Enhanced decoupling and CPR-FSAI preconditioner for fully implicit reservoir simulations in OPM

RESEARCH

Enhanced decoupling and CPR-FSAI preconditioner for fully implicit reservoir simulations in OPM

Artem Mavliutov[1,2] · Andrea Franceschini[2] · Carlo Janna[2,3]

Received: 2 April 2026 / Accepted: 16 July 2026 / Published online: 11 August 2026

Abstract
Efficient and scalable linear solvers are critical for implicit reservoir simulation, where the linear solver can account for up to 90% of total runtime. In this work, we present an algorithmic framework that replaces the inherently sequential ILU stage of the popular CPR preconditioner with a highly parallel FSAI preconditioner enhanced by an augmented decoupling mechanism. To improve FSAI in the presence of strong transport-induced couplings, a local block-diagonal decoupling is applied on small cell-blocks. The resulting fully local decoupling approach combines quasi-IMPES scaling to reduce pressure-saturation couplings, a dynamic row summation to ensure solvability by AMG, and constrained pressure decoupling to improve FSAI preconditioning effect. This yields a highly effective and scalable CPR preconditioning framework. We implemented the preconditioned solver suite in C++/MPI and evaluated it with the OPM simulator on Norne, SPE11C, and Sleipner benchmarks. The resulting framework, *deco*, matches or improves upon default OPM solvers (*DUNE* and *AMGCL*) in a sequential setting (1 MPI rank) and delivers 2–4× speedups in strong-scaling tests with up to 2048 MPI ranks on a single domain.

Keywords Multi-phase flow · Decoupling · Preconditioning · HPC · FSAI · CPR

1 Introduction

In a typical reservoir simulation, the linear system solution is the most time-consuming task, reaching up to 90% of the simulation time [11, 18, 26, 37, 55]. This is due to the fact that the reservoir simulations are typically run for a considerable amount of time with relatively small time steps (especially for challenging problems), and each of them requires the solution of a nonlinear system of equations. Due to the nonlinear nature of the unknowns, Newton-type methods are primarily used, which furthermore produce a sequence of linear system of the form $Ax = b$. These systems are the main bottleneck of the overall simulation process. Since these systems are large, relying on a direct solver is not feasible; as a result, iterative solvers based on Krylov subspace methods became a standard approach for this problem. However, even the most efficient solvers, such as GMRES and BiCGstab, largely depend on the quality of the preconditioner whose choice essentially dictates the convergence rate of the iterative solver. Therefore, the mainstream research focus is targeting the choice and practical implementation aspect of the preconditioning.

The linear systems arising from reservoir simulations have a mixed-physics character because of the coupling of different equations, such as diffusion and transport, which are guided by pressure and saturation, respectively [15, 33, 51]. The pressure (diffusion) equation behaves almost like an elliptic problem, while the transport equation

Andrea Franceschini and Carlo Janna contributed equally to this work.

exhibits a hyperbolic nature. This fundamental distinction strongly influences the design of efficient solvers. In order to take advantage of the natural partitioning of the problem into elliptic (pressure-dominated) and hyperbolic (transport-dominated) components, the Constrained Pressure Residual (CPR) preconditioner [54] was developed. It features two main stages that are combined through a decoupling process: (i) the local smoother (preconditioner) to handle the pressure equation and (ii) the global one serving to relax the transport part and ensure robustness of the full preconditioning approach. In its original formulation, CPR is a two-stage preconditioner, featuring a sequential preconditioning step for each part of the system. In particular, a well-established strategy for the pressure block is the Algebraic Multigrid (AMG) preconditioner [52], while for the global system, typically an incomplete LU (ILU) preconditioner is used [10].

In recent years, a lot of CPR variants have appeared. For example, approaches with different orderings of the stages [55], special treatment of well equations [41], block preconditioning approaches [38, 40], addition of new phases/variables [14, 46, 47], and coupling with poromechanics [8, 9, 56] have improved reservoir simulation process. However, the classical isothermal CPR setting is still the most common modeling choice. In this setting, the most common approaches for decoupling remain local following the IMPES scheme [15] employing quasi-IMPES weights [33, 51] or true-IMPES weights [32, 44]. The pressure IMPES equation is obtained following DRS filtering to improve AMG convergence [23]. These types of decoupling are implemented in state-of-the-art libraries *DUNE* [3] and *AMGCL* [16] and are available directly from OPM Flow.

Despite substantial advances in preconditioning methodology, the majority of the CPR implementations heavily rely on the well-established components of the CPR framework, namely the AMG preconditioner for the elliptic part, and the ILU preconditioner for the transport part.

While ILU is often the preferred preconditioner for hyperbolic problems [20, 39], its scalability remains a major limitation in many engineering applications. Despite significant progress in multilevel and multifrontal ILU implementations, the algorithm is inherently sequential in both the setup and application phases. In particular, the forward and backward triangular solves required during application are intrinsically sequential, making efficient parallelisation challenging and costly [7]. Another difficulty arises from matrix reordering required to minimize fill-in and extract parallelism in ILU setup. Reordering can significantly affect the iteration count compared to the original matrix, especially for ILU0 implementations [4, 49]. Although reordering and multilevel frameworks can improve scalability during the setup phase, they often reduce the quality of the resulting preconditioner relative to that obtained from the original matrix. This adds a substantial overhead, considering that the required linear solver tolerance is by default set to 10^{-2} or 10^{-3}. For these reasons, approximate inverse methods seem to be a better choice, as they are naturally parallel, even though they typically reduce iteration counts less effectively than ILU methods of comparable numerical complexity.

This work proposes to use an approximate inverse preconditioner, in particular the Factorized Sparse Approximate Inverse (FSAI) [57], as a global system relaxation scheme within the CPR framework. Parallel approximate inverse techniques [12] such as FSAI [57] and SPAI [24] have become quite popular choices for preconditioning. Their main advantage with respect to other preconditioners is their practically ideal suitability for being applied in parallel, as they involve only sparse matrix-by-vector products (SpGEMV). The setup phase is typically parallel as well, as it requires the solution of small independent problems that can be efficiently performed either in SIMD or MIMD fashion, or their mix. To the best of our knowledge, FSAI has not yet been used as a global preconditioner within the CPR framework in the context of reservoir simulation. The relaxation property of FSAI with respect to the ILU preconditioner is generally inferior (considering the same complexity of application of the preconditioner in terms of nonzeros). Moreover, since ILU is a more robust and much more popular preconditioning choice, it has become the default global relaxation option. However, FSAI has attracted growing interest over the past decade with the rise of supercomputers and GPUs in scientific computing [2, 29]. These advances enable trading additional FLOPS for preconditioner quality without substantial overhead. Given the current trend in reservoir simulation toward GPU-based workflows [1, 17, 22, 36, 43], FSAI is particularly attractive as it is inherently more GPU-friendly [5, 6, 28] than ILU.

In order to make FSAI amenable to reservoir simulation problems, different decoupling strategies have been analyzed. It was observed that all the reservoir models analyzed in this work include strong block diagonal influence. Therefore, eliminating this influence altogether in a cheap and parallel way would improve any kind of preconditioner. In particular, it was noticed that decoupling locally (on a cell level with FVM discretization) the main diagonal enhances the performance of the global FSAI preconditioner and consequently the overall CPR strategy is improved. This local decoupling, followed by FSAI preconditioning, bears close similarity to block FSAI preconditioner [21]. The main motivation behind this step is the elimination of the strong connections on the main cell-block diagonal coming from the transport equation. This decoupling step follows the Constrained Pressure Decoupling (CPD) approach [34] and is actually a complementary one to the well-established quasi-IMPES [33] decoupling with DRS weights [23]. Therefore, the proposed decoupling process is a 3-step approach, combining quasi-IMPES, DRS, and CPD methods.

Novelty of this work comes from an improved decoupling framework that can be used with any global smoother. Secondly, this work features the FSAI preconditioner being used as a global relaxation scheme for reservoir simulations for the first time. The proposed methodology has been developed in an open-source C++ package called *deco*.

The paper is organized as follows: Section 2 introduces the CPR preconditioning framework, which is followed by the FSAI preconditioner description in Section 3. Furthermore, in Section 4, different decoupling strategies are described and how they are implemented in the proposed framework. Section 5 presents the numerical experiments that have been conducted to justify the proposed methodology. Section 6 summarizes the results of the proposed package *deco* against baseline libraries and provides future directions.

2 CPR preconditioner

The ultimate goal is to solve linear systems of the form:

$$A = \begin{bmatrix} A_{pp} & A_{ps} \\ A_{sp} & A_{ss} \end{bmatrix} \tag{1}$$

where p is the pressure index subscript and s is the saturation variable subscript. The matrix A corresponds to the Jacobian matrix obtained by differentiating the nonlinear residual equations with respect to the unknowns and reduced to the primary variables only, namely oil-pressure and water-saturation.

The CPR preconditioner is a two-stage multiplicative process of the form below, designed specifically for fully implicit reservoir simulation:

$$M_{CPR}^{-1} = M_G^{-1}\left[I - AI_p M_p^{-1} I_p^T G\right] + I_p M_p^{-1} I_p^T G \tag{2}$$

where M_G^{-1} is the global relaxation matrix, $M_p^{-1} \approx \left(I_p^T G A I_p\right)^{-1}$ is the local relaxation matrix in which I_p is the restriction operator to the pressure space, and G is the decoupling matrix. This operation involves one pressure space correction followed by a global correction step. If instead we add an extra global correction step at the beginning, we will obtain a standard two-grid V-cycle AMG operator with a custom prolongation.

3 FSAI preconditioner

The FSAI (Factorized Sparse Approximate Inverse) preconditioner for a general matrix A has the following form:

$$M_G^{-1} = F_U F_L \tag{3}$$

The triangular factors F_L (lower) and F_U (upper) from Eq. 3 are obtained by finding a stationary point of the bilinear functional, which is equivalent to minimizing the residual in nonsymmetric case:

$$\Phi(F_L, F_U) = \mathrm{Tr}[(I - F_L L)(I - U F_U)] \tag{4}$$

where A is assumed to have a factorization form such that $A = LU$.

Minimization of Eq. 4 gives rise to the following set of equations:

$$A[\mathcal{S}_i^L, \mathcal{S}_i^U] f_i^U = -A[\mathcal{S}_i^L, i] \qquad i = 2, ..., n \tag{5}$$

$$A^T[\mathcal{S}_j^U, \mathcal{S}_j^L] f_j^L = -A^T[\mathcal{S}_j^U, j] \qquad j = 2, ..., n \tag{6}$$

where n is the system size, $\mathcal{S}^L$ is the prescribed sparsity pattern of F_L and $\mathcal{S}^U$ is the prescribed sparsity pattern of F_U; $\mathcal{S}_k^L$ and $\mathcal{S}_k^U$ are the set of indices in the k-th row or column of $\mathcal{S}^L$ and $\mathcal{S}^U$ respectively.

The systems Eqs. 5–6 are solved independently for f_j^L and f_i^U for each i column and j row. The solutions are then scattered into the corresponding row and column entries of $F_L[j, :] = [(f_j^L)^T, 1]$ and $F_U[:, i] = [(f_i^U)^T, 1]^T$ respectively. These $2(n-1)$ systems Eqs. 5–6 are completely independent and therefore can be set up in parallel.

In this work, we are using a static version of FSAI [29] in which the combined nonzero pattern of F_L and F_U is the same as that of A. Algorithmically, the static FSAI preconditioner's setup is described in the Algorithm 1. Essentially, the static FSAI computed on the original matrix pattern tries to decouple the original matrix up to a diagonal matrix, without compensating for any fill-in (which ILU does in contrast) following a local Gaussian elimination process. In order to strengthen the preconditioner after this inverse factorization stage, a diagonal scaling is applied. Since the matrix A might have some small off-diagonal couplings, including them in the pattern only adds an overhead without any benefit; therefore, a prefiltering stage is set to eliminate these small entries. In the same way, a postfiltering stage is also implemented to remove the dead load of small entries. The tolerance is set based on the relative diagonal threshold and it is reported in the parameters 14.

Algorithm 1 $[F_L, F_U] = \mathrm{FSAI}(A)$.

1: $\tilde{A} = \mathrm{prefilter}(A)$ ▷ filter the input matrix
2: $\mathcal{S} = \mathrm{pattern}(\tilde{A})$ ▷ generate the preconditioner's pattern (initial)
3: **for** $i = 2, \ldots, n-1,$ **do**
4: $\quad \tilde{A}[\mathcal{S}_i, \mathcal{S}_i]\ F_U[\mathcal{S}_i, i] = -\tilde{A}[\mathcal{S}_i, i]$ ▷ solve two local system
5: $\quad \tilde{A}[\mathcal{S}_i, \mathcal{S}_i]^T F_L[i, \mathcal{S}_i] = -\tilde{A}[i, \mathcal{S}_i]^T$ ▷ with only one LU factorization
6: **end for**
7: $[F_L, F_U] = \mathrm{postfilter}(F_L, F_U)$ ▷ remove small entries
8: $D = \mathrm{diag}((F_L \cdot \tilde{A} \cdot F_U))$ ▷ matrix by matrix product, diagonal output only
9: $[F_L, F_U] = \mathrm{scale}(F_L, F_U, D)$ ▷ diagonal scaling

The application stage of FSAI involves multiplying a vector first by a sparse lower triangular matrix, followed by multiplication with a sparse upper triangular matrix Eq. 3.

4 Decoupling

The main goal of the decoupling operator G is to reduce the strength of the pressure-saturation coupling of A_{ps} in a cost-effective manner. The matrix G can be applied during the application phase, as is shown in the equation (2), or it can be applied before invoking the linear solver. In the latter case, we are actually solving an approximate Schur complement system reduced to the pressure space, and we assume that a fully-implicit solution and an IMPES

approximation are close to each other [13]. It is quite common to solve for the approximate Schur-complement system, and in fact, it is done by default in OPM and MRST, as well as in many other works such as [23, 33, 51, 55]. In this case, the decoupling matrix is cheap and operates on the local cells only; therefore, applying it before invoking the linear solver is possible and adds fill-in only at the cell block couplings already present in the original adjacency graph of A. Instead, when the decoupling is applied inside the preconditioner matrix [38, 46], typically the Schur complement approximation is more accurate but also costly. As a result, it can only be applied at the preconditioner level. The proposed decoupling approach operates at the cell level only; consequently, the decoupling operator is applied beforehand:

$$A\,\mathbf{x} = \mathbf{b} \quad \longrightarrow \quad (GA)\,\mathbf{x} = (G\,\mathbf{b}) \tag{7}$$

The decoupling is the most crucial part of an effective CPR preconditioning framework. In the developed library *deco*, different decoupling approaches are implemented; each of them can be used on its own or combined together. The actual decoupling procedure is outlined in the Algorithm 2.

Algorithm 2 $[\tilde{A}, \tilde{\mathbf{b}}] = \text{decouple}(A, \mathbf{b})$.

1: $[A_r, \mathbf{b}_r] = \text{reorder}(A, \mathbf{b})$ ▷ reorder rows locally
2: $\mathbf{w}_{qI}$ **= qIMPES**(A_r) ▷ compute quasi-IMPES weights
3: $[A_r^w, \mathbf{b}_r^w] = \text{diag\_scale}(A_r, \mathbf{b}_r, \mathbf{w}_{qI})$ ▷ scale the system with $\mathbf{w}_{qI}$
4: $\mathbf{w}_{drs}$ **= DRS**(A_r^w) ▷ compute DRS weights
5: $[\bar{A}_r^w, \bar{\mathbf{b}}_r^w] = \text{IMPES}(A_r^w, \mathbf{b}_r^w, w_{drs})$ ▷ sum pressure aligned parts $\in \mathbf{w}_{drs}$
6: $[D_{cpd}]$ **= CPD**$(\bar{A}_r^w)$ ▷ compute CPD block diagonal matrix
7: $[\tilde{A}, \tilde{\mathbf{b}}] = \text{apply\_CPD}(\bar{A}_r^w, \bar{\mathbf{b}}_r^w, D_{cpd})$ ▷ apply CPD matrix

Any local decoupling process requires that the main diagonal of the system matrix contains only nonzero entries. Therefore, *deco* begins by reordering the linear system so that the main diagonal is nonzero. This reordering step is inexpensive, parallelizable, and adds negligible overhead, yet it significantly strengthens the main diagonal—an essential feature for both decoupling and preconditioning.

quasi-IMPES decoupling The first step of Algorithm 2 involves the diagonal scaling with the quasi-IMPES weights [51]. The weights are computed in such a way that when the IMPES pressure system is formed, its pressure-saturation coupling is reduced:

$$[G]_{ii} = \begin{bmatrix} 1 & -a_{ps}a_{ss}^{-1} \\ 0 & 1 \end{bmatrix}_{ii} \qquad \text{where} \qquad i = 1, ..., n \tag{8}$$

In this way, the approximate Schur complement pressure system results in

$$\tilde{S}_{pp} = GA = A_{pp} - \text{diag}(A_{ps})\text{diag}(A_{ss})^{-1}A_{sp} \tag{9}$$

where $\text{diag}(M)$ is the operator that extracts the diagonal from square matrix M. After obtaining the quasi-IMPES weights, the full system is scaled with the computed coefficients (since this decoupling operator also has a mild preconditioning effect [31]) as opposed to using the weights only for the computation of the IMPES pressure system.

DRS weights In the subsequent step, the DRS (Dynamic Row Sum) weights [23] are computed, which essentially act as a filter to the obtained Schur complement matrix. It is known that the original pressure block A_{pp} can be indefinite with negative eigenvalues related to the increased number of active wells present at the particular time step [23]. Moreover, the indefiniteness of the pressure block may be due to the appearance and disappearance of

the phase at the particular time step [8]. This destroys the elliptic properties of the pressure system for which AMG was originally designed for [52]. As a remedy, the entries of the IMPES weights are allowed to be dynamic:

$$[G]_{ii} = \begin{bmatrix} \delta_1^i & \delta_2^i \\ 0 & 1 \end{bmatrix} \qquad \text{where} \qquad i = 1, ..., n \tag{10}$$

in which δ_x^i for each cell i and the equation x are computed to satisfy two constraints, namely

$$\begin{aligned} \textbf{M-matrix property:} \quad \delta_x^i &= \begin{cases} 0 & \text{if} \quad \left[A_{x,1}\right]^{i,i} < \varepsilon_{dd} \sum_{j=1, j\neq i}^{n} \left|\left[A_{x,1}\right]^{i,j}\right| \\ 1 & \text{else} \end{cases} \\ \textbf{Weak coupling:} \quad \delta_x^i &= \begin{cases} 0 & \text{if} \quad \sum_{j=1}^{n} \left|\left[A_{1,x}\right]^{i,j}\right| < \varepsilon_{ps} \left|\left[A_{1,1}\right]^{i,i}\right| \\ 1 & \text{else} \end{cases} \end{aligned} \tag{11}$$

where ε_{dd} and ε_{ps} are constant parameters that control the extent of violation of these properties.

The DRS weights ensure that the pressure matrix remains solvable with AMG, by keeping the equations used to form the Schur complement that do not severely violate the ellipticity of the pressure matrix. Moreover, the weak pressure saturation couplings are exempt from forming the Schur complement to eliminate the noise (no need to decouple what is already weakly coupled). In the third step, we finally sum up the IMPES scaled pressure-aligned equations that pass the DRS filter.

CPD decoupling The final step imposes an additional Constrained Pressure Decoupling (CPD) [33] which can be used as a stand-alone decoupling process [34] but in our case it strengthens the global smoothing phase of FSAI:

$$G_{ii} = I + \sum_{k=1:bs} \mathbf{e}_k \mathbf{e}_k^T \left([A_{pp}]_{ii} [A]_{ii}^{-1} - I \right) \tag{12}$$

where the block diagonal decoupling matrix G is computed for each cell i, with block size bs corresponding to the number of phases. $\mathbf{e}_k = [\delta_{1,k}\ \delta_{2,k}\ \ldots\ \delta_{bs,k}]^T \in \mathbb{R}^{bs}$ (where $\delta_{i,j} = 1$ if $i = j$, else 0) the identity matrix $I \in \mathbb{R}^{bs}$.

In reservoir simulations, large coefficients often appear in the block-diagonal part of the matrix due to transport-dominated flow [33, 51]. Removing this influence is crucial for the global smoother or FSAI in particular. For example, an application of FSAI to a linear system by design approximately decouples the system up to a diagonal matrix, which is then scaled with the inverse diagonal matrix. However, if large off-diagonal block entries remain (from an algebraic multigrid perspective, this indicates strong connections in the adjacency graph of the matrix), a standard FSAI, in general, will struggle to smooth them adequately.

While block FSAI [21] (with the inner preconditioner set on the $bs \times bs$ block diagonal part) could address this, however, this type of 'decoupling' could only be applied at the application stage and would involve considerable setup and application costs, instead, the CPD decoupling followed by FSAI is conceptually a reversed-order block FSAI, producing the effectiveness of block FSAI approach while keeping the application and setup costs minimal.

5 Experiments

In this section, we will analyze the performance of the proposed kernel on selected benchmarks from OPM Flow during fully-implicit black-oil and CO_2 storage simulations. In doing so, we run two sets of tests:

(i) first, the simulation is run sequentially (1 CPU core with 1 MPI rank and 1 OpenMP thread) with the *CPR:OPM:DUNE* solver since it is used by default and shows slightly better performance among the available

solvers from OPM. This approach uses the decoupling and the CPR implemented in OPM, while the solution mechanisms (ILU, AMG, linear solver) are directly called from *DUNE* [3]. During the simulation with *CPR:OPM:DUNE*, the Jacobian matrices corresponding to Newton iterations that have been reduced to cell variables only (usually at this step, a linear solver is invoked) are dumped to the disk. The dumped matrices are then used to benchmark the linear solver of *deco* in a serial fashion with one processor on the same set of matrices that have been used with *CPR:OPM:DUNE*. For the sake of clarity and easier comparison, when running the simulation with OPM, the preconditioner is rebuilt every time for each linear system to be solved. In practice, however, the rebuilt interval is controlled to reduce the amount of computations needed to set up the preconditioner. This allows us to reuse the preconditioner for the subsequent systems, which are usually similar to the previous iterations.

The main challenge arises from the fact that *CPR:OPM:DUNE* and *CPR:AMGCL* employ ILU0 both as a global preconditioner and as a smoother for the pressure subsystem. Since ILU is a well-established and robust method with strong smoothing properties, it provides a particularly effective CPR pairing. In contrast, our goal is to replace ILU with FSAI, which is inherently more parallel but exhibits weaker smoothing properties at comparable computational complexity. The difficulty, therefore, is to ensure that the resulting CPR–FSAI combination remains competitive with the CPR–ILU approach, even in single-processor settings.

(ii) second, because *CPR:OPM:DUNE* does not have stand-alone API's for running the CPR preconditioned solver on its own, the parallel scalability and efficiency are also tested with *AMGCL* [16] on selected matrices. The goal here is to show the scalability of the proposed methodology. Here a matrix is selected for each model to represent a typical behavior of the linear solver throughout the simulation; moreover, the iteration count in a one-processor setting is matched for an easier interpretation of the results. In this case, the system matrices correspond to a single spatial domain, with the MPI parallelism used to distribute the linear system on different computing cores.

In the first set of tests, we are using the BiCGstab solver since it comes by default in *DUNE* and uses the right preconditioned version (the exit test is done on true residual); the same solver is used in *deco*. However, when investigating the scalability in the second set of tests, we are using the right preconditioned GMRES algorithm in both *AMGCL* and *deco* since BiCGstab is less robust and is more sensitive to fluctuations due to parallel reduction. Unfortunately, GMRES cannot be used on *DUNE* because this package only provides its left preconditioned version while all the other solvers are right preconditioned. A left preconditioned solver has an exit test based on the preconditioned residual and cannot be fairly compared to right preconditioned solvers.

In the following, we will operate with the setup time, i.e., the wall-clock time (in seconds) required to build the preconditioner. The solution time, in turn, is the time required to solve the linear system arising from the linearization step of the Newton solver, assuming that the system has already been reduced to the cell variables only. Usually, a high relative tolerance on the residual (around 10^{-3} or 10^{-2}) is enough to achieve the convergence of the nonlinear solver in reservoir simulations.

The proposed CPR preconditioner and decoupling strategy is implemented in *deco* library. We are using a static version of FSAI [5] preconditioner from Chronos [27, 30], and BoomerAMG [25] from Hypre [19]. In the Appendix, the default setups of the linear solvers of *OPM:CPR:DUNE* (Fig. 12), *CPR:AMGCL* (Fig. 13), and *CPR:deco* (Fig. 14) are provided. To the best of our knowledge, the parameters used here are the most optimal and commonly used in reservoir simulators such as OPM and MRST.

As benchmark problems we select four reservoir models, including both real-world and synthetic cases, often used in scientific papers on this topic. The name, the number of unknowns, the number of cells and the number of non-zeroes of the corresponding system matrices are reported in Table 1.

Norne

The Norne benchmark case [48] is a real-field, fully implicit black-oil model of an oil field in the Norwegian Sea, widely used in research, simulator verification, and HPC studies [41, 45]. It is particularly valuable because it combines realistic geological complexity with computational challenges representative of large-scale industrial simulations. In addition, it is freely available from github.com/OPM. The model includes 36 wells—producers and injectors—distributed across the field, though not all are active simultaneously, with well controls and completions

Table 1 System matrices used in the numerical experiments. For each matrix the table provides the number of unknowns, the number of cells and the number of non-zeroes.

Name	# of unknowns	# of cells	# of non-zeroes
Norne	133,293	44,431	2,883,771
SPE11C	774,000	258,000	16,045,506
Sleipner	3,972,352	1,986,176	55,169,584
SPE11C refined	45,576,000	15,192,000	953,257,680

evolving over the simulation period to match historical operations. The model is discretized over 44,431 cells with corner grid geometry; the model has 3 phases (oil, water, and gas). The horizontal permeability of the reservoir is provided in Fig. 1).

Table 2 shows the end metrics of the entire black-oil simulation performed with one processor (1 MPI rank), focusing on the linear solver part. We observe that the proposed methodology achieves good performance (23% boost in solution time and 9% boost in setup time) even with 1 MPI rank. The major improvements stem from the use of an augmented decoupling strategy together with a classical AMG framework for the pressure system. The use of classical AMG is typically more expensive and less scalable in the setup phase than aggregation-based AMG methods (*DUNE*, *AMGCL*), but it provides a richer coarse-space representation of the pressure operator (which can improve the effectiveness of the preconditioner for this subsystem) and achieves comparable scalability during the application phase. In this approach, we further emphasize the role of the pressure subsystem by applying a stronger and more effective preconditioning strategy to it. Consequently, greater emphasis is placed on the IMPES idea of the CPR preconditioner.

Figure 2 compares AMGCL and *deco* by varying the number of MPI ranks. We see that initially (1 MPI setting) the setup and solution times of both solvers are similar. Later, however, the setup of AMGCL has a better scalability curve, which is due to the fact that AMGCL ILU0 preconditioner in a distributed environment uses a simple block Jacobi partitioning between MPI domains, resulting in a close-to-ideal parallelism of the ILU0 factorization. However, this parallel efficiency is offset by increased solution time due to a deterioration in the preconditioner quality as the couplings between MPI domains are not incorporated. Since in practice the preconditioner is recycled over several Newton steps, the solution time offset using the distributed framework is more painful to the performance. Instead, *deco* linear solver iteration count stays the same with a slight variation

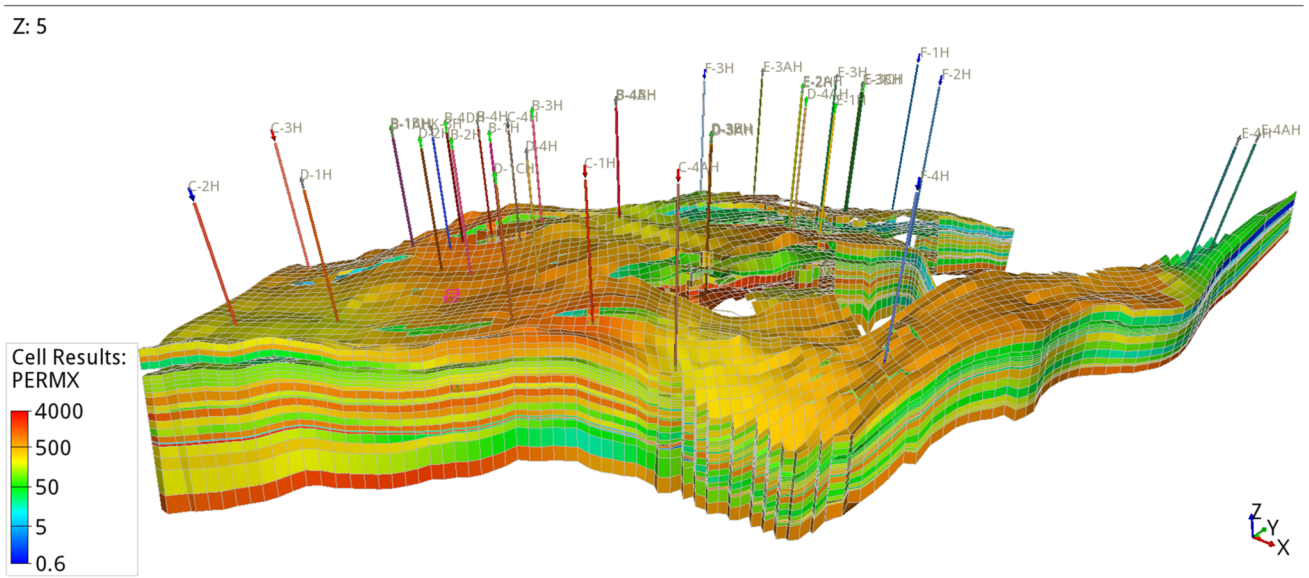

Fig. 1 Mesh grid, horizontal permeability, and well set-up of the Norne reservoir model. The z-direction has been exaggerated 5 times

Table 2 Simulation results for Norne model over 9 years of production using CPR-BiCGstab solver (relative residual tolerance 10^{-3}) in a total of 1,160 linear systems, *1 MPI rank*

	Total time [s]		Iterations	
	Setup	Solve	Total	Average
CPR:OPM:DUNE	285.2	173.2	6,788	5.8
CPR:deco	259.5	132.2	4,805	4.1

of the residual while producing scalability of both setup and solution time. In the final configuration (16 MPI ranks), we see a $2x$ speedup in the solution phase with just 17% degradation in the setup time.

To better understand the impact of the FSAI preconditioner in *deco*, in Fig. 3 we provide a comparison with *deco* where the global FSAI preconditioner is replaced by an ILU factorization. We adopted the ILUK implementation from hypre and use it with zero fill-in (ILU0). As in AMGCL, parallelism is obtained through a block Jacobi decomposition across MPI ranks.

At a single MPI rank, the ILU0 configuration requires only five iterations, compared with six iterations for the FSAI-based solver. However, the FSAI preconditioner maintains a constant iteration count as the number of MPI ranks increases because the preconditioner that is set-up is always the same regardless of the number of ranks. In contrast, the block Jacobi ILU0 approach exhibits a gradual increase in the number of iterations because the diagonal blocks becomes smaller with a degradation of the preconditioner quality.

Although the iteration count increases only modestly in this particular experiment, significantly greater degradation can be expected in practice for highly ill-conditioned matrices encountered during the simulation. Despite the small increase in iterations observed here, the FSAI-based solver still achieves a 50% speedup in the solution phase, which is attributed solely to the use of the FSAI preconditioner as alternative to ILU0.

SPE11C

SPE11C is a synthetically constructed CO_2 storage model designed to represent conditions typical of deep saline formations beneath the North Sea [42]. The model is freely available, for example, through [35]. The reservoir is discretized using a corner-point grid and modeled with three fluid phases: brine, CO_2, and fresh water. The CO_2 injection and multiphase flow processes are simulated using specialized CO_2–brine PVT properties to capture dissolution and phase behavior accurately [45], for details see SPE11C Benchmark. Because of its large size (258, 000 cells with 3 phases), complex geological features, multiphase flow physics, and a large range in the permeability field (from 0.00001 mD up to 2, 000 mD, see Fig. 4), SPE11C serves as a demanding benchmark for high-performance computing.

In Table 3, we see comparative results with two solvers during a simulation of the SPE11C model over a year of CO_2 injection; the simulation is computed with one processor (1 MPI rank). Both solvers perform similarly on this benchmark with slightly better results using *deco* (13% boost in the solution part, the speedup of the setup is

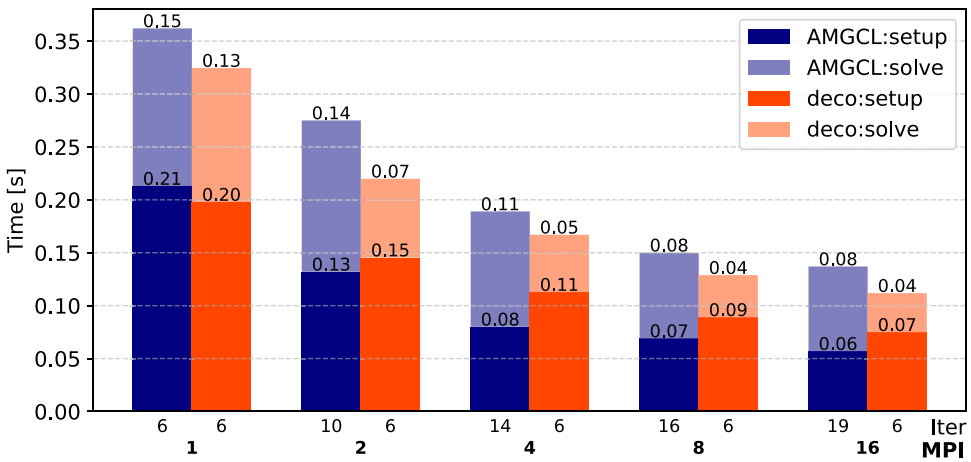

Fig. 2 Scaling of setup and solution time versus the number of MPI ranks for the Norne model. Bars show AMGCL (blue) and deco (orange) with CPR right-preconditioned GMRES solver at relative tolerance 10^{-3}. GMRES iteration counts are indicated below each bar, and MPI ranks are shown below

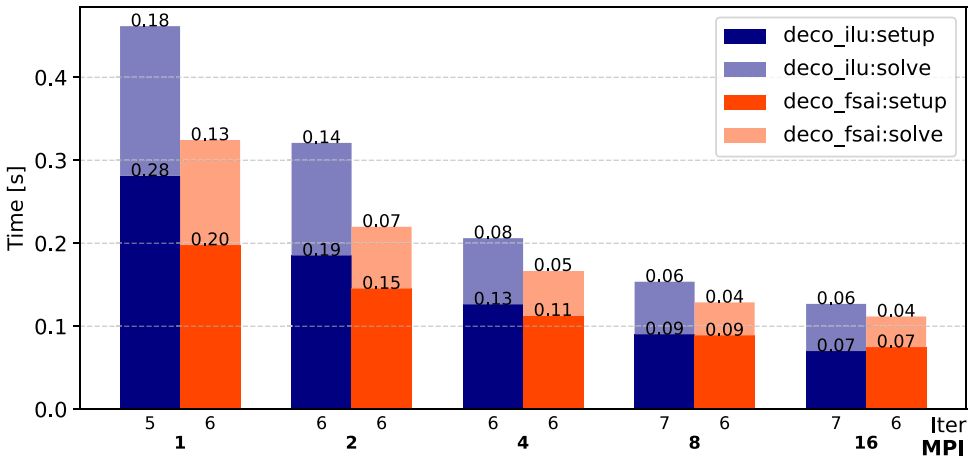

Fig. 3 Scaling of setup and solution time versus the number of MPI ranks for the Norne model. Bars show deco with ILU0 (blue) and deco with FSAI (orange) with CPR right-preconditioned GMRES solver at relative tolerance 10^{-3}. GMRES iteration counts are indicated below each bar, and MPI ranks are shown below

not so much important since in practical simulation the preconditioner will be reused for several linear systems). The CPR preconditioner works well on this model, and it is possible to adopt large time steps, resulting in just 22 linear systems for the 1-year sequestration process.

In Fig. 5, we see the scalability of the solution and setup time of two CPR preconditioned GMRES solvers with the same relative residual tolerance. Looking at the scaling comparison between AMGCL and deco, we can notice that the setup time of AMGCL is much larger than that of deco during the MPI configuration from 1 through 8. This is primarily due to the fact that the global preconditioner is constructed on the filtered matrix in the case of deco, which is incorporated inside the FSAI preconditioner. This filtering, while being a cheap and embarrassingly parallel operation, does induce overhead while reducing the computational complexity of FSAI substantially. In general, apart from the filtering advantage, we see a similar scalability trend as in the Norne case, with slightly better scaling due to the increased size of the matrix, and therefore a better parallelism ($> 2x$ speedup in 16 MPI configuration).

Sleipner

The Sleipner 2019 Benchmark Model is a publicly available reference dataset derived from the Sleipner CO_2 storage project in the North Sea. The model uses corner-point grid geometry to represent a 3D volume spanning

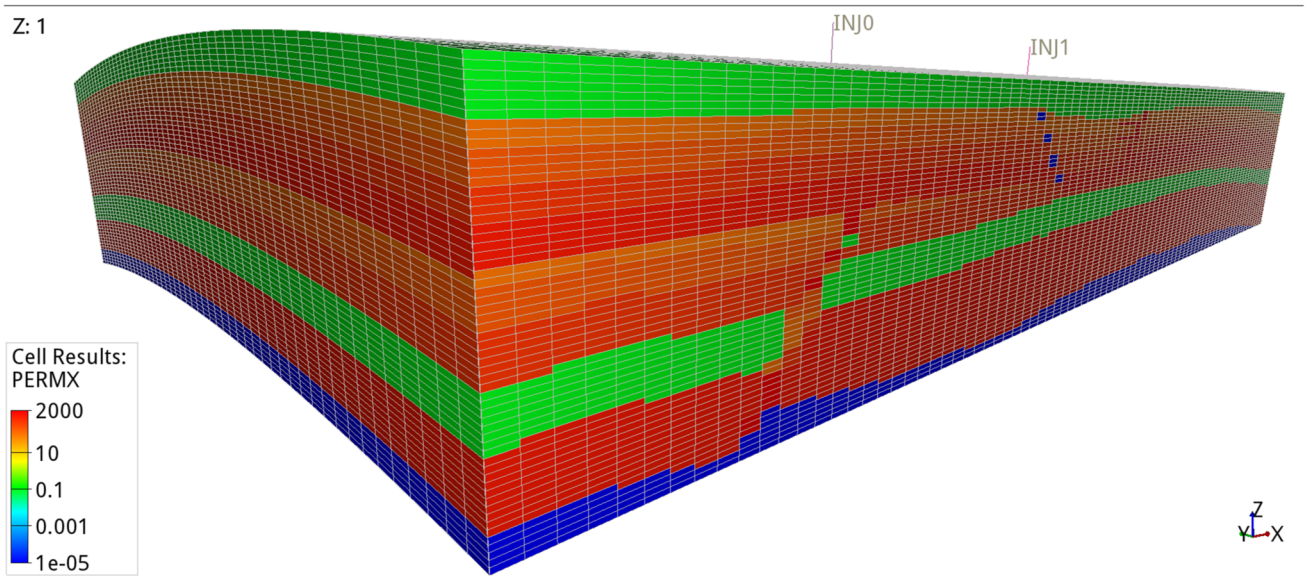

Fig. 4 Mesh grid, horizontal permeability, and well set-up of the SPE11C reservoir model

Table 3 Simulation results for SPE11C model over 1 year of injection using CPR-BiCGstab solver (relative residual tolerance 10^{-3}) in a total of 22 linear systems, *1 MPI rank*

	Total time [s]		Iterations	
	Setup	Solve	Total	Average
CPR:OPM:DUNE	38.4	14.8	89	4.1
CPR:deco	26.9	12.9	58	2.6

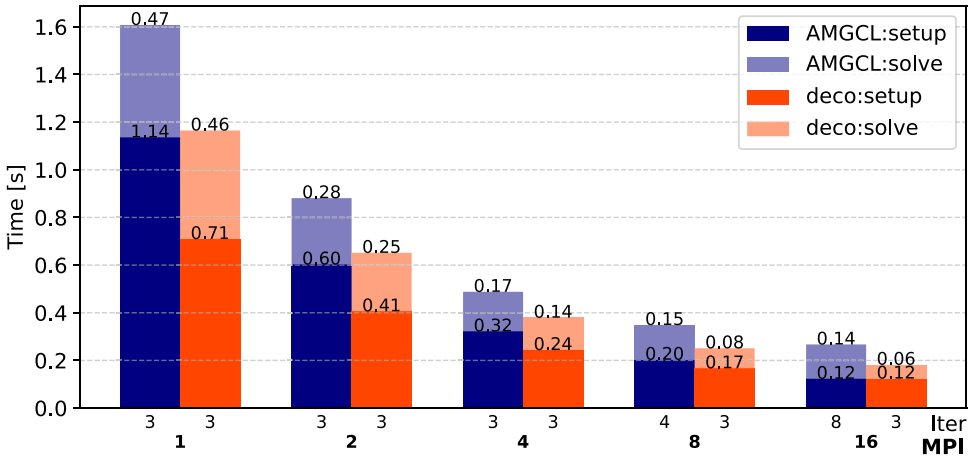

Fig. 5 Scaling of setup and solution time versus the number of MPI ranks for the SPE11C model. Bars show AMGCL (blue) and deco (orange) with CPR right-preconditioned GMRES solver at relative tolerance 10^{-3}. GMRES iteration counts are indicated below each bar, and MPI ranks are shown below

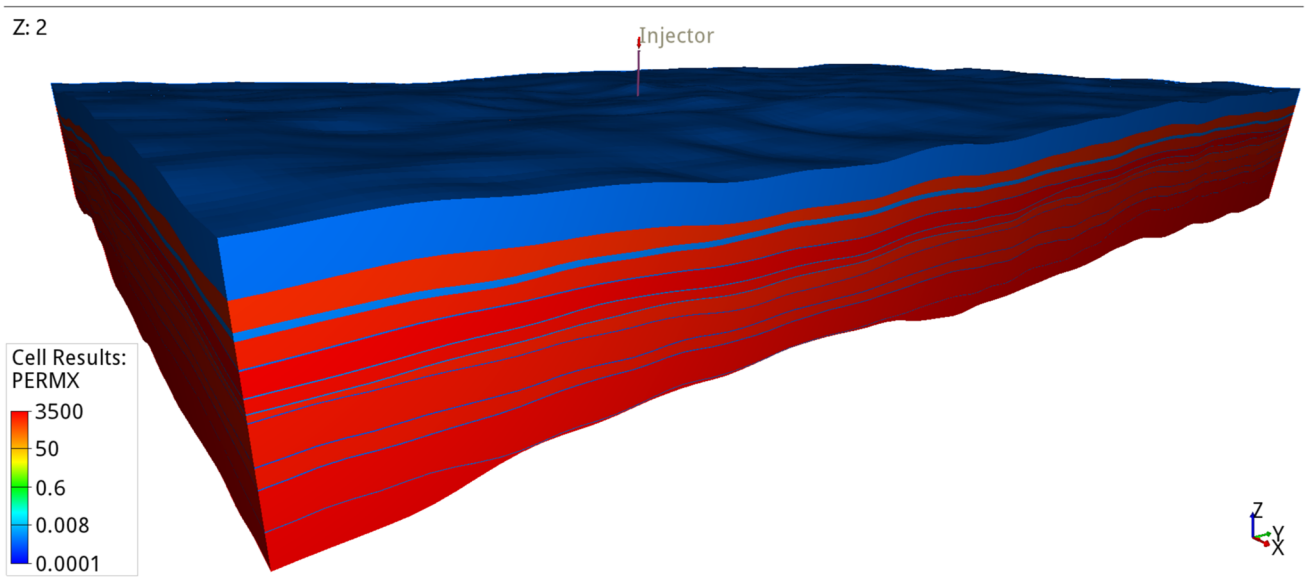

Fig. 6 Mesh grid, horizontal permeability, and well set-up of the Sleipner reservoir model. The z-direction has been exaggerated by a factor 2

Table 4 Simulation results for Sleipner model over 1 month of injection using CPR-BiCGstab solver (relative residual tolerance 10^{-2}) in a total of 32 linear systems, *1 MPI rank*

	Total time [s]		Iterations	
	Setup	Solve	Total	Average
CPR:OPM:DUNE	233.41	59.56	48	1.5
CPR:deco	159.72	64.28	68	2.1

approximately 3.2 × 5.9 km laterally and up to 300 m in depth. The model is configured for two-phase flow: CO_2 and brine [50]. The Sleipner model poses a demanding benchmark for scalability and solver performance. It's approximately a 2-million-cell grid (1, 986, 176 cells to be precise) places substantial demands on the linear solvers used in fully implicit simulations. As such, the benchmark is not only a geologically realistic CO_2 storage case, but also a computationally demanding and numerically challenging test for reservoir simulation solver libraries.

In Table 4, the results of the simulation are presented over a one-month period of injection. In comparison to the previous models, the Sleipner model is much more demanding, allowing it to proceed only with small time steps. Moreover, the layered layout with sharp jumps in the permeability tensor (Fig. 6) produces extreme and isolated eigenvalues of the system matrix, which are harmful for the convergence of the iterative solver, imposing an extra difficulty [53] with respect to other models. Despite this, the solution times are comparable with only 8% slowdown of *CPR:deco*. The reason for the degradation stems from the fact that ILU is much more robust, making it difficult to compete with FSAI; however, the main bottleneck for the FSAI-AMG CPR preconditioner to work efficiently is the effective decoupling process. Since ILU0 is a comparatively stronger smoother with respect to FSAI, assuming a similar nonzero footprint, it is able to smooth a wider range of error modes associated with the corresponding generalized eigenvalue problem; thus, it can potentially compensate for a "weaker" decoupling and pressure correction. If the decoupling has efficiently *(i)* separated pressure and saturation variables with *(ii)* the pressure system being amenable to AMG (consequence of DRS weights Eq. 10 and M-matrix property) and *(iii)* the global system being amenable to FSAI (consequence of CPD decoupling (12) and elimination of strong connections away from the main diagonal), then the FSAI-AMG pair will work out successfully within the CPR preconditioner context. Another advantage of FSAI over ILU preconditioners is that it cannot suffer a breakdown during its setup, which is one of the main drawbacks of incomplete factorization methods for ill-conditioned problems.

In general, however, the performance is satisfactory, taking into account that the major improvement comes from the scalability of the proposed approach.

Finally, in Fig. 7, we see a similar scalability pattern again as in the previous two models. Even though AMGCL at 16 MPI configuration matches the 16 MPI configuration of *deco* in total time, we want to highlight that the offset in the setup time is not comparable with the boost in the solution time, where the speedup of the proposed framework is ≈ 4× times – a consequence of an even larger system size.

SPE11C_refined

In order to demonstrate the actual scalability of our framework, we have generated a larger SPE11C model with 45 million of unknowns and 1 billion of nonzeros and used it perform a strong scalability test with up to 2048 MPI ranks. The experiments have been run on the Leonardo supercomputer (Booster partition) hosted at the CINECA

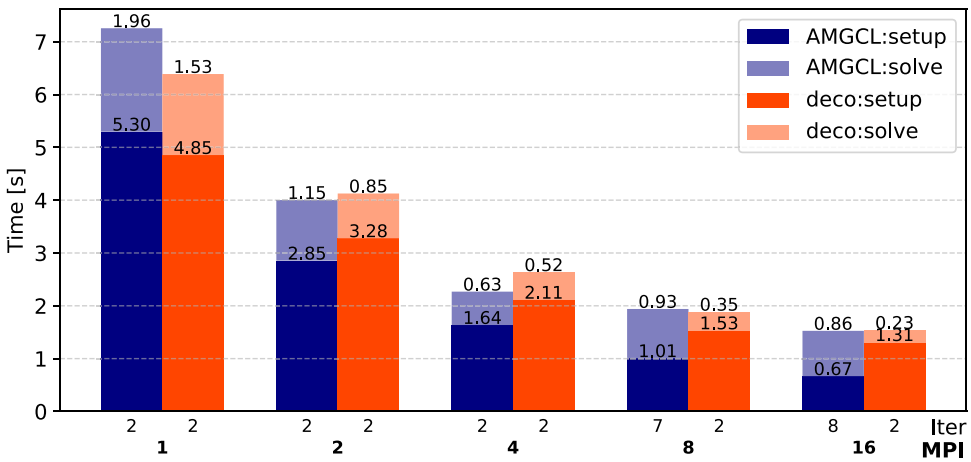

Fig. 7 Scaling of setup and solution time versus the number of MPI ranks for the Sleipner model. Bars show AMGCL (blue) and deco (orange) with CPR right-preconditioned GMRES solver at relative tolerance 10^{-2}. GMRES iteration counts are indicated below each bar, and MPI ranks are shown below

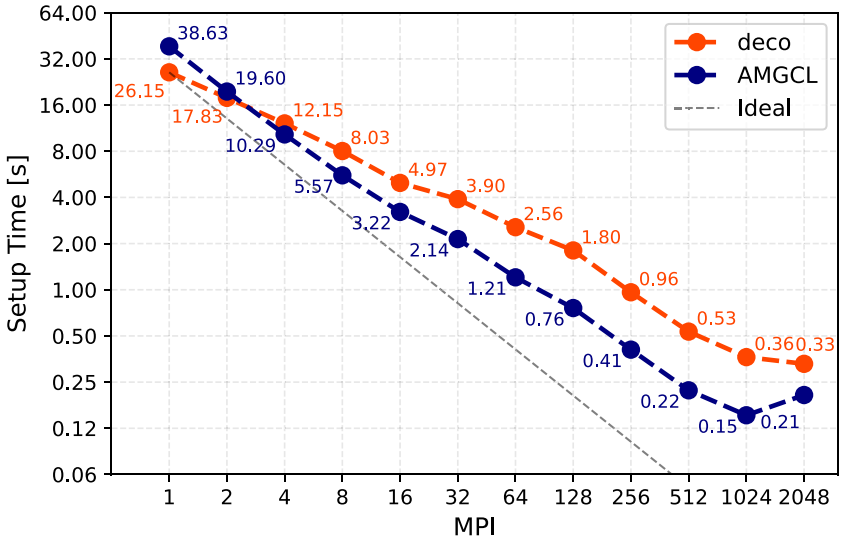

Fig. 8 Setup time versus the number of MPI ranks for the SPE11C refined model. *AMGCL* is shown in blue and deco in orange. The dashed line represents the ideal setup time that is obtained by dividing the serial time by the number of MPI ranks

computer center. Leonardo is a cluster composed of 3456 nodes, each equipped with a 32-cores Intel Ice Lake CPU (Intel Xeon Platinum 8358) at 2.60GHz and Mellanox Infiniband HDR DragonFly+ network at 200 Gb/s.

Figure 8 compares the scalability of the CPR preconditioner setup phase for *deco* and *AMGCL*. Both implementations exhibit good scalability quite close to the ideal scalability represented by the dashed line. *AMGCL* consistently outperforms *deco*, achieving, on average, approximately a twofold reduction in setup time. This observation is consistent with the scalability results obtained for up to 16 MPI ranks shown in Figs. 2, 5, and 7, where *AMGCL* likewise demonstrated a faster setup phase.

Looking at the scalability of the solution time (Fig. 9) we see again that both sovers perform quite well. However, *AMGCL* requires a number of iterations to converge that grows with the number of MPI ranks, a trend already seen in Figs. 2, 5, and 7. In the solution stage *deco* shows a speed-up of about 3× over *AMGCL*.

For the scalability study, we relaxed the stopping tolerance of *AMGCL* to obtain iteration counts at one MPI rank comparable to those of *deco*. Otherwise, *AMGCL* consistently required substantially more iterations, as its MPI implementation currently relies only on the standard quasi-IMPES decoupling strategy.

Finally, to better understand the relative weight of each stage of the *deco* set-up and application, Figs. 10 and 11 provide a detailed breakdown of the time spent for each component of the preconditioner during setup and application for a single-MPI-rank and 2048-MPI-rank runs. The main observation is that the relative weight

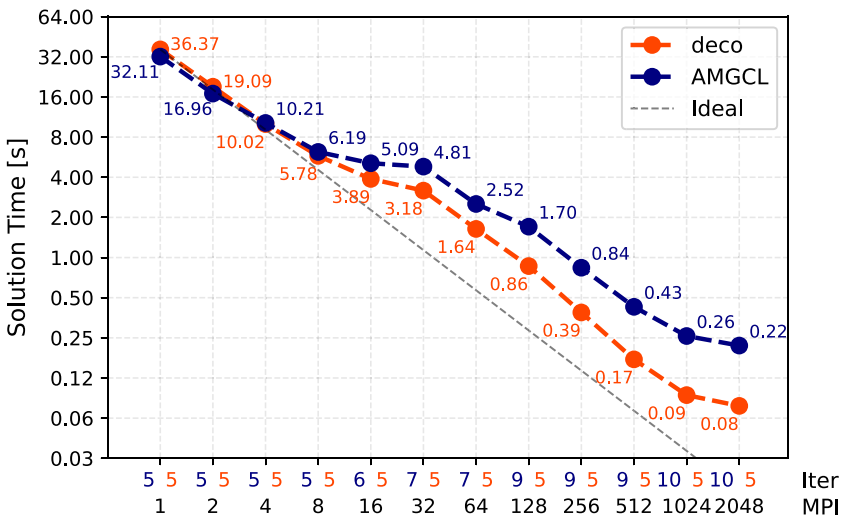

Fig. 9 Solution time of the CPR right-preconditioned GMRES solver versus the number of MPI ranks for the SPE11C refined model. *AMGCL* is shown in blue and deco in orange. The dashed line represents the ideal setup time that is obtained by dividing the serial time by the number of MPI ranks

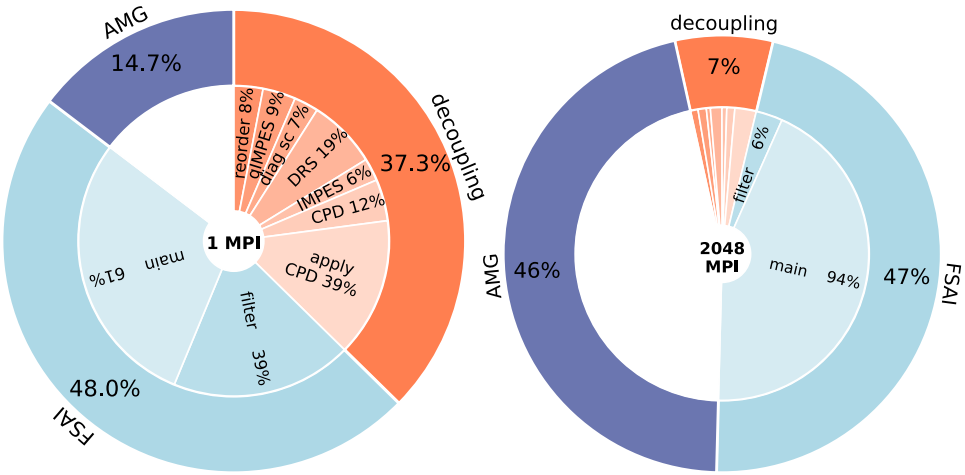

Fig. 10 Breakdown of the computational times of the setup of the CPR preconditioner from *deco* for the SPE11C_refined model using 1 (left) and 2048 (right) MPI ranks. All the stages of the decoupling process and FSAI computation are further detailed

of AMG grows with the number of MPI ranks especially in the set-up phase. This is not surprising and it is a consequence of the multilevel nature of AMG. The use of multiple ranks at the lower levels is significantly less efficient because very small problems are highly fragmented across the cores. By contrast, both the decoupling process and FSAI are applied to the fine level only and for this reason prove more scalable. The decoupling stage, in particular, exhibits excellent scalability because it is embarrassingly parallel and requires no communication between MPI ranks, as each partition contains complete cells. The same situation holds for the filtering process in the FSAI set-up. No communication takes place during filtering and so its relative weight significantly decreases in the run with 2048 ranks.

6 Conclusion

In this work, we have successfully introduced the FSAI preconditioner as a global smoother in the CPR framework. In order to make the new CPR effective, we have augmented the quasi-IMPES decoupling with the CPD Eq. 12 stage. For comparison, we have used two libraries: *DUNE* [3] library for the serial computation with one processor for the purpose of validation of the proposed methodology on an entire simulation; furthermore, *AMGCL* [16] library was used to investigate parallel scalability of the proposed framework. The results indicate a comparable performance in a one-processor setting against *DUNE* and approximately $2-4\times$ times performance boost in

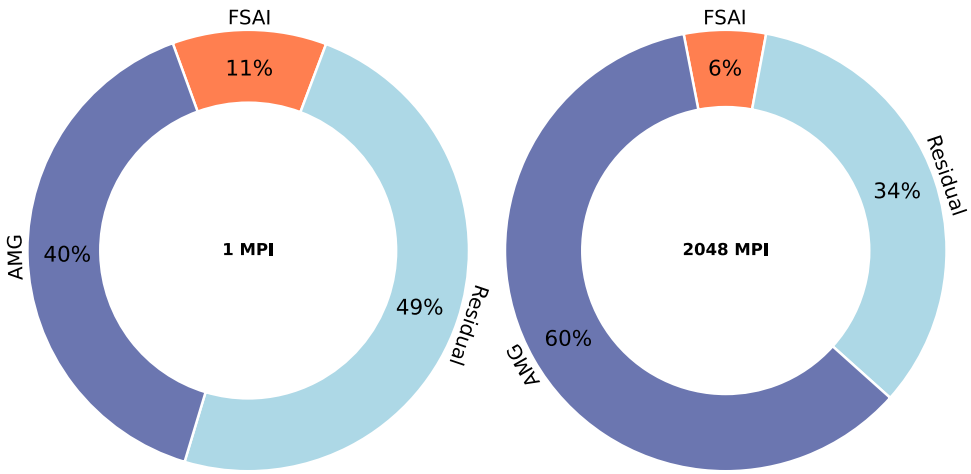

Fig. 11 Breakdown of the computational time of the application of the CPR preconditioner from *deco* for the SPE11C_refined model using 1 (left) and 2048 (right) MPI ranks

parallel for the selected models against *AMGCL* library. A detailed scalability study with up to 2,048 cores is also provided for AMGCL and deco.

For the scalability of our approach, we used a distributed MPI framework. It is true that the majority of the reservoir simulators achieve parallel scalability through a domain decomposition (DD) approach across the reservoir grid, in which each local domain is solved with shared memory parallelism OpenMP/OpenCL/CUDA. Although this enables some degree of parallelism, the linear solver itself remains largely sequential, with only limited acceleration from shared-memory parallelization [45]. To fully exploit the resources of modern computing clusters and scale beyond the DD limit, distributed linear solvers are required. However, parallelizing ILU across distributed frameworks – especially ILU0 – remains a bottleneck. FSAI, instead, proved to be an efficient alternative according to our results.

In future work, the *deco* library will be fully ported to GPU where it is expected to achieve even higher speedups against other ILU-based approaches.

Appendix: Software parameters

Fig. 12 The default setup used in *OPM:CPR:DUNE* solver

```
{
  "maxiter": "50",
  "tol": "0.001",
  "verbosity": "10",
  "solver": "bicgstab",
  "preconditioner": {
      "type": "cpr",
      "weight_type": "quasiimpes",
      "finesmoother": {
          "type": "ParOverILU0",
          "relaxation": "1"
      },
      "verbosity": "0",
      "coarsesolver": {
          "maxiter": "1",
          "tol": "0.1",
          "solver": "loopsolver",
          "verbosity": "0",
          "preconditioner": {
              "type": "amg",
              "alpha": "0.333",
              "relaxation": "1",
              "iterations": "1",
              "coarsenTarget": "1200",
              "pre_smooth": "1",
              "post_smooth": "1",
              "beta": "0",
              "smoother": "ILU0",
              "verbosity": "0",
              "maxlevel": "15",
              "skip_isolated": "0",
              "accumulate": "1",
              "prolongationdamping": "1",
              "maxdistance": "2",
              "maxconnectivity": "15",
              "maxaggsize": "6",
              "minaggsize": "4"
          }
      }
  }
}
```

Fig. 13 The default setup used in *CPR:AMGCL* solver

```
{
    "solver": {
        "tol": "0.001",
        "maxiter": "100",
        "M": "50",
        "type": "gmres",
        "check_after": "false"
    },
    "precond": {
        "block_size": "3",
        "pprecond": {
            "coarsening": {
                "type": "aggregation",
                "over_interp": "1",
                "aggr": {
                    "eps_strong": "0.08"
                }
            },
            "direct_coarse": "true",
            "ncycle": "1",
            "npre": "1",
            "npost": "1",
            "pre_cycles": "1",
            "relax": {
                "type": "ilu0",
                "damping": "1"
            }
        },
        "sprecond": {
            "type": "ilu0",
            "damping": "1"
        },
    }
}
```

Fig. 14 The default setup used in *CPR:deco*

```
"solver": {
    "type": "gmres",
    "reltol": "1e-3",
    "restart": "50",
    "itmax": "50",
    "verbosity": "1"
},
"decoupling": {
  "weights": "quasi-IMPES",
  "DRS": {
    "e_dd": "0.2",
    "e_ps": "0.00"
    },
  "CPD": "1",
  "seq": "2"
},
"prec_loc": {
  "name": "hypre AMG",
  "Cycle": "V-cycle",
  "coarsening": "HMIS",
  "relaxation": "Gauss-Seidel",
  "prolongation": "EXTI+I",
  "A-matrix drop_tol": "1e-4",
  "Prol_trunc_fac": "1e-3",
  "SoC_alpha": "0.5",
  "MaxCoarseSize": "300"
},
"prec_glob": {
  "name": "Chronos FSAI",
  "type": "static",
  "power pattern": "1",
  "pre_filter": "0.8",
  "post_filter": "0.8"
}
```

Author Contributions A.M. developed and implemented the algorithm, performed the experiments, and wrote the first draft of the manuscript; A.F. and C.J. designed the study, supervised the project, analysed the data, and revised the manuscript. All authors read and approved the manuscript.

Funding Open access funding provided by Chalmers University of Technology. This research did not receive funding.

Data Availability All the models (Norne, SPE11C, Sleipner) that have been used for testing are available in the OPM project: https://github.com/OPM

Declarations

Competing Interests The authors declare no competing interests.

Ethics Approval Not applicable.

Consent to Participate Not applicable.

Consent to Publish Not applicable.

References

1. Andersen, T.M., Torben, J., Lye, K.O., Rasmussen, A.F., Lie, K.-A.: Graphics processing unit-accelerated incomplete LU preconditioners for reservoir simulation. SPE J. **30**(12), 7873–7892 (2025). https://doi.org/10.2118/223873-PA

2. Anzt, H., Huckle, T.K., Bräckle, J., Dongarra, J.: Incomplete sparse approximate inverses for parallel preconditioning. Parallel Comput. **71**, 1–22 (2018). https://doi.org/10.1016/j.parco.2017.10.003, https://www.sciencedirect.com/science/article/pii/S016781911730176X
3. Bastian, P., Blatt, M., Dedner, A., Dreier, N.-A., Engwer, C., Fritze, R., Gräser, C., Grüninger, C., Kempf, D., Klöfkorn, R., Ohlberger, M., Sander, O.: The Dune framework: Basic concepts and recent developments. Comput. Math. Appl. **81**, 75–112 (2021). https://doi.org/10.1016/j.camwa.2020.06.007, https://www.sciencedirect.com/science/article/pii/S089812212030256X
4. Benzi, M., Szyld, D.B., van Duin, A.: Orderings for incomplete factorization preconditioning of nonsymmetric problems. SIAM J. Sci. Comput. **20**(5), 1652–1670 (1999). https://doi.org/10.1137/S1064827597326845
5. Bernaschi, M., Bisson, M., Fantozzi, C., Janna, C.: A factored sparse approximate inverse preconditioned conjugate gradient solver on graphics processing units. SIAM J. Sci. Comput. **38**, C53–C72 (2016). https://doi.org/10.1137/15M1027826
6. Bernaschi, M., Carrozzo, M., Franceschini, A., Janna, C.: A dynamic pattern factored sparse approximate inverse preconditioner on graphics processing units. SIAM J. Sci. Comput. **41**(3), C139–C160 (2019). https://doi.org/10.1137/18M1197461
7. Booth, J.D., Bolet, G.: An on-node scalable sparse incomplete LU factorization for a many-core iterative solver with Javelin. Parallel Comput. **94–95**, 102622 (2020). https://doi.org/10.1016/j.parco.2020.102622, https://www.sciencedirect.com/science/article/pii/S0167819120300156
8. Bui, Q.M., Wang, L., Osei-Kuffuor, D.: Algebraic multigrid preconditioners for two-phase flow in porous media with phase transitions. Adv. Water Resour. **114**, 19–28 (2018). https://api.semanticscholar.org/CorpusID:119132671
9. Bui, Q.M., Osei-Kuffuor, D., Castelletto, N., White, J.A.: A scalable multigrid reduction framework for multiphase poromechanics of heterogeneous media. SIAM J. Sci. Comput. **42**(2), B379–B396 (2020). https://doi.org/10.1137/19M1256117
10. Cao, H., Tchelepi, H.A., Wallis, J., Yardumian, H.: Parallel scalable unstructured CPR-type linear solver for reservoir simulation. In: SPE Annual Technical Conference and Exhibition, SPE Annual Technical Conference and Exhibition, pp. SPE–96809–MS, (2005). https://doi.org/10.2118/96809-MS
11. Chen, Z., Huan, G., Ma, Y.: Computational Methods for Multiphase Flows in Porous Media, vol. 2. SIAM, Philadelphia (2006)
12. Chow, E., Saad, Y.: Approximate inverse preconditioners via sparse-sparse iterations. SIAM J. Sci. Comput. **19**(3), 995–1023 (1998). https://doi.org/10.1137/S1064827594270415
13. Coats, K.H.: A note on IMPES and some IMPES-based simulation models. SPE J. **5**(03), 245–251 (2000). https://doi.org/10.2118/65092-PA
14. Cremon, M.A., Castelletto, N., White, J.A.: Multi-stage preconditioners for thermal-compositional-reactive flow in porous media. J. Comput. Phys. **418**, 109607 (2020). https://doi.org/10.1016/j.jcp.2020.109607. https://www.sciencedirect.com/science/article/pii/S0021999120303818
15. Dawson, C., Klie, H., Wheeler, M., Woodward, C.: A parallel, implicit, cell-centered method for two-phase flow with a preconditioned Newton-Krylov solver. Comput. Geosci. **1**, 215–249 (1997). https://doi.org/10.1023/A:1011521413158
16. Demidov, D., Mu, L., Wang, B.: Accelerating linear solvers for Stokes problems with C++ metaprogramming. J. Comput. Sci. **49**, 101285 (2021). https://doi.org/10.1016/j.jocs.2020.101285, http://www.sciencedirect.com/science/article/pii/S1877750320305809
17. Esler, K., Mukundakrishnan, K., Natoli, V., Shumway, J., Zhang, Y., Gilman, J.: Realizing the potential of GPUs for reservoir simulation. **2014**(1), 1–16, (2014). https://doi.org/10.3997/2214-4609.20141771.
18. Esler, K., Gandham, R., Patacchini, L., Garipov, T., Samardzic, A., Panfili, P., Caresani, F., Pizzolato, A., Cominelli, A.: A graphics processing unit-based, industrial grade compositional reservoir simulator. SPE J. **27**(01), 597–612 (2022). https://doi.org/10.2118/203929-PA
19. Falgout, R., Yang, U.: Hypre: A library of high performance preconditioners. **2331**, 632–641 (2002). https://doi.org/10.1007/3-540-47789-6sps66
20. Ferronato, M.: Preconditioning for sparse linear systems at the dawn of the 21st century: history, current developments, and future perspectives. ISRN Appl. Math. **2012**, 1–49 (2012). https://doi.org/10.5402/2012/127647
21. Ferronato, M., Janna, C., Pini, G.: A generalized Block FSAI preconditioner for nonsymmetric linear systems. J. Comput. Appl. Math. **256**, 230–241 (2014). https://doi.org/10.1016/j.cam.2013.07.049, https://www.sciencedirect.com/science/article/pii/S0377042713003944
22. Gandham, R., Zhang, Y., Esler, K., Natoli, V.: Improving GPU throughput of reservoir simulations using NVIDIA MPS and MIG. **2021**(1), 1–5. (2021). https://doi.org/10.3997/2214-4609.2021612025.
23. 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
24. Grote, M.J., Huckle, T.: Parallel preconditioning with sparse approximate inverses. SIAM J. Sci. Comput. **18**(3), 838–853 (1997). https://doi.org/10.1137/S1064827594276552
25. Henson, V., Yang, U.: BoomerAMG: A parallel algebraic multigrid solver and preconditioner. Appl. Numer. Math. **41**, 155–177 (2002). https://doi.org/10.1016/S0168-9274(01)00115-5

26. Hu, X., Wu, S., Wu, X.-H., Xu, J., Zhang, C.-S., Zhang, S., Zikatanov, L.: Combined preconditioning with applications in reservoir simulation. Multiscale Model. Simul. **11**(2), 507–521 (2013). https://doi.org/10.1137/120885188
27. Isotton, G., Frigo, M., Spiezia, N., Janna, C.: Chronos: A general purpose classical AMG solver for high performance computing. SIAM J. Sci. Comput. **43**, C335–C357 (2021). https://doi.org/10.1137/21M1398586
28. Isotton, G., Janna, C., Bernaschi, M.: A GPU-accelerated adaptive FSAI preconditioner for massively parallel simulations. Int. J. High Performance Comput. Appl. **36**(2), 153–166 (2022). https://doi.org/10.1177/10943420211017188
29. Janna, C., Ferronato, M., Sartoretto, F., Gambolati, G.: FSAIPACK: A software package for high-performance factored sparse approximate inverse preconditioning. ACM Trans. Math. Softw. **41**(2), (2015). https://doi.org/10.1145/2629475
30. Janna, C., Frigo, M., Isotton, G., Spiezia, N., Colombo, D.: Chronos: A GPU-accelerated and energy efficient linear solver for large scale simulations in geoscience. pp. 1–4, (2023). https://doi.org/10.3997/2214-4609.2023630006
31. Klie, H., Rame, M., Wheeler, M.: Two-stage preconditions for inexact newton methods in multi-phase reservoir simulation, vol. 02, (1997)
32. Krogstad, S., Lie, K.-A., Møyner, O., Nilsen, H.M., Raynaud, X., Skaflestad, B.: MRST-AD - an Open-Source Framework for Rapid Prototyping and Evaluation of Reservoir Simulation Problems. In: SPE Reservoir Simulation Conference, p. D022S002R004. (2015) . SPE-173317-MS. D022S002R004
33. Lacroix, S., Vassilevski, Y., Wheeler, M.: Decoupling preconditioners in the implicit parallel accurate reservoir simulator (IPARS). Numer. Linear Algebra Appl. **8**, 537–549 (2001). https://doi.org/10.1002/nla.264
34. Lacroix, S., Vassilevski, Y., Wheeler, J., Wheeler, M.: Iterative solution methods for modeling multiphase flow in porous media fully implicitly. SIAM J. Sci. Comput. **25**(3), 905–926 (2003). https://doi.org/10.1137/S106482750240443X
35. Landa-Marbán, D., Sandve, T.H.: Pyopmspe11: A Python framework using OPM Flow for the SPE11 benchmark project. J Open Source Softw **10**(105), 7357 (2025). https://doi.org/10.21105/joss.07357
36. Lye, K.O., Andersen, T.M., Rasmussen, A.F., Torben, J.: Gpu-ISTL - extending OPM flow with GPU linear solvers. J. Open Source Softw. **10**(109), 7740 (2025). https://doi.org/10.21105/joss.07740
37. Magras, J.-F., Quandalle, P., Bia, P.: High-performance reservoir simulation with parallel ATHOS. volume SPE Reservoir Simulation Symposium of SPE Reservoir Simulation Conference, pp. SPE–66342–MS, (2001). https://doi.org/10.2118/66342-MS
38. Nardean, S., Ferronato, M., Abushaikha, A.: A novel block non-symmetric preconditioner for mixed-hybrid finite-element-based Darcy flow simulations. J. Comput. Phys. **442**, 110513 (2021). https://doi.org/10.1016/j.jcp.2021.110513
39. Nardean, S., Ferronato, M., Abushaikha, A.: Linear solvers for reservoir simulation problems: An overview and recent developments. *Archives of Computational Methods in Engineering*, **29**, (2022). https://doi.org/10.1007/s11831-022-09739-2
40. Nardean, S., Ferronato, M., Abushaikha, A.: Block constrained pressure residual preconditioning for two-phase flow in porous media by mixed hybrid finite elements. Comput. Geosci. **28**, 1–20 (2023). https://doi.org/10.1007/s10596-023-10238-x
41. Nilsen, H.M., Ahmed, E., Rasmussen, A.F., Bao, K., Skille, T.: Constrained pressure residual preconditioner including wells for reservoir simulation. In: SPE Reservoir Simulation Conference, SPE Reservoir Simulation Conference, p. D021S009R001 (2023). https://doi.org/10.2118/212172-MS
42. Nordbotten, J.M., Ferno, M.A., Flemisch, B., Kovscek, A.R., Lie, K.-A.: The 11th society of petroleum engineers comparative solution project: Problem definition. SPE J. **29**(05), 2507–2524 (2024). https://doi.org/10.2118/218015-PA
43. Panfili, P., Patacchini, L., Ferrari, A.: Implementation of Soreide and Whitson EoS in a GPU-based reservoir simulator. Comput. Geosci. **28**, 341–354 (2024). https://doi.org/10.1007/s10596-023-10257-8
44. Qiao, C., Wu, S., Xu, J., Zhang, C.-S.: Analytical decoupling techniques for fully implicit reservoir simulation. J. Comput. Phys. **336**, 664–681 (2017). https://doi.org/10.1016/j.jcp.2017.02.037, https://www.sciencedirect.com/science/article/pii/S0021999117301316
45. Rasmussen, A., Sandve, T., Bao, K., Lauser, A., Hove, J., Skaflestad, B., Klöfkorn, R., Blatt, M., Rustad, A., Sævareid, O., Lie, K., 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
46. Roy, T., Jönsthövel, T.B., Lemon, C., Wathen, A.J.: A block preconditioner for non-isothermal flow in porous media. J. Comput. Phys. **395**, 636–652 (2019). https://doi.org/10.1016/j.jcp.2019.06.038, https://www.sciencedirect.com/science/article/pii/S0021999119304462
47. Roy, T., Jönsthövel, T.B., Lemon, C., Wathen, A.J.: A constrained pressure-temperature residual (CPTR) method for non-isothermal multiphase flow in porous media. SIAM J. Sci. Comput. **42**(4), B1014–B1040 (2020). https://doi.org/10.1137/19M1292023
48. Rwechungura, R., Suwartadi, E., Dadashpour, M., Kleppe, J., Foss, B.: The norne field case - a unique comparative case study. (2010). https://doi.org/10.2523/127538-MS
49. Saad, Y.: Iterative methods for sparse linear systems. SIAM, (2003). https://doi.org/10.1137/1.9780898718003.ch4
50. Sandve, T.H., Gasda, S.E., Rasmussen, A., Rustad, A.B.: Convective dissolution in field scale CO2 storage simulations using the OPM flow simulator. In: Proceedings of the trondheim conference on CO2 capture, transport and storage (TCCS-11), Trondheim, Norway. Norce, Bergen, Norway; SINTEF, Oslo, Norway; Equinor, Trondheim, Norway. Author affiliations: 1 Norce, Bergen, Norway; 2 SINTEF, Oslo, Norway; 3 Equinor, Trondheim, Norway (2021)

51. Scheichl, R., Masson, R., Wendebourg, J.: Decoupling and block preconditioning for sedimentary basin simulations. Comput. Geosci. **7**(4), 295–318 (2003). https://doi.org/10.1023/B:COMG.0000005244.61636.4e
52. Stueben, K., Clees, T., Klie, H., Lu, B., Wheeler, M.: Algebraic multigrid methods (AMG) for the efficient solution of fully implicit formulations in reservoir simulation. SPE Reservoir Simulation Symposium Proceedings, (2007). https://doi.org/10.2118/105832-MS
53. van der Linden, J., Jönsthövel, T., Lukyanov, A., Vuik, C.: The parallel subdomain-levelset deflation method in reservoir simulation. J. Comput. Phys. **304**, 340–358 (2016). https://doi.org/10.1016/j.jcp.2015.10.016, https://www.sciencedirect.com/science/article/pii/S0021999115006804
54. Wallis, J.R., Kendall, R.P., Little, T.E.: Constrained residual acceleration of conjugate residual methods. In: SPE Reservoir Simulation Symposium, SPE Reservoir Simulation Conference, pp. SPE–13536–MS, (1985). https://doi.org/10.2118/13536-MS
55. Wang, K., Liu, H., Luo, J., Chen, Z.: Efficient CPR-type preconditioner and its adaptive strategies for large-scale parallel reservoir simulations. J. Comput. Appl. Math. **328**, 443–468 (2018). https://doi.org/10.1016/j.cam.2017.07.022, https://www.sciencedirect.com/science/article/pii/S0377042717303734
56. White, J.A., Castelletto, N., Klevtsov, S., Bui, Q.M., Osei-Kuffuor, D., Tchelepi, H.A.: A two-stage preconditioner for multiphase poromechanics in reservoir simulation. Comput. Methods Appl. Mech. Eng. **357**, 112575 (2019). https://doi.org/10.1016/j.cma.2019.112575, https://www.sciencedirect.com/science/article/pii/S0045782519304402
57. Yeremin, A.Y., Nikishin, A.A.: Factorized-sparse-approximate-inverse preconditionings of linear systems with unsymmetric matrices. J. Math. Sci. **101**, 2448–2457 (2004). https://doi.org/10.1023/B:JOTH.0000026282.08256.45

Authors and Affiliations

Artem Mavliutov[1,2] · Andrea Franceschini[2] · Carlo Janna[2,3]

✉ Artem Mavliutov
artemma@chalmers.se

Andrea Franceschini
andrea.franceschini@unipd.it

Carlo Janna
carlo.janna@unipd.it

[1] Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, Gothenburg, Sweden

[2] Department of Civil, Environmental and Architectural Engineering, University of Padova, Padova, Italy

[3] M3E S.r.l., Padova, Italy