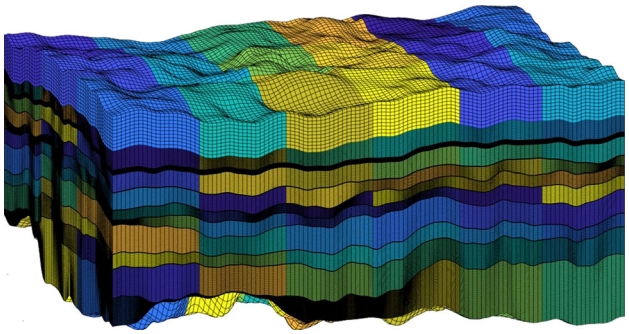


Fig. 7 The 2019 Sleipner benchmark model. The colors show a structured partition made up of 1632 coarse blocks plotted as varying colors. Note the thin black stripes, comprised of finely gridded shales that impede vertical migration of CO_2 and separate the rock volume into different flow compartments

compartment, with a fraction of the plume migrating horizontally and a fraction penetrating into the overlying shale, and so on.

The simulation model uses an immiscible, compressible two-phase description of the CO_2 brine system, but is otherwise identical to the case released by OPM. To partition the model, we subdivide it into a 7×13 coarse grid in the lateral direction. This single-layered partition is then vertically split by the flow compartments so that each new coarse block is entirely made up of either shale or high permeable sands. In total, this gives 1632 coarse blocks.

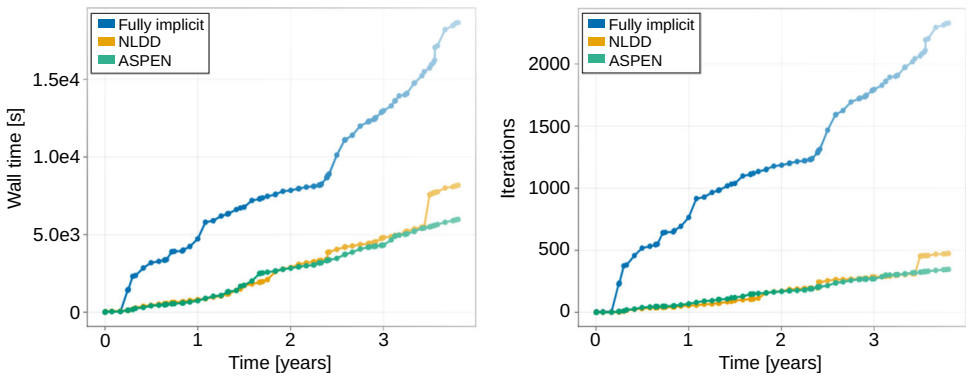
We simulate the case using standard Newton and ASPEN, both using a single thread. We observe, in Fig. 8, that ASPEN again gives significant improvements in the computational performance by reducing the number of iterations by almost a factor three. As before, each ASPEN iteration remains more costly than the standard Newton counterpart, however, and the total speedup factor is just over 1.8, going from 27.6 hours runtime for Newton to 15.2 hours for ASPEN.

4 Concluding remarks and outlook

Through a large number of numerical experiments, some of which are reported herein and in [12, 13], we have observed that ASPEN is more robust and not only manages to reduce the number of nonlinear iterations, especially in cases with localized, strong, and unbalanced nonlinearities, but also manages to solve time steps for which Newton struggles. Through the use of ASPEN (or variants thereof), one can therefore hope to use longer timesteps and reduce the number of unwanted timestep cuts in the presence of severe nonlinearities.

Each step with ASPEN is on the other hand more expensive and hybrid strategies that use ASPEN for "difficult" steps and Newton for all other would be advantageous. We presented two possible strategies in Case 2, but these are mere illustrations of the possibilities that lie in combining the robustness of ASPEN with the efficiency of Newton. It is not difficult to imagine more sophisticated strategies that, e.g., switch to ASPEN instead of cutting the time step if Newton fails (with the upper iteration limit not set too high) or start with one iteration with ASPEN, switch to Newton, but then switch back to ASPEN if Newton does not work well, etc. It is also possible to compare the convergence in nor-

Fig. 6 Computational performance test for the real asset model with standard Newton, NLDD, or ASPEN as nonlinear solvers. The left plot shows cumulative run time and the right plot the cumulative number of global iterations used per time step



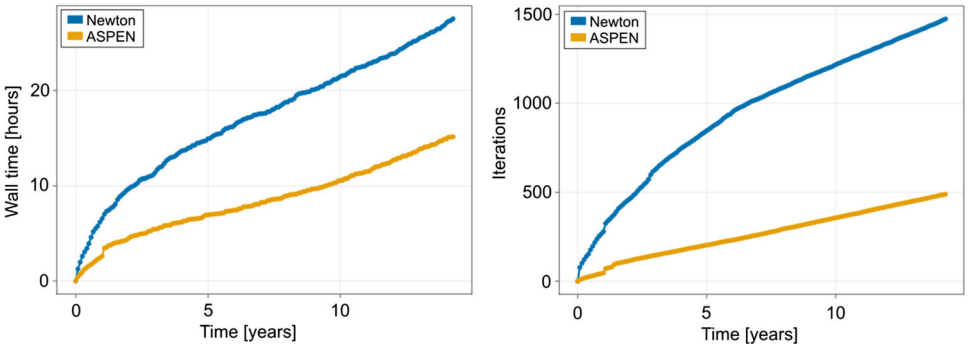


Fig. 8 Computational performance test for the Sleipner CO_2 storage model, simulated by a fully implicit method with either standard dampened Newton or ASPEN as nonlinear solver. The left plot shows cumulative runtime and the right plot the cumulative number of global iterations used per time step

malized mass-balance and CNV residuals, and, e.g., use the ratio between the two and the magnitude of the CNV residual to switch from Newton to ASPEN (or back). Switching between the two methods within a single time step is not problematic since the two use equivalent residuals.

We have yet not systematically investigated to what extent the improved robustness of ASPEN (or similar methods) can be used to take longer time steps to reduce computational time. For optimal performance, one needs to find the sweet spot among the counteracting factors of reduced runtime from fewer time steps and increased runtime from more iterations per step and reduced accuracy for longer steps. Likewise, use of inexact methods to avoid "over-solving", i.e., wasting iterations by solving states that are far from a converged solution with tight tolerances, has been discussed by Jiang et al. [11] for the sequentially fully implicit method. Similar considerations likely apply to other nonlinear decomposition methods and should be investigated.

Acknowledgements The authors thank TotalEnergies EP Norge AS for funding and TotalEnergies for the permission to publish this research. The Sleipner model is published by Equinor ASA on behalf of the Sleipner Group. The specific version used herein is based on the dataset available from the Open Porous Media initiative.

Funding Open access funding provided by SINTEF

Declarations

Competing interests The research leading to the results reported in this manuscript received partial funding from TotalEnergies EP Norge AS. Arthur Moncorgé is employed by TotalEnergies SE. Apart from this, the authors have no financial or competing interests to declare that are relevant to the content of this article.

References

1. Cai, X.-C., Keyes, D.E.: Nonlinearly preconditioned inexact Newton algorithms. SIAM J. Sci. Comput. **24**(1), 183–200 (2002). https://doi.org/10.1137/S106482750037620X
2. Liu, L., Keyes, D.E.: Field-split preconditioned inexact Newton algorithms. SIAM J. Sci. Comput. **37**(3), 1388–1409 (2015). https://doi.org/10.1137/140970379
3. Skogestad, J.O., Keilegavlen, E., Nordbotten, J.M.: Domain decomposition strategies for nonlinear flow problems in porous media. J. Comput. Phys. **234**, 439–451 (2013). https://doi.org/10.1016/j.jcp.2012.10.001
4. Liu, L., Keyes, D.E., Sun, S.: Fully implicit two-phase reservoir simulation with the additive Schwarz preconditioned inexact Newton method. In: SPE Reservoir Characterization and Simulation Conference and Exhibition, 16-18 September, Abu Dhabi, UAE (2013)https://doi.org/10.2118/166062-MS
5. Luo, L., Cai, X.-C., Keyes, D.E.: Nonlinear preconditioning strategies for two-phase flows in porous media discretized by a fully implicit discontinuous Galerkin method. SIAM J. Sci. Comput. , 317–344 (2021). https://doi.org/10.1137/20M1344652
6. Dolean, V., Gander, M.J., Kheriji, W., Kwok, F., Masson, R.: Nonlinear preconditioning: How to use a nonlinear Schwarz method to precondition Newton's method. SIAM J. Sci. Comput. **38**(6), 3357–3380 (2016). https://doi.org/10.1137/15M102887X
7. Li, J., Wong, Z.Y., Tomin, P., Tchelepi, H.: Sequential implicit Newton method for coupled multi-segment wells. In: SPE Reservoir Simulation Conference, Galveston, Texas, USA, April 2019 (2019). https://doi.org/10.2118/193833-MS
8. Li, J., Tomin, P., Tchelepi, H.: Sequential fully implicit Newton method for compositional flow and transport. J. Comput. Phys. **444**, 110541 (2021). https://doi.org/10.1016/j.jcp.2021.110541
9. Jiang, J., Tchelepi, H.A.: Nonlinear acceleration of sequential fully implicit (SFI) method for coupled flow and transport in porous media. Comput. Methods App. Mech. Eng. **352**, 246–275 (2019). https://doi.org/10.1016/j.cma.2019.04.030
10. Zhou, Y., Jiang, J., Tomin, P.: Inexact methods for black-oil sequential fully implicit SFI scheme. In: SPE Reservoir Simulation Conference, On-Demand, October 2021 (2021). https://doi.org/10.2118/203900-MS
11. Jiang, J., Tomin, P., Zhou, Y.: Inexact methods for sequential fully implicit (SFI) reservoir simulation. Comput. Geosci. **25**(5), 1709–1730 (2021). https://doi.org/10.1007/s10596-021-10072-z
12. Klemetsdal, Ø., Moncorgé, A., Møyner, O., Lie, K.-A.: A numerical study of the additive Schwarz preconditioned exact Newton

method (ASPEN) as a nonlinear preconditioner for immiscible and compositional porous media flow. Comput. Geosci. **26**, 1045–1063 (2022). https://doi.org/10.1007/s10596-021-10090-x
13. Klemetsdal, Ø.S., Moncorgé, A., Nilsen, H.M., Møyner, O., Lie, K.-A.: An adaptive sequential fully implicit domain-decomposition solver. SPE J. 27(01), 566–578 (2021). https://doi.org/10.2118/203991-PA
14. Lie, K.-A.: An Introduction to Reservoir Simulation Using MATLAB/GNU Octave: User Guide for the MATLAB Reservoir Simulation Toolbox (MRST). Cambridge University Press (2019). https://doi.org/10.1017/9781108591416
15. Luo, L., Cai, X.-C., Yan, Z., Xu, L., Keyes, D.E.: A multilayer nonlinear elimination preconditioned inexact Newton method for steady-state incompressible flow problems in three dimensions. SIAM Journal of Scientific Computing **42**(6), 1404–1428 (2020). https://doi.org/10.1137/19M1307184
16. Lie, K.-A., Møyner, O. (eds.): Advanced Modeling with the MATLAB Reservoir Simulation Toolbox. Cambridge University Press, (2021). https://doi.org/10.1017/9781009019781
17. Rasmussen, A.F., Sandve, T.H., Bao, K., Lauser, A., Hove, J., Skaflestad, B., Klöfkorn, R., Blatt, M., Rustad, A.B., Sævareid, O., Lie, K.-A., Thune, A.: The Open Porous Media Flow reservoir simulator. Comput. Math. Appl. **81**, 159–185 (2021). https://doi.org/10.1016/j.camwa.2020.05.014
18. Younis, R.M.: Modern advances in software and solution algorithms for reservoir simulation. PhD thesis, Stanford University (2011)
19. Zhou, Y.: Parallel general-purpose reservoir simulation with coupled reservoir models and multisegment wells. PhD thesis, Stanford University (2012)
20. Wallis, J.R., Kendall, R.P., Little, T.E.: Constrained residual acceleration of conjugate residual methods. In: SPE Reservoir Simulation Symposium, Dallas, Texas, February 1985 (1985). https://doi.org/10.2118/13536-MS
21. Gries, S., Stüben, K., Brown, G.L., Chen, D., Collins, D.A.: Preconditioning for efficiently applying algebraic multigrid in fully implicit reservoir simulations. SPE J. **19**(04), 726–736 (2014). https://doi.org/10.2118/163608-PA
22. Fonseca, R.M., Rossa, E.D., Emerick, A.A., Hanea, R.G., Jansen, J.D.: Introduction to the special issue: Overview of OLYMPUS optimization benchmark challenge. Comput. Geosci. **24**(6), 1933–1941 (2020). https://doi.org/10.1007/s10596-020-10003-4