

Parallel Multi-Stage Preconditioners with Adaptive Setup for the Black Oil Model*

Li Zhao[†], Chunsheng Feng[‡], Chensong Zhang[§], and Shi Shu[†]

Abstract. The black oil model is widely used to describe multiphase porous media flow in the petroleum industry. The fully implicit method features strong stability and weak constraints on timestep sizes; hence, it is commonly used in current mainstream commercial reservoir simulators. In this paper, a Constrained Pressure Residual (CPR) preconditioner with an adaptive “setup phase” is developed to improve the parallel efficiency of a petroleum reservoir simulation. Furthermore, we propose a multi-color Gauss–Seidel (GS) algorithm for the algebraic multigrid method based on the coefficient matrix of strong connections. Numerical experiments show that the proposed preconditioner can improve the parallel performance for both OpenMP and Compute Unified Device Architecture (CUDA) implements. Moreover, the proposed algorithm yields good parallel speedup as well as the same convergence behavior as the corresponding single-thread algorithm. In particular, for a three-phase benchmark problem (about 3.28 million degrees of freedom), the parallel speedup of the OpenMP version is over 6.5 with 16 threads, and the CUDA version reaches more than 9.5.

Key words. Black oil model; fully implicit method; parallel computing; multi-color Gauss–Seidel smoother; multi-stage preconditioners.

AMS subject classifications. 49M20, 65F10, 68W10, 76S05

1. Introduction. Research on petroleum reservoir simulation can be traced back to the 1950s. To describe and predict the transportation of hydrocarbons, various mathematical models have been established, such as the black oil model, compositional model, thermal recovery model, and chemical flooding model [1–5]. The black oil model consists of multiple coupled nonlinear partial differential equations (PDEs). It is a fundamental mathematical model to describe the three-phase flow in petroleum reservoirs and is widely used in simulating primary and secondary recovery.

After 70 years of development, there is a large body of research on the numerical methods of the black oil model, including the Simultaneous Solution (SS) method [6], Fully Implicit Method (FIM) [7], IMPLICIT Pressure Explicit Saturation (IMPES) method [8], and Adaptive Implicit Method (AIM) [9]. Compared with other methods, FIM is commonly used in mainstream commercial reservoir simulators because of its unconditional stability with respect to timestep sizes. However, a coupled Jacobian linear algebraic system needs to be solved in each Newton iteration step. Owing to the complexity of the practical engineering problem, such systems are difficult to solve with traditional linear solvers. In the reservoir simulation, the solution time of Jacobian systems easily occupies more than 80% of the whole simulation time. Therefore, how to efficiently solve coupled Jacobian systems, especially on modern computers, is a problem that still attracts a lot attention today.

Typically, linear solution methods can be divided into two phases, the “setup phase” (SETUP) and the “solve phase” (SOLVE). These methods can usually be categorized as direct methods [10] and iterative methods [11]. Compared with direct methods, iterative methods have the advantages of low memory and computation complexity and potentially good parallel scalability [12]. The linear algebraic systems arising from fully implicit petroleum reservoir

*This is a preprint manuscript, which is submitted to Computers and Geosciences Journal.

[†]School of Mathematics and Computational Science, Xiangtan University, Xiangtan, Hunan 411105, P. R. China

[‡]National Center for Applied Mathematics in Hunan, Xiangtan University, Xiangtan 411105, P. R. China; Hunan Shaofeng Institute for Applied Mathematics, Xiangtan 411105, P. R. China.

[§]Academy of Mathematics and System Sciences, Beijing 100190, P. R. China. (Corresponding author: Chensong Zhang, Email: zhangcs@lsec.cc.ac.cn).

simulation are usually solved by iterative methods. In particular, Krylov subspace methods [11] (e.g., GMRES and BiCGstab) are frequently adopted. For ill-conditioned linear systems, the preconditioning technique [13] is needed to accelerate the convergence of the iterative methods. The preconditioners for reservoir simulation include: Incomplete LU (ILU) factorization [14], Algebraic MultiGrid (AMG) [15,16], Constrained Pressure Residual (CPR) [17–20], and Multi-Stage Preconditioner (MSP) [21–23]. The ILU method is relatively easy to implement, but as the problem size increases, its convergence deteriorates. The advantages of the AMG method are that it is easy to use and effective on elliptic problems. Owing to the asymmetry, heterogeneity, and nonlinear coupling feature of petroleum reservoir problems, the performance of the AMG method also deteriorates. The CPR method combines the advantages of ILU and AMG, and the MSP method is a generalization of CPR.

As multi-core and many-core architectures have become more popular, parallel computing for petroleum reservoir simulation is now a subject of great interest. In recent years, there has been some work on parallel algorithms for reservoir problems [24–34], and the references therein. For example, [26] designed an OpenMP parallel algorithm with high efficiency and a low memory cost for standard interpolation and coarse grid operator of AMG, under the framework of Fast Auxiliary Space Preconditioning (FASP, <http://www.multigrid.org/fasp/>). [31,32] developed a Method of Subspace Correction (MSC) based on [26] and realized a cost-effective OpenMP parallel reservoir numerical simulation. [28] designed a GPU parallel algorithm based on the METIS partition for the IMPES method. [33] studied the GPU parallel algorithm of ILU and AMG based on a hybrid sparse storage format.

In this paper, we focus on the solution method for the linear algebraic systems arising from the fully implicit discretization of the black oil model, aiming to improve the parallel efficiency of the CPR preconditioner. The main contributions of this work are listed as follows:

- We propose an adaptive SETUP CPR preconditioner (denoted as ASCPR) to improve the efficiency and parallel performance of the solver. A practical adaptive criterion is proposed to judge whether a new SETUP is necessary. The technology can bring two benefits: (1) The efficiency of the solver is improved because the number of SETUP calls can be significantly reduced; (2) the parallel performance is improved because there are many essentially sequential algorithms in the SETUP (i.e., the parallel speedup of these algorithms is low).
- We propose an efficient parallel algorithm for the Gauss-Seidel (GS) relaxation in AMG methods. Starting from the strong connections of coefficient matrix, we design an algorithm for algebraic multi-color grouping. The algorithm has two desirable features: (1) Not relying on the grid (completely transformed into algebraic behavior); (2) yielding the same convergence behavior as the corresponding single-thread algorithm. Furthermore, we use an adjustable strength threshold to filter small matrix entries (enhancing the sparseness) to improve the parallel performance of the algorithm.

The rest of the paper is organized as follows. Section 2 introduces the black oil model and its fully implicit discrete systems. Section 3 reviews the CPR-type preconditioners. In Section 4, an adaptive SETUP CPR preconditioner is proposed. In Section 5, the parallel implementation of multi-color GS based on the coefficient matrix of strong connections is given. In Section 6, numerical experiments are given. Section 7 provides the summary of the work of this paper.

2. Preliminaries.

2.1. The black oil model. This paper considers the following three-phase standard black oil model of water, oil, or gas in porous media [1, 2, 4]. The mass conservation equations of water, oil, and gas, respectively, are

$$(2.1) \quad \frac{\partial}{\partial t} \left(\phi \frac{S_w}{B_w} \right) = -\nabla \cdot \left(\frac{1}{B_w} \mathbf{u}_w \right) + \frac{Q_W}{B_w},$$

$$(2.2) \quad \frac{\partial}{\partial t} \left(\phi \frac{S_o}{B_o} \right) = -\nabla \cdot \left(\frac{1}{B_o} \mathbf{u}_o \right) + \frac{Q_O}{B_o},$$

$$(2.3) \quad \frac{\partial}{\partial t} \left[\phi \left(\frac{S_g}{B_g} + \frac{R_{so} S_o}{B_o} \right) \right] = -\nabla \cdot \left(\frac{1}{B_g} \mathbf{u}_g + \frac{R_{so}}{B_o} \mathbf{u}_o \right) + \frac{Q_G}{B_g} + \frac{R_{so} Q_O}{B_o}.$$

Here, S_α is the saturation of phase α ($\alpha = w, o, g$ represents the water phase, oil phase, and gas phase, respectively), B_α is the volume coefficient of phase α , \mathbf{u}_α is the velocity of phase α , ϕ is the porosity of the rock, R_{so} is the dissolved gas-oil ratio, and Q_β is the injection and production rate of component β ($\beta = W, O, G$ represents the water component, oil component, and gas component, respectively) under the ground standard status.

Assume that the three-phase fluid flow in porous media satisfies Darcy's law:

$$(2.4) \quad \mathbf{u}_\alpha = -\frac{\kappa \kappa_{r\alpha}}{\mu_\alpha} (\nabla P_\alpha - \rho_\alpha \mathbf{g} \nabla z), \quad \alpha = w, o, g,$$

where κ is the absolute permeability, $\kappa_{r\alpha}$ is the relative permeability of phase α , μ_α is the viscosity coefficient of phase α , P_α is the pressure of phase α , ρ_α is the density of phase α , \mathbf{g} is the gravity acceleration, and z is the depth.

The unknown quantities S_α and P_α in Eqs. (2.1)–(2.4) also satisfy the following constitutive relation:

- Saturation constraint equation:

$$(2.5) \quad S_w + S_o + S_g = 1.$$

- Capillary pressure equations:

$$(2.6) \quad \begin{aligned} P_w &= P_o - P_{cow}, \\ P_g &= P_o - P_{cgo}, \end{aligned}$$

where P_{cow} is the capillary pressure between the oil and water phases, and P_{cgo} is the capillary pressure between the gas and oil phases.

2.2. Discretization and algorithm flowchart. The FIM scheme is currently commonly used in mainstream commercial reservoir simulators. This is because the scheme has the characteristics of strong stability and weak constraint on the timestep sizes. These characteristics highlight the advantages of the FIM, especially when the nonlinearity of the models is relatively strong.

In this paper, we use FIM to discretize the governing Eqs. (2.1)–(2.3). That is, the time direction is discretized by the backward Euler method, and the spatial direction is discretized by the upstream weighted central finite difference method [2, 7]. After discretization, the coupled nonlinear algebraic equations are obtained. Such equations are linearized by adopting the Newton method to form the Jacobian system $Ax = b$ of the reservoir equation with implicit wells, namely:

$$(2.7) \quad \begin{pmatrix} A_{RR} & A_{RW} \\ A_{WR} & A_{WW} \end{pmatrix} \begin{pmatrix} x_R \\ x_W \end{pmatrix} = \begin{pmatrix} b_R \\ b_W \end{pmatrix},$$

where A_{RR} and A_{RW} are the derivatives of the reservoir equations for reservoir variables and well variables, respectively; A_{WR} and A_{WW} are the derivatives of the well equations for reservoir variables and well variables, respectively; x_R and x_W are reservoir and bottom-hole flowing pressure variables, respectively; and b_R and b_W are the right-hand side vectors that correspond to the reservoir fields and the implicit wells, respectively.

The subsystem corresponding to the reservoir equations in the discrete system (2.7) is $A_{RR}x_R = b_R$; that is,

$$(2.8) \quad \begin{pmatrix} A_{PP} & A_{PS_w} & A_{PS_o} \\ A_{S_w P} & A_{S_w S_w} & A_{S_w S_o} \\ A_{S_o P} & A_{S_o S_w} & A_{S_o S_o} \end{pmatrix} \begin{pmatrix} x_P \\ x_{S_w} \\ x_{S_o} \end{pmatrix} = \begin{pmatrix} b_P \\ b_{S_w} \\ b_{S_o} \end{pmatrix},$$

where P, S_w , and S_o are primary variables corresponding to oil pressure, water saturation, and oil saturation, respectively.

Remark 1. For convenience, we do not describe how to deal with well equations.

In the following, we present a general algorithm flowchart of the petroleum reservoir simulation; see Fig. 1.

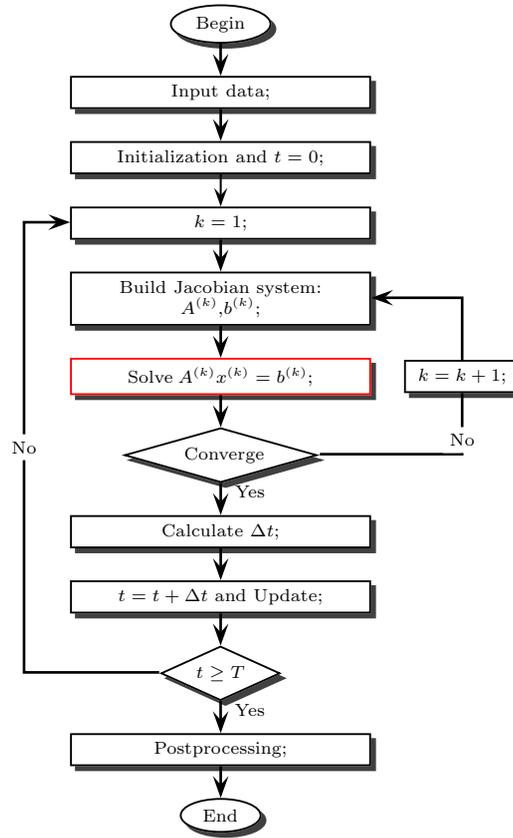


Fig. 1. Algorithm flowchart of the petroleum reservoir simulation.

According to Fig. 1, the algorithm flowchart includes two loops: the outer loop (time marching) and the inner loop (Newton iterations). In each Newton iteration, a Jacobian system $A^{(k)}x^{(k)} = b^{(k)}$ (superscript k is the number of Newton iterations) needs to be solved, which is the main computational work to be carried out.

3. The CPR-type preconditioners. The primary variables usually consists of oil pressure P and saturations S (including S_w and S_o) in FIM, which have different mathematical properties,

respectively. For example, the pressure equation is parabolic, and the saturation equation is hyperbolic [35]. These properties provide a theoretical basis for the design of multiplicative subspace correction methods [13].

3.1. CPR preconditioner. First, the transfer operator $\Pi_P : \mathcal{V}_P \rightarrow \mathcal{V}$ is defined, where \mathcal{V}_P and \mathcal{V} are the pressure variables space and the variables space of the whole reservoir, respectively. Next, a well-known two-stage preconditioner, the Constrained Pressure Residual (CPR) [17–20] preconditioner B , is defined as

$$(3.1) \quad I - BA = (I - RA)(I - \Pi_P B_P \Pi_P^T A),$$

where B_P is solved by the AMG method, and the relaxation (or smoothing) operator R uses the Block ILU (BILU) method [14].

Finally, the CPR preconditioning algorithm is shown in Algorithm 3.1.

Algorithm 3.1 CPR method

Input: A, b, x ;

Output: x ;

- 1 $r \leftarrow b - Ax$;
 - 2 $x \leftarrow x + \Pi_P B_P \Pi_P^T r$;
 - 3 $r \leftarrow b - Ax$;
 - 4 $x \leftarrow x + Rr$.
 - 5 **return** x .
-

3.2. Red-Black GS method. As know, compared with the Jacobi algorithm, the GS algorithm uses the updated values in the iterative process. Hence, the GS algorithm obtains a better convergence rate and is widely used as a smoother of AMG. Now, the parallel red-black GS (also referred to multi-color GS) algorithm on the structured grid is fairly mature [11, 36]. We take a 2D structured grid as an example to present two-color and four-color vertex-grouping diagrams; see Fig. 2.

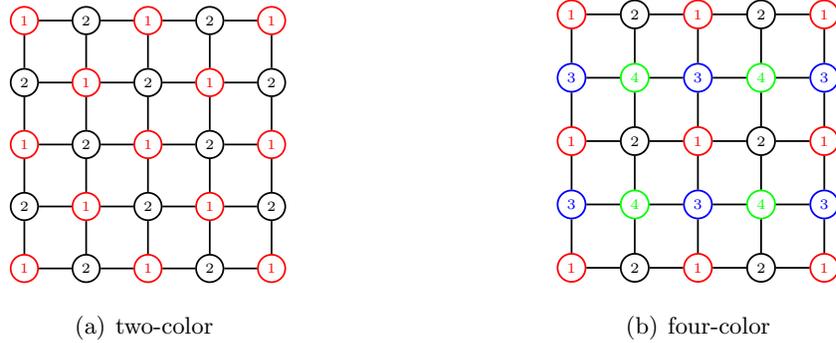


Fig. 2. Two-color and four-color vertex-grouping diagrams.

In Fig. 2(a), the vertices are divided into two groups and marked as red and black points; that is, vertex set V is divided into V_1 and V_2 . In Fig. 2(b), the vertices are divided into four groups, marked as red, black, blue, and green; that is, the vertex set V is divided into V_1 , V_2 , V_3 , and V_4 . The multi-color GS algorithm aims to perform parallel smoothing on the vertices of the same color; that is, (1) for the case of two colors, first, all-red vertices (V_1) are smoothed in parallel, and then all-black (V_2) vertices are smoothed in parallel; (2) for the case of four colors, first, all-red vertices (V_1) are smoothed in parallel; second, all-black vertices (V_2) are

smoothed in parallel; third, all-blue vertices (V_3) are smoothed in parallel; and finally, all-green vertices (V_4) are smoothed in parallel.

Note that different vertices sets are sequential, and the interior of the vertices set is entirely parallel. From the perspective of parallel effects, the above two smooth orderings can yield the same number of iterations as the sequential algorithm. From the scope of application, the two-color grouping is only applicable to the five-point stencil and the four-color grouping can be applied to the nine-point stencil. Similarly, the multi-color GS algorithm of the 2D structured grid can be extended to the 3D structured grid.

The popular parallel variant of GS is the red-black GS algorithm based on structured grids, and it is not suitable for unstructured grids. As a consequence, the application range of the algorithm is limited. Moreover, there is also a hybrid method (combining Jacobi and GS), but its convergence rate deteriorates. From the perspective of parallel implementation, the GS algorithm, an essentially sequential algorithm, is not conducive to yielding the same convergence behavior as the corresponding single-thread algorithm and obtaining high parallel efficiency at the same time.

Finally, we discuss some shortcomings of the standard CPR method.

- (i) Petroleum reservoir simulation is a time-dependent and nonlinear problem. The Jacobian systems need to be solved in each Newton iteration step. The matrix structure of these systems is similar. The CPR method does not take full advantage of the similarity.
- (ii) The CPR method contains many essentially sequential steps in the SETUP, which result in low parallel efficiency.
- (iii) The GS method is commonly used as the smoother in AMG methods. When we try to improve the parallel performance of the smoother, the convergence rate of AMG methods usually deteriorates.

In view of the shortcomings (i) and (ii) mentioned above, we discuss how to reuse similar matrix structures to improve the performance of CPR in Section 4. Furthermore, for the shortcoming (iii), we propose a multi-color GS method from the algebraic point of view in Section 5.

4. An adaptive SETUP CPR preconditioner. In this section, we propose an efficient CPR preconditioner using an adaptive SETUP strategy. For the sake of simplicity, we employ CPR as the preconditioner and restart GMRES as the iterative method (denoted as CPR-GMRES) to illustrate the fundamental idea of the ASCPR preconditioner; the corresponding algorithm is denoted as ASCPR-GMRES. We develop ASCPR-GMRES to efficiently solve the Jacobian systems and take the right preconditioner as an example to describe its implementation; see Algorithm 4.1 and Algorithm 4.2.

Note that the CPR preconditioner $B^{(k)}$ is generated by exploiting an adaptive strategy in Algorithm 4.1. The concrete implementation of the strategy is as follows; see Algorithm 4.2.

- If $k = 1$, the preconditioner $B^{(1)}$ is generated by calling Algorithm 3.1 (Remark 2);
- If $k > 1$, the establishment of the preconditioner $B^{(k)}$ can be viewed as the following two steps. First, we obtain $It^{(k-1)}$, which is the number of iterations obtained by solving the previous Jacobian system $A^{(k-1)}x^{(k-1)} = b^{(k-1)}$. Furthermore, there are two situations when judging the size of $It^{(k-1)}$ and μ (given a threshold greater than or equal to 0). If $It^{(k-1)} \leq \mu$, the preconditioner $B^{(k)}$ adopts the previous preconditioner $B^{(k-1)}$; otherwise, the preconditioner $B^{(k)}$ is generated by calling Algorithm 3.1.

Remark 2. Algorithm 3.1 is a preconditioning method ($w = \text{CPR}(A, g, w_0)$ i.e., $w = Bg$). For the convenience of describing Algorithms 4.1 and 4.2, we assume Algorithm 3.1 creates a CPR preconditioner B .

Remark 3. If the sizes of matrices $A^{(k-1)}$ and $A^{(k)}$ are not the same, we must regenerate the preconditioner $B^{(k)}$.

Algorithm 4.1 ASCPR-GMRES method

Input: $B^{(k-1)}, It^{(k-1)}, A^{(k)}, b^{(k)}, x_0, \mu, k, m, tol, MaxIt$;
Output: $x^{(k)}, It^{(k)}, B^{(k)}$;

- 1 $B^{(k)} = \text{ASCPR}(B^{(k-1)}, It^{(k-1)}, \mu, k)$;
- 2 Compute $r_0 \leftarrow b - A^{(k)}x_0$, $p_1 \leftarrow r_0/\|r_0\|$;
- 3 **for** $It = 1, \dots, MaxIt$ **do**
- 4 **for** $j = 1, \dots, m$ **do**
- 5 Compute $\bar{p} \leftarrow A^{(k)}(B^{(k)}p_j)$;
- 6 Compute $h_{i,j} \leftarrow (\bar{p}, p_i), i = 1, \dots, j$;
- 7 Compute $\tilde{p}_{j+1} \leftarrow \bar{p} - \sum_{i=1}^j h_{i,j}p_i$;
- 8 Compute $h_{j+1,j} \leftarrow \|\tilde{p}_{j+1}\|$;
- 9 **if** $h_{j+1,j} = 0$ **then**
- 10 $m \leftarrow j$; **break**;
- 11 **end**
- 12 Compute $p_{j+1} \leftarrow \tilde{p}_{j+1}/h_{j+1,j}$;
- 13 **end**
- 14 Solve the following minimization problem:

$$y_m = \arg \min_{y \in \mathbb{R}^m} \|\beta e_1 - \bar{H}_m y\|,$$

where $\beta = \|p_1\|$, $e_1 = (1, 0, \dots, 0)^T \in \mathbb{R}^{m+1}$, $\bar{H}_m := (h_{i,j}) \in \mathbb{R}^{(m+1) \times m}$;
- 15 $x_m \leftarrow x_0 + B^{(k)}(P_m y_m)$, here $P_m := (p_1, p_2, \dots, p_m)$;
- 16 Compute $r_m \leftarrow b - A^{(k)}x_m$;
- 17 **if** $\|r_m\|/\|r_0\| < tol$ **then**
- 18 **break**;
- 19 **else**
- 20 $x_0 \leftarrow x_m$;
- 21 $p_1 \leftarrow r_m/\|r_m\|$;
- 22 **end**
- 23 **end**
- 24 $x^{(k)} \leftarrow x_m, It^{(k)} \leftarrow It$;
- 25 **return** $x^{(k)}, It^{(k)}, B^{(k)}$.

Algorithm 4.2 ASCPR method

Input: $B^{(k-1)}, It^{(k-1)}, \mu, k$;
Output: $B^{(k)}$;

- 1 **if** $k > 1$ and $It^{(k-1)} \leq \mu$ **then**
- 2 $B^{(k)} \leftarrow B^{(k-1)}$;
- 3 **else**
- 4 The preconditioner $B^{(k)}$ is generated by calling Algorithm 3.1.
- 5 **end**
- 6 **return** $B^{(k)}$.

We introduce a practical threshold μ as a criterion for the adaptive SETUP preconditioner, which aims to improve the performance of the ASCPR-GMRES. To begin with, we explain the main idea of this approach. The $It^{(k-1)} \leq \mu$ means $B^{(k-1)}$ is an effective preconditioner for

Jacobian system $A^{(k-1)}x^{(k-1)} = b^{(k-1)}$ because the smaller number of iterations $It^{(k-1)}$, the more $B^{(k-1)}$ approximates the inverse of matrix $A^{(k-1)}$. Because the structure of these matrices is similar and the CPR preconditioner does not require high accuracy, the preconditioner $B^{(k-1)}$ can also be used as a preconditioner for Jacobian system $A^{(k)}x^{(k)} = b^{(k)}$. Moreover, the approach can improve the performance of the solver from the following two aspects.

- First, the efficiency of the solver is improved because the number of SETUP calls can be reduced.
- Second, the parallel performance of the solver is improved because the proportion of low parallel speedup is reduced in the solver.

Choosing a suitable μ is important to the performance of the solver. Finally, we discuss the choice of μ . If μ is too small, the number of SETUP calls is not considerably reduced. As a result, the performance of the solver is not significantly improved. Specifically, when $\mu = 0$, ASCPR-GMRES degenerates into CPR-GMRES. On the contrary, if μ is too large, the number of iterations of the solver is dramatically increased, thereby affecting the performance of the solver. Generally, the optimal μ is determined through numerical experiments according to concrete problems.

5. A multi-color GS method. In this section, we propose a parallel GS algorithm from the algebraic point of view, aiming to overcoming the limitations of the conventional red-black GS algorithm; it yields the same convergence behavior as the corresponding single-thread algorithm and obtains a good parallel speedup.

To this end, the concept of an adjacency graph is introduced. Note that an adjacency graph corresponds to a sparse matrix, and the nonzero entries of the matrix reflect the connectivity relationship between vertices in the graph. Assume that a sparse matrix $A \in \mathbb{R}^{n \times n}$ is symmetric. Let $G_A(V, E)$ be the (undirected) adjacency graph corresponding to the sparse matrix $A = (a_{ij})_{n \times n}$, where $V = \{v_1, v_2, \dots, v_n\}$ is the vertices set, and $E = \{(v_i, v_j) : \forall i \neq j, a_{ij} \neq 0\}$ is the edges set (each nonzero entry a_{ij} on the non diagonal of A corresponds to an edge (v_i, v_j)).

We are now in the position to give the design goals of this algorithm for grouping vertices and the parallel GS implementation based on the strong connections of A .

5.1. Algorithm design goals. The goal of our algorithm design is to divide the vertices set V into c subsets V_1, V_2, \dots, V_c ($1 \leq c \leq n$), and these subsets shall satisfy the following four conditions:

- $V = V_1 \cup V_2 \dots \cup V_c$;
- $V_i \cap V_j = \emptyset$, $i \neq j$, $1 \leq i, j \leq c$;
- Vertices in any subset are not connected, i.e., $a_{ij} = a_{ji} = 0, \forall v_i, v_j \in V_\ell$ ($\ell = 1, \dots, c$);
and
- The number of subsets c should be as small as possible.

It is easy to see that the smaller the number of groupings c , the more difficult the grouping, and the larger the parallel granularity. The classic GS is, in fact, equivalent to the situation when c is equal to n .

Remark 4. The red-black GS algorithm also satisfies the above four conditions with $c = 2$.

5.2. Parallel GS algorithm. In 2003, [11] gave an upper bound estimate of the total number of colors based on the graph theory. To proceed, we briefly review this upper bound estimate.

Proposition 5.1 (Upper bound estimation). *The number of multi-color groupings c of undirected graph $G_A(V, E)$ does not exceed $\text{degree}(G_A(V, E)) + 1$; that is, c does not exceed the maximum number of nonzero entries in each row of matrix A .*

According to Proposition 5.1, the number of groups c depends on the maximum number of nonzero entries in each row of matrix A . When the matrix A is relatively dense (such as the coarse grid matrix in AMG), the number of groups c is large, which contradicts the condition

(d) of Section 5.1. In extreme cases, the row nonzero entries of a matrix can be equal to the order of the matrix (this implies that $c = n$). Too many groups bring up two difficulties: low efficiency of the grouping algorithm and poor parallel performance. For the latter, because there are only small number of degrees of freedom within each group, it results in fine parallel granularity.

For this reason, $G_A(V, E)$ needs to be preprocessed to strengthen its sparseness before grouping the degrees of freedom. There are many ways to enhance the sparseness of $G_A(V, E)$. When the sparseness of $G_A(V, E)$ is enhanced, the number of groups c becomes smaller. At the same time, the independence of vertex set V_ℓ ($\ell = 1, \dots, c$) becomes worse, which affects the parallel results. Therefore, choosing an appropriate strategy is of great importance to enhance the sparseness of $G_A(V, E)$. The situation is considered where the solution vector is relatively smooth. It is found that the smaller nonzero entries (in the sense of absolute value) in each row of the matrix play a negligible role when a certain degree of freedom is smoothed by GS. In this paper, we propose a strategy to filter small nonzero entries. The matrix whose small nonzero entries are filtered is called the so-called “matrix of strong connections.” This concept is defined as follows.

The matrix $A = (a_{ij})_{n \times n}$ corresponds to the matrix of strong connections $S(A, \theta)$ (denoted as S), and its entries are defined as

$$(5.1) \quad S_{ij} = \begin{cases} 1, & |a_{ij}| > \theta \sum_{k=1}^n |a_{ik}| \\ 0, & |a_{ij}| \leq \theta \sum_{k=1}^n |a_{ik}| \end{cases} \quad \forall i, j = 1, 2, \dots, n, i \neq j,$$

where θ ($0 \leq \theta \leq 1$) is a given threshold, $S_{ij} = 1$ when there is a strong adjacent edge between v_i and v_j , and $S_{ij} = 0$ means that there is no strong adjacent edge between v_i and v_j .

To describe our algorithm conveniently, we introduce the following notations.

- The set S_i represents the vertex set that is strongly connected to the vertex v_i , i.e., $S_i = \{j : S_{ij} \neq 0, j = 1, 2, \dots, n\}$.
- The set \bar{S}_i represents the vertex set that is strongly connected to the vertex v_i (including v_i) and whose colors are undetermined. That is, $\bar{S}_i = \{j : j \in S_i \cup \{i\} \text{ and color of } j \text{ is undetermined}\}$.
- The set \hat{S}_i represents the vertex set that is the next most strongly connected to the vertex v_i (the vertices on “the second circle”) and whose colors are undetermined. That is, $\hat{S}_i = \{j : j \in W_i \text{ and color of } j \text{ is undetermined}\}$, where $W_i = \{j : \forall k \in S_i, j \in S_k / (S_i \cup \{i\})\}$.
- The cardinality $|\cdot|$ represents the number of entries in the set \cdot . In particular, $|S_i|$ represents the influence value of the vertex v_i .

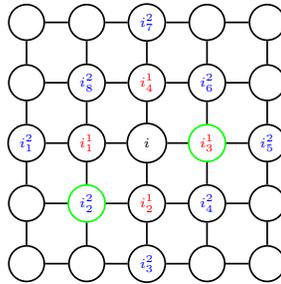


Fig. 3. Schematic diagram of notations explanation.

Let us explain these notations with a simple schematic diagram (see Fig. 3). Assume

that the all edges are strongly adjacent edges in Fig. 3; moreover, the vertices i_3^1 and i_2^2 are given the splitting attribute (i.e., they have their own color). Hence, $S_i = \{i_1^1, i_2^1, i_3^1, i_4^1\}$, $\bar{S}_i = \{i, i_1^1, i_2^1, i_3^1\}$, $\hat{S}_i = \{i_1^2, i_3^2, i_4^2, i_5^2, i_6^2, i_7^2, i_8^2\}$, and $|S_i| = 4$.

In the following, our algorithms are presented. To begin, we propose a greedy splitting algorithm for the vertices set V based on the matrix of strong connections (denoted as VerticesSplitting); see Algorithm 5.1. The proposed Algorithm 5.2 gives a vertices grouping algorithm corresponding to the matrix A (denoted as VerticesGrouping).

Algorithm 5.1 VerticesSplitting method

Input: V, S ;
Output: W, \bar{W} ;

```

1 Set  $W \leftarrow \emptyset, \bar{W} \leftarrow \emptyset, \widehat{W} \leftarrow \emptyset$ ;
2 while  $V \neq \emptyset$  do
3   if  $\widehat{W} \neq \emptyset$  then
4     | Any take  $v_i \in \widehat{W}$  and  $|S_i| \geq |S_j|, \forall v_i, v_j \in \widehat{W}$ ;
5   else
6     | Any take  $v_i \in V$  and  $|S_i| \geq |S_j|, \forall v_i, v_j \in V$ ;
7   end
8   if  $v_i$  is not strongly connected to any vertices in the set  $W$  (i.e.,  $S_{ij} = 0, \forall j \in W$ ) then
9     |  $W \leftarrow W \cup v_i, V \leftarrow V/v_i$ ;
10    | if  $v_i \in \widehat{W}$  then
11      |  $\widehat{W} \leftarrow \widehat{W}/v_i$ ;
12    | end
13    |  $\bar{W} \leftarrow \bar{W} \cup S_i$ ;
14    |  $V \leftarrow V/S_i$ ;
15    |  $\widehat{W} \leftarrow \widehat{W} \cup \hat{S}_i$ ;
16  else
17    |  $\bar{W} \leftarrow \bar{W} \cup v_i$ ;
18    |  $V \leftarrow V/v_i$ ;
19    | if  $v_i \in \widehat{W}$  then
20      |  $\widehat{W} \leftarrow \widehat{W}/v_i$ ;
21    | end
22  end
23 end
24 return  $W, \bar{W}$ .
```

Algorithm 5.2 VerticesGrouping method

Input: V, S ;
Output: V_ℓ ($\ell = 1, \dots, c$);

```

1 Set  $c \leftarrow 0$ ;
2 while  $V \neq \emptyset$  do
3   |  $c \leftarrow c + 1$ ;
4   | Call Algorithm 5.1 to generate  $V_c$  and  $\bar{V}_c$ ;
5   | Let  $V \leftarrow \bar{V}_c$ ;
6 end
7 return  $V_\ell$  ( $\ell = 1, \dots, c$ ).
```

As can be easily noticed from Algorithms 5.1 and 5.2, the grouping numbers c and the

degree of independence of the vertices set V_ℓ ($\ell = 1, \dots, c$) depend on the choice of strength threshold θ . The smaller the value of θ , the better the degree of independence of the vertices set, but the greater the c . Especially, when $\theta = 0$, the degree of independence of the vertices set is the best (complete independence), but c is the largest. Moreover, when $\theta = 1$, the degree of independence of the vertices set is the worst, and $c = 1$ (at this time, the proposed algorithm degenerates to the classic GS algorithm). In this sense, it is necessary to balance the grouping numbers and the degree of independence within the vertices set. In order to better satisfy condition (d), we use an approach to weaken the condition (c) slightly in our algorithms. We expect that this approach can slightly improve the parallel performance.

Next, two properties of our proposed algorithms are given.

Proposition 5.2 (Matrix diagonalization). *The block matrix $A_{V_\ell V_\ell}$ (with V_ℓ as the row and column indices, $\ell = 1, \dots, c$) is the diagonal matrix if the row and column indices of matrix A are rearranged by the indices set $\{V_\ell\}_{\ell=1}^c$.*

Proposition 5.3 (Finite termination). *Algorithm 5.2 terminates within a finite number of steps; that is, $c \leq |V|$.*

Proof. To prove that Algorithm 5.2 terminates within a finite step, just prove $V_c \neq \emptyset$ in line 5 of Algorithm 5.2. According to Algorithm 5.1, if $V \neq \emptyset$, V_c (i.e., the output variable W of Algorithm 5.1) contains at least one vertex. From lines 3–7 of Algorithm 5.2, $c \leq |V|$ can be obtained. ■

Note that the Algorithm 5.2 can split $G_A(V, E)$ into subgraphs $G_{A_\ell}(V_\ell, E_\ell)$, $\ell = 1, \dots, c$. The $S(A_\ell, \theta)$ is denoted as the adjacency matrix corresponding to each subgraph. Moreover, each subgraph corresponds to a submatrix $A_\ell(V_\ell)$ of matrix A . According to Proposition 5.2, the diagonal blocks of the submatrix $A_\ell(V_\ell)$ are diagonal, and the GS smoothing of the submatrix $A_\ell(V_\ell)$ is completely parallel at this time. Furthermore, a parallel (or multi-color) GS algorithm based on the strong connections of the matrix is given by Algorithm 5.3, denoted as PGS-SCM.

Algorithm 5.3 PGS-SCM method

Input: A, x, b, θ ;

Output: x ;

- 1 Using the matrix A and the formula (5.1) to generate vertices set V and matrix of strong connections S , respectively;
 - 2 Call Algorithm 5.2 to generate independent vertices subset V_ℓ ($\ell = 1, \dots, c$);
 - 3 Using V_ℓ to split the matrix A into submatrix A_ℓ ($\ell = 1, \dots, c$);
 - 4 **for** $\ell = 1, \dots, c$ **do**
 - 5 | Parallel call the classic GS algorithm of submatrix A_ℓ ;
 - 6 **end**
 - 7 **return** x .
-

Finally, the proposed algorithms can be parallelized for multi-core and many-core architectures. We develop the OpenMP version and the CUDA version of the parallel program, respectively. Furthermore, these algorithms are integrated into the FASP framework. The results of the numerical experiments will be given in the next section.

6. Numerical experiments. In this section, to demonstrate the performance of the proposed methods, we consider the two-phase and three-phase test problems based on the SPE10 benchmark. The numerical experiments are performed on a machine with Intel Xeon Platinum 8260 CPU (32 cores, 2.40GHz), 128GB DRAM, and NVIDIA Tesla T4 GPU (2560 cores, 16GB Memory).

Here we provide some details of the ASCPR-GMRES method. For the CPR preconditioner, in the first stage, we use the Unsmoothed Aggregation AMG (UA-AMG) method [37] to approximate the inverse of the pressure coefficient matrix, where the aggregation strategy is the so-called non-symmetric pairwise matching aggregation (NPAIR) [38], the cycle type is the Nonlinear AMLI-cycle [39], the smoothing operator is PGS-SCM, the degree of freedom of the coarsest space is set to be 10000, and the coarsest space solver is a direct solver. In the second stage, we use the BILU method based on Level Scheduling (LS) [11] to approximate the inverse of the coefficient matrix. For the restarted GMRES method, the restarting number m is 28, the maximum number of iterations $MaxIt$ is 100, and the tolerance error of relative residual norm tol is 10^{-5} .

6.1. Two-phase SPE10. The standard two-phase SPE10 [40] benchmark is tested to demonstrate the performance of the proposed methods. The model dimensions are $1200 \times 2200 \times 170$ (ft), and the number of grid cells is $60 \times 220 \times 80$ (the total number of grid cells is 1,122,000 and the number of active cells is 1,094,422). First, we verify convergence behavior and parallel performance of the PGS-SCM method. Furthermore, we test the parallel performance for the ASCPR-GMRES-OMP and ASCPR-GMRES-CUDA methods, where the ASCPR-GMRES-OMP and ASCPR-GMRES-CUDA correspond to solver versions in OpenMP and CUDA, respectively. Finally, our results based on the in-house simulator of PetroChina, HiSim 2.0 [31], are compared with the results obtained using a commercial simulator, tNavigator (2020 version) [41].

6.1.1. PGS-SCM method. To evaluate the convergence behavior and parallel performance of the proposed PGS-SCM method, we employ the CPR-GMRES method as the solver for the petroleum reservoir simulation. Furthermore, we employ the parallel GS method based on natural ordering (denoted as PGS-NO) as a reference for comparison.

Example 6.1. The two-phase example is considered, and the numerical simulation conducted for 2000 days. We use the different number of threads ($NT = 1, 2, 4, 8, \text{ and } 16$) to test the parallel performance of CPR-GMRES-PGS-NO and CPR-GMRES-PGS-SCM. The impacts of the different strength thresholds θ ($\theta = 0, 0.05, 0.1, \text{ and } 0.3$) on CPR-GMRES-PGS-SCM are also tested.

Tab. 1 lists the total number of linear iterations ($Iter$), the total wall time in seconds ($Time$), and the parallel speedup ($Speedup$ defined in Remark 5).

Remark 5. The calculation formula of speedup is

$$Speedup = \frac{T_1}{T_n},$$

where T_1 represents the wall time obtained by a single thread (core), and T_n represents the wall time obtained by n threads.

It can be seen from Tab. 1 that, for CPR-GMRES-PGS-NO, the total number of linear iterations gradually increases as NT increases. In particular, if $NT = 16$, the total number of linear iterations increases by 97 compared with the single-thread case (the speedup is about 5.28). On the other hand, the number of iterations of the parallel GS algorithm based on natural ordering is not as stable. That is, as the number of threads increases, the number of iterations increases, which affects the parallel speedup as well. However, for CPR-GMRES-PGS-SCM (when $\theta = 0$), the total number of linear iterations is not changed with a larger NT . Such results indicate that the proposed PGS-SCM method yields same convergence behavior as the corresponding single-thread algorithm. Moreover, from the perspective of parallel performance, CPR-GMRES-PGS-SCM gets a higher speedup compared with CPR-GMRES-PGS-NO. For example, when $NT = 16$, the speedup of CPR-GMRES-PGS-SCM is also higher (the speedup

Tab. 1

Iter, Time(s), and Speedup of the two solvers with different NT for the two-phase SPE10 problem.

Solvers	θ	NT	1	2	4	8	16
CPR-GMRES-PGS-NO	—	<i>Iter</i>	5823	5826	5842	5882	5920
		<i>Time</i>	4701.26	2577.35	1497.12	988.09	890.78
		<i>Speedup</i>	1.00	1.82	3.14	4.76	5.28
CPR-GMRES-PGS-SCM	0	<i>Iter</i>	5837	5837	5837	5837	5837
		<i>Time</i>	4780.42	2610.02	1501.12	976.80	847.26
		<i>Speedup</i>	1.00	1.83	3.18	4.89	5.64
	0.05	<i>Iter</i>	5823	5822	5818	5822	5827
		<i>Time</i>	4753.00	2593.07	1491.97	970.26	829.24
		<i>Speedup</i>	1.00	1.83	3.19	4.90	5.73
	0.1	<i>Iter</i>	5832	5833	5826	5825	5846
		<i>Time</i>	4782.52	2599.12	1497.25	977.93	850.82
		<i>Speedup</i>	1.00	1.84	3.19	4.89	5.62
	0.3	<i>Iter</i>	5821	5839	5841	5876	5912
		<i>Time</i>	4732.72	2615.05	1510.71	981.28	872.32
		<i>Speedup</i>	1.00	1.81	3.13	4.82	5.43

is about 5.73).

Next, we discuss the influence of the strength threshold θ on the parallel performance for the CPR-GMRES-PGS-SCM solver. When θ is small (for example, $\theta = 0.0$ or 0.05), the total number of linear iterations changes little when NT increases. However, if θ gets larger (for example, $\theta = 0.1$ or 0.3), the varied range of the total number of linear iterations is enlarged with the increase of NT. If $NT = 16$ is considered, as θ increases, the total number of linear iterations first decreases and then increases, and the speedup first increases and then decreases. In particular, when $\theta = 0.05$, the minimum total number of linear iterations is 5827, and the speedup is the highest. This shows that the strength threshold θ affects the convergence as well as the parallel performance.

6.1.2. ASCPR-GMRES-OMP method.

Example 6.2. For the two-phase SPE10 example, the numerical simulation is conducted for 2000 days, and $\theta = 0.05$. We explore the parallel performance of ASCPR-GMRES-OMP for four different values of μ (i.e., $\mu = 0, 20, 30$, and 40), when $NT = 1, 2, 4, 8$, and 16 , respectively.

To assess the effects of different thresholds μ on the performance of ASCPR-GMRES-OMP for the two-phase SPE10 problem, Tab. 2 lists the number of SETUP calls (*SetupCalls*), the ratio of SETUP in the total solution time (*SetupRatio*), the total number of linear iterations (*Iter*), total solution time in seconds (*Time*), new parallel speedup (*Speedup**, see Remark 6), and parallel speedup (*Speedup*).

Remark 6. In order to compare the parallel speedup of ASCPR-GMRES-OMP fairly, we

Tab. 2

SetupCalls, SetupRatio, Iter, Time(s), Speedup, and Speedup for the two-phase SPE10 problem.*

	μ	1	2	4	8	16
<i>SetupCalls</i>	0	239	239	239	239	239
	20	188	188	188	188	188
	30	58	58	58	58	58
	40	33	33	33	33	33
<i>SetupRatio</i>	0	11.08%	14.59%	20.63%	29.34%	41.38%
	20	9.45%	12.46%	17.55%	25.45%	38.14%
	30	5.19%	6.52%	9.20%	14.05%	26.09%
	40	4.05%	4.99%	6.62%	11.03%	21.83%
<i>Iter</i>	0	5823	5822	5818	5822	5827
	20	5855	5854	5853	5855	5857
	30	6309	6308	6315	6317	6319
	40	7033	7034	7037	7041	7044
<i>Time</i>	0	4753.00	2593.07	1491.97	970.26	829.24
	20	4821.98	2606.18	1488.95	949.98	804.56
	30	4919.81	2617.66	1444.92	877.85	718.13
	40	5475.72	2880.97	1570.72	946.35	769.34
<i>Speedup*</i>	0	1.00	1.83	3.19	4.90	5.73
	20	0.99	1.82	3.19	5.00	5.91
	30	0.97	1.82	3.29	5.41	6.62
	40	0.87	1.65	3.03	5.02	6.18
<i>Speedup</i>	0	1.00	1.83	3.19	4.90	5.73
	20	1.00	1.85	3.24	5.08	5.99
	30	1.00	1.88	3.40	5.60	6.85
	40	1.00	1.90	3.49	5.79	7.12

propose a new parallel speedup (denoted as $Speedup^*$), and it is defined as follows:

$$Speedup^* = \frac{T_1^0}{T_n^\mu},$$

where T_1^0 represents the wall time obtained by a single thread when the general preconditioner ($\mu = 0$) is used, and T_n^μ represents the wall time obtained by n threads when the adaptive SETUP threshold μ is used.

From Tab. 2, we first look at the number of SETUP calls, the ratio of SETUP in the total solution time, and the total number of linear iterations. When NT is fixed, both the number of SETUP calls and the ratio of SETUP in the total solution time decrease as μ increases. In particular, when $\mu = 40$, there are only 33 SETUP calls. When μ is fixed, the number of SETUP calls does not change with respect to NT, but the ratio of SETUP in the total solution time is increasing. If NT is fixed, the total number of linear iterations is increasing as μ increases. These observations indicate that the number of SETUP calls is significantly decreased with the increase of μ , while the total number of linear iterations is also increased. Furthermore, we focus on the total solution time and the new parallel speedup. When NT = 16 and $\mu = 30$, the total solution time is 718.13 (s), and the corresponding new parallel speedup is 6.62. Compared with the regular CPR preconditioner, the total solution time of ASCPR-GMRES-OMP is reduced from 829.24 (s) to 718.13 (s), and the new parallel speedup is increased from 5.73 to 6.62. These results show that the proposed method can improve the parallel performance of the solver.

6.1.3. ASCPR-GMRES-CUDA method.

Example 6.3. For the two-phase SPE10 example, the numerical simulation is conducted for 2000 days, and $\theta = 0.05$. We explore the impacts of μ ($\mu = 0, 20, 30$, and 40) on the parallel performance of ASCPR-GMRES-CUDA.

Tab. 3 lists *SetupCalls*, *SetupRatio*, *Iter*, *Time*, and *Speedup* of ASCPR-GMRES-CUDA for the two-phase SPE10 problem. The single-thread calculation results of the OpenMP version program ASCPR-GMRES-OMP(1) are also added for comparison. For ASCPR-GMRES-CUDA solver, as μ increases, both the number of SETUP calls and the ratio of SETUP in the total solution time decrease, the total number of linear iterations gradually increases, and the parallel speedup increases. The CUDA version can be 6.22 times faster than the ASCPR-GMRES-OMP(1). In case $\mu = 40$, the total solution time of ASCPR-GMRES-CUDA is reduced from 763.66 (s) to 424.60 (s), which can be 1.80 times faster compared with $\mu = 0$. At the same time, compared with ASCPR-GMRES-OMP(1), the speedup of ASCPR-GMRES-CUDA reaches 11.19.

Tab. 3

SetupCalls, SetupRatio, Iter, Time(s), and Speedup (compared to ASCPR-GMRES-OMP(1)) of the different μ for the two-phase SPE10 problem.

Solvers	μ	<i>SetupCalls</i>	<i>SetupRatio</i>	<i>Iter</i>	<i>Time</i>	<i>Speedup</i>
ASCPR-GMRES-OMP(1)	0	239	12.22%	5823	4753.00	—
	0	239	71.39%	5525	763.66	6.22
ASCPR-GMRES-CUDA	20	185	66.11%	5659	677.13	7.02
	30	52	41.99%	6182	432.45	10.99
	40	34	34.58%	6740	424.60	11.19

Finally, we present the effects of different μ on the timestep size Δt for ASCPR-GMRES-OMP and ASCPR-GMRES-CUDA solvers, see Fig. 4. It can be seen from Fig. 4(a) that when $\mu = 0$, the timestep size is consistent for ASCPR-GMRES-OMP with different thread numbers. From Figs. 4(b) and 4(c), when μ changes, the timestep sizes of the ASCPR-GMRES-OMP and ASCPR-GMRES-CUDA solvers rarely change, and the effect on the overall numerical results can be ignored.

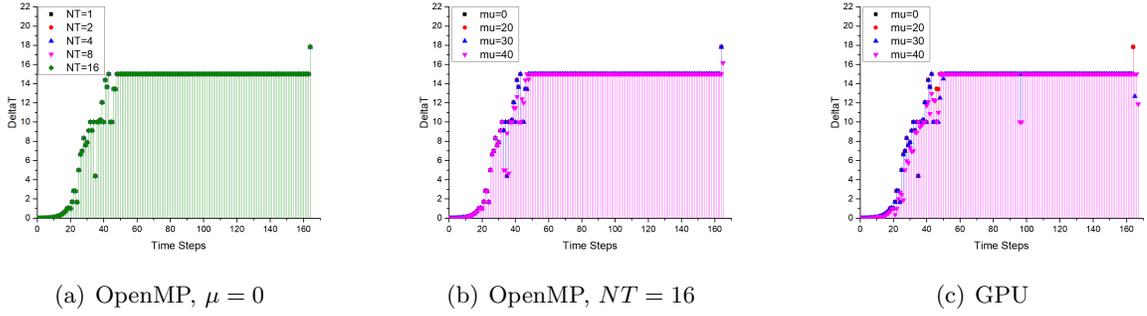


Fig. 4. The timestep size Δt of ASCPR-GMRES-OMP and ASCPR-GMRES-CUDA for the two-phase SPE10 problem.

6.1.4. Performance comparisons with commercial simulator. To better evaluate the performance of the proposed methods, we also test the same problem with a commercial simulator for comparison. The default solving method and parameters are used, where the maximum number of iterations $MaxIt$ is 1000 and the tolerance of relative residual norm tol is set to 10^{-5} . We compare the experimental results of the OpenMP and GPU versions for commercial and our simulators, respectively.

To begin with, Figs. 5 and 6 display the field oil production rate and average pressure graphs. Through quantitative comparison, it can be found that our simulation results are consistent with the results of commercial simulator.

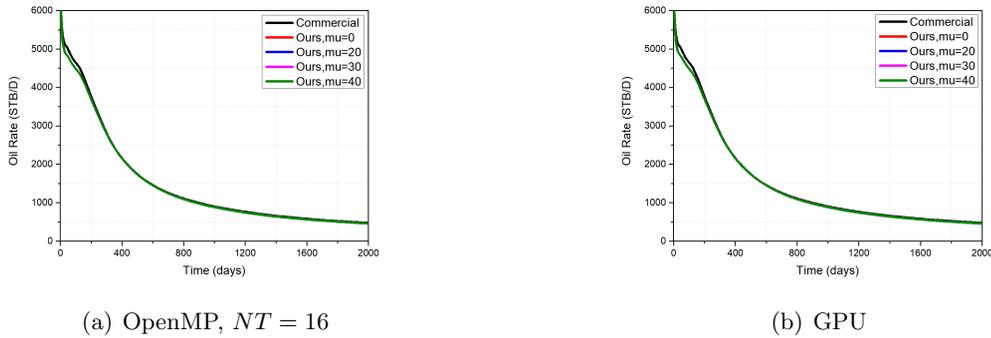


Fig. 5. Field oil production rate of OpenMP and GPU for the two-phase SPE10 problem.

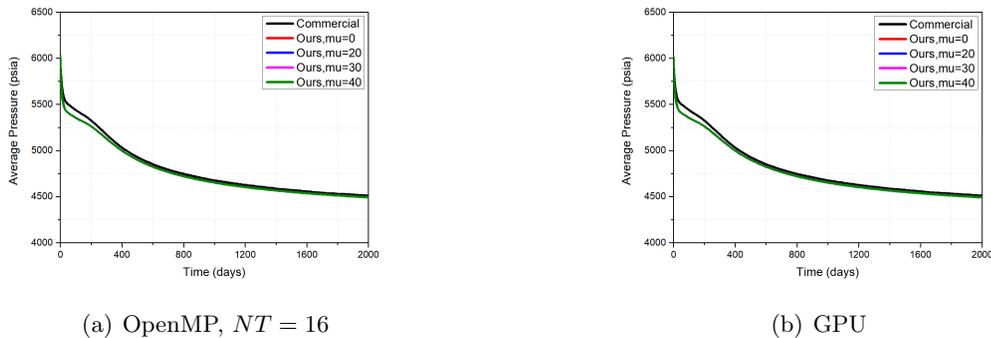


Fig. 6. Average pressure of OpenMP and GPU for the two-phase SPE10 problem.

Tab. 4

NumTSteps, *NumNSteps*, *Iter*, *AvgIter*, *Time(h)*, and *Speedup* comparisons of the OpenMP version of commercial and our simulators for the two-phase SPE10 problem.

Simulators	μ	NT	1	2	4	8	16
Commercial	—	<i>NumTSteps</i>	671	785	891	1031	1100
		<i>NumNSteps</i>	1115	1244	1328	1458	1515
		<i>Iter</i>	150152	160254	161432	167757	175452
		<i>AvgIter</i>	134.7	128.8	121.6	115.1	115.8
		<i>Time</i>	14.62	8.45	4.71	2.72	1.54
		<i>Speedup</i>	1.00	1.73	3.10	5.37	9.49
Ours	0	<i>NumTSteps</i>	164	164	164	164	164
		<i>NumNSteps</i>	239	239	239	239	239
		<i>Iter</i>	5823	5822	5818	5822	5827
		<i>AvgIter</i>	24.4	24.4	24.3	24.4	24.4
		<i>Time</i>	1.48	0.88	0.57	0.43	0.39
		<i>Speedup</i>	1.00	1.68	2.60	3.44	3.79
	20	<i>NumTSteps</i>	164	164	164	164	164
		<i>NumNSteps</i>	239	239	239	239	239
		<i>Iter</i>	5855	5854	5853	5855	5857
		<i>AvgIter</i>	24.5	24.5	24.5	24.5	24.5
		<i>Time</i>	1.50	0.88	0.57	0.42	0.38
		<i>Speedup</i>	1.00	1.70	2.63	3.57	3.95
	30	<i>NumTSteps</i>	164	164	164	164	164
		<i>NumNSteps</i>	239	239	239	239	239
		<i>Iter</i>	6309	6308	6315	6317	6319
		<i>AvgIter</i>	26.4	26.4	26.4	26.4	26.4
		<i>Time</i>	1.52	0.88	0.56	0.40	0.36
		<i>Speedup</i>	1.00	1.73	2.71	3.80	4.22
	40	<i>NumTSteps</i>	164	164	164	164	165
		<i>NumNSteps</i>	246	246	246	246	247
		<i>Iter</i>	7033	7034	7037	7041	7064
		<i>AvgIter</i>	28.6	28.6	28.6	28.6	28.6
		<i>Time</i>	1.68	0.96	0.60	0.42	0.37
		<i>Speedup</i>	1.00	1.75	2.80	4.00	4.54

Tabs. 4 and 5 present the number of time steps (*NumTSteps*), the number of Newton iterations (*NumNSteps*), the number of linear iterations (*Iter*), the average number of linear iterations per Newton iteration (*AvgIter*), the total simulation time (*Time*), and the parallel speedup (*Speedup*) for the OpenMP and GPU versions of commercial and our simulators, respectively. For the OpenMP version of the commercial simulator, when the number of threads is changed from 1 to 16, the simulation time is reduced from 14.62 (h) to 1.54 (h). At this point, the maximum speedup reaches 9.49 and the average number of linear iterations per Newton iteration exceeds 115. In our simulator, when $\mu = 30$ and the number of threads varied from 1

to 16, the simulation time was reduced from 1.52 (h) to 0.36 (h). At this point, the maximum speedup reaches 4.22 and the average number of linear iterations per Newton iteration is 26.4. This indicates that the proposed method can speed up the simulation time by 4.27 times compared with the commercial simulator. On the one hand, the commercial software chooses some algorithms with high parallel speedup. Usually, such algorithms are easier to parallelize, while they take more iterations to converge. It can be seen from the average number of linear iterations per Newton iteration—Its *AvgIter* is about 4 times more than ours. On the other hand, increasing the number of iterations also improves the parallel speedup. This is because the proportion of solver in the SOLVE increases, and the parallel speedup of the SOLVE phase is usually higher than that of SETUP. Finally, it is worth mentioning that we only parallelize the linear solver in our simulator, while the rest of our simulator is still sequential.

Tab. 5

NumTSteps, NumNSteps, Iter, AvgIter, Time(h), and Speedup (compared with the commercial simulator) comparisons of the GPU version of commercial and our simulators for the two-phase SPE10 problem.

Simulators	μ	<i>NumTSteps</i>	<i>NumNSteps</i>	<i>Iter</i>	<i>AvgIter</i>	<i>Time</i>	<i>Speedup</i>
Commercial	—	1004	1431	170276	119.0	3.070	—
Ours	0	164	239	5525	23.1	0.387	7.93
	20	164	240	5659	23.6	0.358	8.57
	30	165	240	6182	25.8	0.280	10.97
	40	167	244	6740	27.6	0.278	11.05

According to Tab.5, we can summarize similar conclusions for GPUs. Compared with the commercial simulator, our obtained numbers of time steps, Newton iterations, and linear iterations are much smaller, and the simulation time is shorter. Especially when $\mu = 40$, the speedup of our simulator can achieve 11.05 compared to commercial simulator.

6.2. Three-phase SPE10. A three-phase benchmark problem is obtained by changing the fluid properties of the original two-phase SPE10 [40]. We provide modification information from two-phase to three-phase for the SPE10 benchmark problem. Some keywords and values are added to the input file of the original two-phase SPE10 problem. We change the phase states, fluid properties, and relative permeabilities as follows.

- Phase states:

Tab. 6

Phase states for two-phase and three-phase.

Two-phase	Three-phase
OIL	OIL
WATER	WATER
	GAS
	DISGAS

Remark 7. Note that the keyword “DISGAS” indicates that dissolved gas in oil is considered in three-phase example.

- Fluid properties:
where P_o , B_o , and μ_o denote the pressure, volume coefficient, and viscosity coefficient of oil phase, respectively. P_{bub} and R_{so} denote the bubble point pressure for oil and

Tab. 7

PVDO: PVT properties of dead oil (no dissolved gas) for two-phase. PVCO: PVT properties of live oil in compressibility form (with dissolved gas) for three-phase. Left: PVDO, right: PVCO.

P_o	B_o	μ_o	P_{bub}	R_{so}	B_o	μ_o	κ_o	vc_o
300	1.05	2.85	400	0.0165	1.01200	3.5057	1.1388e-6	0
800	1.02	2.99	4000	0.1130	1.01278	2.9972	1.1388e-6	0
8000	1.01	3.00	10000	0.1810	1.15500	2.6675	1.1388e-6	0

dissolved gas-oil ratio; B_o and μ_o denote the volume coefficient and viscosity coefficient of saturated oil; κ_o and vc_o denote the compressibility and viscosity compressibility of undersaturated oil.

Tab. 8

PMAX: Maximum pressure during the simulation.

P_{\max}	\hat{P}_{\max}	\hat{P}_{\min}	N_{nodes}
16000	0.0	1.0E+20	30

where P_{\max} denotes the maximum pressure that could be reached during the simulation. \hat{P}_{\max} , \hat{P}_{\min} , and N_{nodes} denote maximum pressure to extend the range of pressures, minimum pressure to extend the lower range of pressures, and the number of nodes for checking total compressibility of the system.

Tab. 9

PVDG: PVT properties of dry gas (no vaporized oil). left: two-phase, right: three-phase.

P_g	B_g	μ_g	P_g	B_g	μ_g
300	1.98	0.0162	400	1.96	0.0140
800	1.11	0.0197	4000	0.84	0.0160
8000	0.60	0.0330	8000	0.59	0.0175
			10000	0.42	0.0195

where P_g , B_g , and μ_g denote the pressure, volume coefficient, and viscosity coefficient of gas phase, respectively.

- Gas-oil relative permeabilities and capillary pressure:

where S_g , κ_{rg} , κ_{rog} , and p_{cog} denote the gas saturation, gas relative permeability, oil relative permeability (when oil, gas and connate water are present), and capillary pressure between the gas and oil phases, respectively.

Next we are going to verify convergence behavior and parallel performance of the PGS-SCM method for the three-phase SPE10 problem. Also, the correctness and parallel performance for the ASCPR-GMRES-OMP and ASCPR-GMRES-CUDA methods are tested and the performance of our and commercial simulators are compared.

6.2.1. PGS-SCM method.

Example 6.4. The three-phase example is considered, and the numerical simulation conducted for 100 days. We use the different number of threads ($NT = 1, 2, 4, 8,$ and 16) to test the parallel performance of CPR-GMRES-PGS-NO and CPR-GMRES-PGS-SCM. The impacts of the different strength thresholds θ ($\theta = 0, 0.05, 0.1,$ and 0.3) on CPR-GMRES-PGS-SCM are also tested.

Tab. 10

SGOF: gas/oil relative permeabilities and capillary pressure versus gas saturation for three-phase.

S_g	κ_{rg}	κ_{rog}	p_{cog}
0.00	0.0000	1.00	0.0
0.04	0.0000	0.60	0.2
0.10	0.0220	0.33	0.5
0.20	0.1000	0.10	1.0
0.30	0.2400	0.02	1.5
0.40	0.3400	0.00	2.0
0.50	0.4200	0.00	2.5
0.60	0.5000	0.00	3.0
0.70	0.8125	0.00	3.5
0.80	1.0000	0.00	3.9

Tab. 11 lists the experimental results of Example 6.4, including the total number of linear iterations (*Iter*), the total solution time (*Time*), and the parallel speedup (*Speedup*).

Tab. 11

Iter, Time(s), and Speedup of the two solvers with different NT for the three-phase SPE10 problem.

Solvers	θ	NT	1	2	4	8	16
CPR-GMRES-PGS-NO	—	<i>Iter</i>	3336	3338	3372	3439	3527
		<i>Time</i>	3406.71	2050.70	1258.79	804.38	687.76
		<i>Speedup</i>	1.00	1.66	2.71	4.24	4.95
CPR-GMRES-PGS-SCM	0	<i>Iter</i>	3334	3334	3334	3334	3334
		<i>Time</i>	3489.93	1937.48	1125.65	723.21	608.78
		<i>Speedup</i>	1.00	1.80	3.10	4.83	5.73
	0.05	<i>Iter</i>	3236	3236	3237	3237	3237
		<i>Time</i>	3446.89	1920.54	1102.48	711.16	598.86
		<i>Speedup</i>	1.00	1.79	3.13	4.85	5.76
	0.1	<i>Iter</i>	3379	3381	3381	3385	3385
		<i>Time</i>	3551.89	1976.98	1141.65	745.85	624.98
		<i>Speedup</i>	1.00	1.80	3.11	4.76	5.68
	0.3	<i>Iter</i>	3314	3315	3470	3496	3477
		<i>Time</i>	3421.46	1909.81	1154.44	756.08	631.75
		<i>Speedup</i>	1.00	1.79	2.96	4.53	5.42

It can be seen from Tab. 11 that, for CPR-GMRES-PGS-NO solver, the total number of linear iterations gradually increases as NT increases. When $NT = 16$, the total number of linear iterations increases by 191 compared with the single-thread result, and the speedup is 4.95. This implies that the number of iterations of the parallel GS algorithm based on natural

ordering is not stable. That is, as the number of threads increases, the number of iterations increases, which affects the parallel speedup of the solver. However, for CPR-GMRES-PGS-SCM solver (when $\theta = 0$), the total number of linear iterations is not changed with a larger NT. Such results indicate that the proposed PGS-SCM yields the same convergence behavior as the corresponding single-thread algorithm, which verifies the effectiveness of the algorithm. Moreover, from the perspective of parallel performance, CPR-GMRES-PGS-SCM gets a higher speedup compared with CPR-GMRES-PGS-NO. For example, when $NT = 16$, the speedup of CPR-GMRES-PGS-SCM increases to 5.73 (from 4.95 to 5.73). Hence, the proposed PGS-SCM method can obtain a better parallel speedup.

Next, we discuss the influence of different strength thresholds θ on the parallel performance for CPR-GMRES-PGS-SCM solver. When θ is small (for example, $\theta = 0$ or 0.05), the total number of linear iterations changes little with the increase of NT and can even be considered unchanged. However, when θ is larger (for example, $\theta = 0.1$ or 0.3), the varied range of the total number of linear iterations is enlarged with the increase of NT. These results show that as θ increases, the independence of the degrees of freedom is decreased, which affects the number of iterations. If $NT = 16$ is considered, the proposed method obtains some meaningful results. As θ increases, the total number of linear iterations first decreases and then increases, and the speedup first increases and then decreases. Especially when $\theta = 0.05$, the minimum total number of linear iterations is 3237, and the speedup is the highest, reaching 5.76. All in all, the strength threshold θ affects the stability of the number of iterations and parallel performance. In this experiment, when $\theta = 0.05$, the obtained number of iterations is the least, and the parallel speedup is the highest.

6.2.2. ASCPR-GMRES-OMP method. In order to verify the correctness and parallel performance of ASCPR-GMRES-OMP, we discuss the influences of different thresholds μ on the experimental results.

Example 6.5. For the three-phase example, the numerical simulation is conducted for 100 days, and $\theta = 0.05$. We explore the parallel performance of ASCPR-GMRES-OMP for five groups of μ (i.e., $\mu = 0, 10, 20, 30$, and 40), when $NT = 1, 2, 4, 8$, and 16, respectively.

First, to verify the correctness of ASCPR-GMRES-OMP, we present the field oil production rate, field gas production rate, and average pressure graphs of five groups μ (when $NT = 16$); see Fig. 7.

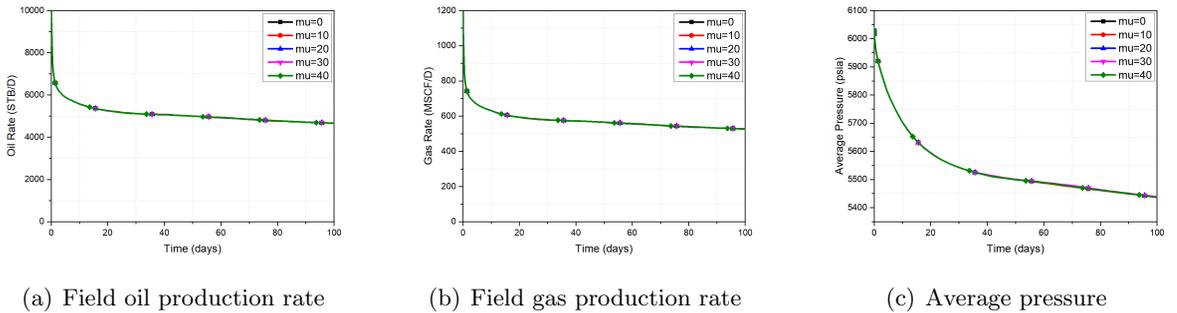


Fig. 7. Comparison charts of field oil production rate, field gas production rate, and average pressure of five groups μ for the three-phase SPE10 problem (OpenMP, $NT=16$).

From Fig. 7, we can see that the field oil production rate, field gas production rate, and average pressure obtained by the adaptive SETUP CPR preconditioner ($\mu = 10, 20, 30$, and 40) and the general CPR preconditioner ($\mu = 0$) completely coincide, indicating that the ASCPR-GMRES-OMP method is correct.

Then, we also discuss the impacts of different thresholds μ on the parallel speedup of ASCPR-GMRES-OMP. Tab. 12 lists the number of SETUP calls (*SetupCalls*), the ratio of SETUP in the total solution time (*SetupRatio*), the total number of linear iterations (*Iter*), total solution time (*Time*), new parallel speedup (*Speedup**), and parallel speedup (*Speedup*).

Tab. 12

*SetupCalls, SetupRatio, Iter, Time(s), Speedup**, and *Speedup* of different μ and NT for the three-phase SPE10 problem.

	μ	1	2	4	8	16
<i>SetupCalls</i>	0	178	178	178	178	178
	10	141	141	141	141	141
	20	98	98	98	98	98
	30	25	25	25	25	25
	40	13	13	13	13	13
<i>SetupRatio</i>	0	13.13%	16.51%	22.06%	30.68%	40.44%
	10	12.04%	15.01%	19.88%	28.25%	38.75%
	20	8.76%	10.73%	14.05%	20.74%	33.23%
	30	5.33%	6.23%	8.11%	11.39%	25.06%
	40	4.67%	5.47%	6.91%	10.53%	19.83%
<i>Iter</i>	0	3236	3236	3237	3237	3237
	10	3253	3253	3253	3253	3253
	20	3464	3463	3460	3461	3460
	30	3891	3891	3890	3889	3996
	40	4111	4111	4118	4125	4106
<i>Time</i>	0	3446.89	1920.54	1102.48	711.16	598.86
	10	3348.65	1853.37	1080.48	685.87	573.68
	20	3446.32	1885.40	1063.75	654.28	522.75
	30	3973.86	2145.97	1165.28	676.72	558.70
	40	4229.11	2239.03	1210.80	717.30	582.31
<i>Speedup*</i>	0	1.00	1.79	3.13	4.85	5.76
	10	1.03	1.86	3.19	5.03	6.01
	20	1.00	1.83	3.24	5.27	6.59
	30	0.87	1.61	2.96	5.09	6.17
	40	0.82	1.54	2.85	4.81	5.92
<i>Speedup</i>	0	1.00	1.79	3.13	4.85	5.76
	10	1.00	1.81	3.10	4.88	5.84
	20	1.00	1.83	3.24	5.27	6.59
	30	1.00	1.85	3.41	5.87	7.11
	40	1.00	1.89	3.49	5.90	7.26

According to the results of Tab. 12, we first discuss the number of SETUP calls, the ratio of SETUP in the total solution time, and the total number of linear iterations. For the number of SETUP calls and the ratio of SETUP in the total solution time, when NT is fixed, both

the number of SETUP calls and the ratio of SETUP in the total solution time decrease as μ increases. Especially when $\mu = 40$, there are only 13 SETUP calls. When μ is fixed, the number of SETUP calls is not changed with the increase of NT, but the ratio of SETUP in the total solution time is gradually increased (this also reflects fact that the parallel speedup of SOLVE is higher than that of SETUP). For the total number of linear iterations, when NT is fixed, the total number of linear iterations is gradually increasing as μ increases. In short, these results indicate that the number of SETUP calls is significantly decreased with the increase of μ , but the total number of linear iterations is also increased.

What is more, we discuss the total solution time and the new parallel speedup (only discuss the impacts of the change of μ on the results). Let's take NT=16 as an example. When μ increases by 0, 10, 20, 30, and 40 in turn, the total solution time is first decreased and then increased, and the new parallel speedup is first increased and then decreased. Especially when $\mu = 20$, the total solution time is 522.75 (s), and the corresponding the new parallel speedup is 6.59. Compared with the case of the general CPR preconditioner, the total solution time of ASCPR-GMRES-OMP is reduced from 598.86 (s) to 522.75 (s), and the new parallel speedup is increased from 5.76 to 6.59. These results show that the proposed method can further improve the parallel performance of the solver.

Finally, we discuss the changes in the parallel speedup. As μ increases, the parallel speedup is increased. Specifically, when $\mu = 40$ and NT = 16, the parallel speedup reaches 7.26. These results are consistent with what we expected. However, our goal is not to pursue the highest parallel speedup but the highest new parallel speedup (or the shortest total solution time).

6.2.3. ASCPR-GMRES-CUDA method. In order to verify the correctness and parallel performance of ASCPR-GMRES-CUDA, we explore the influences of different thresholds μ on the experimental results.

Example 6.6. For the three-phase example, the numerical simulation is conducted for 100 days, and $\theta = 0.05$. We explore the impacts of μ ($\mu = 0, 10, 20, 30$, and 40) on the parallel performance of ASCPR-GMRES-CUDA.

To verify the correctness of ASCPR-GMRES-CUDA, we plot the field oil production rate, field gas production rate, and average pressure graphs of five sets μ , as shown in Fig. 8.

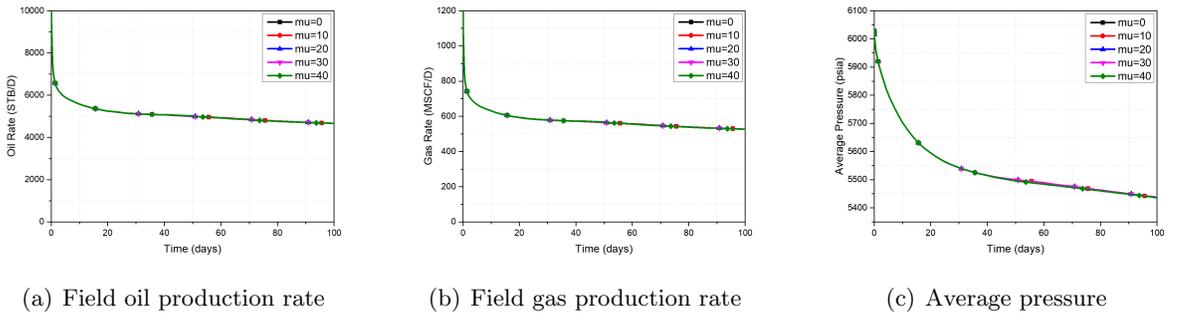


Fig. 8. The field oil production rate, field gas production rate, and average pressure comparison chart of five sets μ for the three-phase SPE10 problem (CUDA).

From Fig. 8, we can see that the field oil production rate, field gas production rate, and average pressure obtained by the adaptive SETUP CPR preconditioner (i.e., $\mu = 10, 20, 30$, and 40) and the general CPR preconditioner (i.e., $\mu = 0$) completely coincide, indicating that the ASCPR-GMRES-CUDA is correct.

In addition, we study the parallel performance of the ASCPR-GMRES-CUDA. Tab. 13 lists *SetupCalls*, *SetupRatio*, *Iter*, *Time*, and *Speedup* of ASCPR-GMRES-CUDA. The single-thread

calculation results of the OpenMP version program (denoted as ASCPR-GMRES-OMP(1)) are also added for comparison.

Tab. 13

SetupCalls, SetupRatio, Iter, Time(s), and Speedup (compared to ASCPR-GMRES-OMP(1)) of the different μ for the three-phase SPE10 problem.

Solvers	μ	SetupCalls	SetupRatio	Iter	Time	Speedup
ASCPR-GMRES-OMP(1)	0	178	13.13%	3236	3446.89	—
ASCPR-GMRES-CUDA	0	178	74.01%	3270	594.96	5.79
	10	146	70.97%	3302	534.33	6.45
	20	69	61.20%	3424	413.12	8.34
	30	24	46.24%	4065	355.80	9.69
	40	17	44.21%	4239	362.97	9.50

It can be seen from Tab. 13 that for ASCPR-GMRES-CUDA solver, as μ increases, both the number of SETUP calls and the ratio of SETUP in the total solution time decreases, the total number of linear iterations gradually increases, and the parallel speedup first increases and then decreases. The CUDA version can be 5.79 times faster than the ASCPR-GMRES-OMP(1) (when $\mu = 0$). In particular, when $\mu = 30$, compared with $\mu = 0$, the total solution time of ASCPR-GMRES-CUDA is reduced from 594.96 (s) to 355.80 (s), which can be 1.67 times faster. At the same time, compared with ASCPR-GMRES-OMP(1), the speedup of ASCPR-GMRES-CUDA reaches 9.69. Therefore, the proposed method obtains distinct acceleration effects and is more suitable for GPU architecture.

Finally, we present the effects of different μ on the timestep size Δt for ASCPR-GMRES-OMP and ASCPR-GMRES-CUDA solvers, see Fig. 9. It can be seen from Fig. 9(a) that when $\mu = 0$, the timestep size is consistent for ASCPR-GMRES-OMP with different thread numbers. From Figs. 9(b) and 9(c), the change in the timestep sizes of the ASCPR-GMRES-OMP and ASCPR-GMRES-CUDA solvers can be ignored when μ changes.

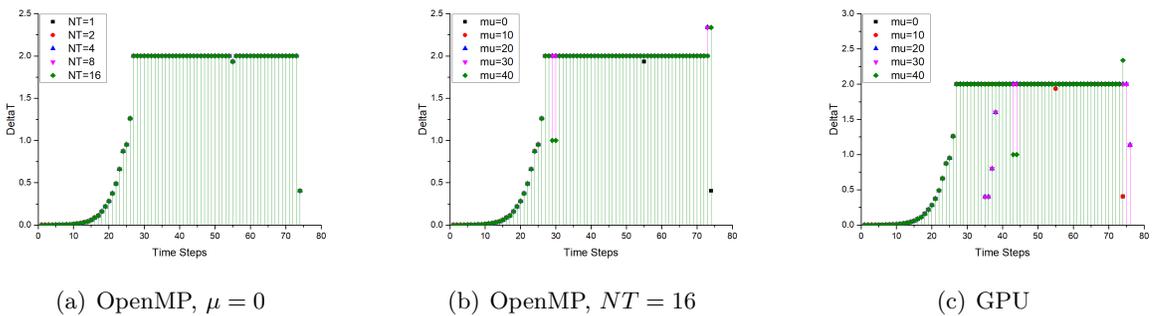


Fig. 9. The timestep size Δt of ASCPR-GMRES-OMP and ASCPR-GMRES-CUDA for the three-phase SPE10 problem.

6.2.4. Performance comparisons with commercial simulator. Also, we compare the experimental results of the OpenMP and GPU versions for commercial and our simulators, respectively. To begin with, Figs. 10(a) and 10(c) display the field oil production rate, field gas production rate, and average pressure graphs. Through quantitative comparison, it can be found that our simulation results are very consistent with the results of commercial simulator.

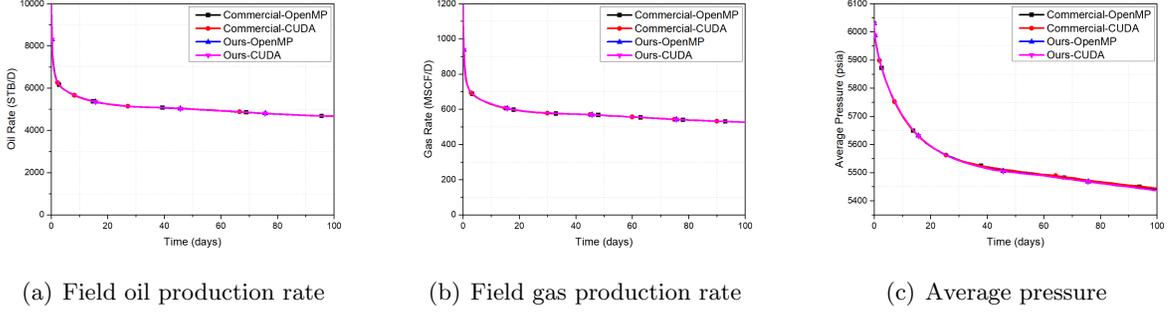


Fig. 10. The field oil production rate, field gas production rate, and average pressure comparison charts of commercial and our simulators for the three-phase SPE10 problem. OpenMP version: $NT = 16$, Ours: $\mu = 0$.

Next, Tabs. 14 and 15 present the number of time steps ($NumTSteps$), the number of Newton iterations ($NumNSteps$), the number of linear iterations ($Iter$), the average number of linear iterations per Newton iteration ($AvgIter$), the total simulation time ($Time$), and the parallel speedup ($Speedup$) for the OpenMP and GPU versions of commercial and our simulators, respectively.

It can be seen from Tab. 14 that there is more the number of time steps, Newton iterations, and linear iterations in commercial simulator. When the number of threads increases from 1 to 16, linear iterations increase dramatically (or the average number of linear iterations per Newton iteration increases), and the simulation time is reduced from 7.626 (h) to 0.590 (h). At this point, the maximum speedup reaches 12.92 and the average number of linear iterations per Newton iteration exceeds 260. In our simulator, the number of time steps, Newton iterations, and linear iterations are less. Note that the number of linear iterations is basically stable as the number of threads increases from 1 to 16. When $\mu = 30$ and the number of threads varied from 1 to 16, the simulation time was reduced from 1.288 (h) to 0.317 (h). At this point, the maximum speedup reaches 4.06 and the average number of linear iterations per Newton iteration is about 22. This indicates that the proposed method can speed up the simulation time by 1.86 times compared to commercial simulator.

According to Tab. 15, we can summarize similar conclusions. Compared with the commercial simulator, our obtained number of time steps, Newton iterations, and linear iterations is smaller, and the simulation time is shorter. Especially when $\mu = 30$, the speedup of our simulator can achieve 3.86 compared to commercial simulator.

Tab. 14

NumTSteps, *NumNSteps*, *Iter*, *AvgIter*, *Time(h)*, and *Speedup* comparisons of the OpenMP version of commercial and our simulators for the three-phase SPE10 problem.

Simulators	μ	NT	1	2	4	8	16
Commercial	—	<i>NumTSteps</i>	114	113	115	115	114
		<i>NumNSteps</i>	233	229	232	231	230
		<i>Iter</i>	60876	64693	67787	70595	73864
		<i>AvgIter</i>	261.3	282.5	292.2	305.6	321.1
		<i>Time</i>	7.626	4.031	2.125	1.108	0.590
		<i>Speedup</i>	1.00	1.89	3.59	6.88	12.92
Ours	0	<i>NumTSteps</i>	73	73	73	73	73
		<i>NumNSteps</i>	178	178	178	178	178
		<i>Iter</i>	3236	3236	3237	3237	3237
		<i>AvgIter</i>	18.2	18.2	18.2	18.2	18.2
		<i>Time</i>	1.144	0.715	0.482	0.371	0.345
		<i>Speedup</i>	1.00	1.60	2.37	3.08	3.32
	10	<i>NumTSteps</i>	73	73	73	73	73
		<i>NumNSteps</i>	178	178	178	178	178
		<i>Iter</i>	3253	3253	3253	3253	3253
		<i>AvgIter</i>	18.3	18.3	18.3	18.3	18.3
		<i>Time</i>	1.115	0.692	0.475	0.363	0.337
		<i>Speedup</i>	1.00	1.61	2.35	3.07	3.31
	20	<i>NumTSteps</i>	73	73	73	73	73
		<i>NumNSteps</i>	178	178	178	178	178
		<i>Iter</i>	3464	3463	3460	3461	3460
		<i>AvgIter</i>	19.5	19.5	19.4	19.4	19.4
		<i>Time</i>	1.140	0.699	0.467	0.354	0.319
		<i>Speedup</i>	1.00	1.63	2.44	3.22	3.57
	30	<i>NumTSteps</i>	74	74	74	73	73
		<i>NumNSteps</i>	178	178	178	178	178
		<i>Iter</i>	3891	3891	3890	3889	3996
		<i>AvgIter</i>	21.9	21.9	21.9	21.8	22.5
		<i>Time</i>	1.288	0.774	0.503	0.358	0.317
		<i>Speedup</i>	1.00	1.66	2.56	3.60	4.06
40	<i>NumTSteps</i>	74	74	74	74	74	
	<i>NumNSteps</i>	178	178	178	178	178	
	<i>Iter</i>	4111	4111	4118	4125	4105	
	<i>AvgIter</i>	23.1	23.1	23.1	23.2	23.1	
	<i>Time</i>	1.245	0.749	0.488	0.360	0.324	
	<i>Speedup</i>	1.00	1.66	2.55	3.46	3.84	

Tab. 15

NumTSteps, NumNSteps, Iter, AvgIter, Time(h), and Speedup (compared with the commercial simulator) comparisons of the GPU version of commercial and our simulators for the three-phase SPE10 problem.

Simulators	μ	<i>NumTSteps</i>	<i>NumNSteps</i>	<i>Iter</i>	<i>AvgIter</i>	<i>Time</i>	<i>Speedup</i>
Commercial	—	117	233	71415	306.5	1.004	—
Ours	0	74	178	3270	18.4	0.339	2.96
	10	74	178	3302	18.6	0.317	3.17
	20	76	178	3424	19.2	0.282	3.56
	30	76	178	4065	22.8	0.260	3.86
	40	74	178	4239	23.8	0.263	3.81

7. Conclusions. In this paper, we investigated an efficient parallel CPR preconditioner for the linear algebraic systems arising from the fully implicit discretization of the black oil model. First, for two difficulties of the preconditioner, the computation cost is large, and the parallel speedup is low in SETUP. We proposed an adaptive SETUP CPR preconditioner to improve the efficiency and parallel performance of the preconditioner. Furthermore, we proposed an efficient parallel GS algorithm based on the coefficient matrix of strong connections. The algorithm can be adapted to unstructured grids, yielded the same convergence behavior as the corresponding single-thread algorithm, and obtained a good parallel speedup. This paper took the CPR preconditioner as an example to illustrate our proposed methods, which can easily be extended to multi-stage preconditioners. In the future, the parallel performance of the adaptive SETUP multi-stage preconditioners needs to be further improved. This paper only considered OpenMP and CUDA implementations for the proposed parallel GS algorithm, so further research will be conducted for MPI parallelism.

Acknowledgments. This work was supported by the Postgraduate Scientific Research Innovation Project of Hunan Province (No. CX20210607) and Postgraduate Scientific Research Innovation Project of Xiangtan University (No. XDCX2021B110). Feng was partially supported by the Excellent Youth Foundation of SINOPEC (No. P20009). Zhang was partially supported by the National Science Foundation of China (No. 11971472). Shu was partially supported by the National Science Foundation of China (No. 11971414). We would like to thank Prof. Ruizhong Jiang (China University of Petroleum) for providing numerical results by tNavigator.

Code availability section.

- Name of code: faspsolver, faspcpr
- Program language : the code is written in C
- Repository: <https://github.com/FaspDevTeam/faspsolver>
- Repository: <https://github.com/zhaoli0321/faspcpr>
- Prerequisites: available at the repository
- Description: available at the repository

REFERENCES

- [1] K. Aziz and A. Settari. *Petroleum Reservoir Simulation*. Applied Science Publishers, London, 687pp, 1979.
- [2] Z.X. Chen, G.R. Huan, and Y.L. Ma. *Computational Methods for Multiphase Flows in Porous Media*. 2006.

- [3] A. Goudarzi, M. Delshad, and K. Sepehrnoori. A chemical EOR benchmark study of different reservoir simulators. *Computers & Geosciences*, 94:96–109, 2016.
- [4] D.W. Peaceman. *Fundamentals of Numerical Reservoir Simulation*. Developments in Petroleum Science, United States, 190pp, 1977.
- [5] H. Valiollahi, Z. Ziabakhsh, and P.L.J. Zitha. Mathematical modeling of chemical oil-soluble transport for water control in porous media. *Computers & Geosciences*, 45:240–249, 2012.
- [6] J.W. Sheldon, B. Zondek, and W.T. Cardwell. One-dimensional, incompressible, noncapillary, two-phase fluid flow in a porous medium. *Transactions of the AIME*, 216(1):290–296, 1959.
- [7] J. Douglas, Jr., D.W. Peaceman, and H.H. Rachford, Jr. A method for calculating multi-dimensional immiscible displacement. *Transactions of the AIME*, 216:297–308, 1959.
- [8] H.L. Stone and A.O. Garder, Jr. Analysis of gas-cap or dissolved-gas drive reservoirs. *Society of Petroleum Engineers Journal*, 1:92–104, 1961.
- [9] D.A. Collins, L.X. Nghiem, Y.K. Li, and J.E. Grabonstotter. An efficient approach to adaptive-implicit compositional simulation with an equation of state. *SPE Reservoir Engineering*, 7(02):259–264, 1992.
- [10] T.A. Davis. *Direct Methods for Sparse Linear Systems*. 2006.
- [11] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, second edition, 2003.
- [12] J.T. Camargo, J.A. White, N. Castelletto, and R.I. Borja. Preconditioners for multiphase poromechanics with strong capillarity. *International Journal for Numerical and Analytical Methods in Geomechanics*, 45(9):1141–1168, 2021.
- [13] J.C. Xu. Iterative methods by space decomposition and subspace correction. *SIAM Review*, 34(4):581–613, 1992.
- [14] J.A. Meyerink. Iterative methods for the solution of linear equations based on incomplete block factorization of the matrix. In *SPE Reservoir Simulation Symposium*, volume SPE-12262. OnePetro, 1983.
- [15] A. Brandt, S.F. McCormick, and J.W. Ruge. Algebraic multigrid (AMG) for sparse matrix equations. In *Sparsity and its Applications*, pages 257–284. Cambridge University Press, 1984.
- [16] R.D. Falgout. An introduction to algebraic multigrid computing. *Computing in Science and Engineering*, 8(6):24–33, 2006.
- [17] H. Cao, H.A. Tchelepi, J.R. Wallis, and H.E. Yardumian. Parallel scalable unstructured CPR-Type linear solver for reservoir simulation. In *SPE Annual Technical Conference and Exhibition*, volume SPE-96809. OnePetro, 2005.
- [18] Z. Li, S.H. Wu, C.S. Zhang, J.C. Xu, C.S. Feng, and X.Z. Hu. Numerical studies of a class of linear solvers for fine-scale petroleum reservoir simulation. *Computing & Visualization in Science*, 18(2-3):93–102, 2017.
- [19] J.R. Wallis. Incomplete gaussian elimination as a preconditioning for generalized conjugate gradient acceleration. In *SPE Reservoir Simulation Symposium*. OnePetro, 1983.
- [20] J.R. Wallis, R.P. Kendall, and T.E. Little. Constrained residual acceleration of conjugate residual methods. In *SPE Reservoir Simulation Symposium*. OnePetro, 1985.
- [21] T.M. Al-Shaalan, H.M. Klie, A.H. Dogru, and M.F. Wheeler. Studies of robust two stage preconditioners for the solution of fully implicit multiphase flow problems. In *SPE Reservoir Simulation Symposium*. OnePetro, 2009.
- [22] X.Z. Hu, J.C. Xu, and C.S. Zhang. Application of auxiliary space preconditioning in field-scale reservoir simulation. *Science China Mathematics*, 56(12):2737–2751, 2013.
- [23] K. Stüben, T. Clees, H. Klie, B. Lu, and M.F. Wheeler. Algebraic multigrid methods (AMG) for the efficient solution of fully implicit formulations in reservoir simulation. In *SPE Reservoir Simulation Symposium*. OnePetro, 2007.
- [24] H.M. Bücker, A.I. Kauerauf, and A. Rasch. A smooth transition from serial to parallel processing in the industrial petroleum system modeling package petromod. *Computers & Geosciences*, 34(11):1473–1479, 2008.
- [25] A.H. Dogru, L.S.K. Fung, U. Middy, T. Al-Shaalan, and J.A. Pita. A next-generation parallel reservoir simulator for giant reservoirs. In *SPE Reservoir Simulation Symposium*, volume SPE-119272. OnePetro, 2009.
- [26] C.S. Feng, S. Shu, J.C. Xu, and C.S. Zhang. A multi-stage preconditioner for the black oil model and its OpenMP implementation. *Lecture Notes in Computational Science and Engineering*, 98:141–153, 2014.
- [27] M. Mesbah, A. Vatani, M. Siavashi, and M.H. Doranegard. Parallel processing of numerical simulation of two-phase flow in fractured reservoirs considering the effect of natural flow barriers using the streamline simulation method. *International Journal of Heat and Mass Transfer*, 131:574–583, 2019.
- [28] H. Sudan, H. Klie, R. Li, and Y. Saad. High performance manycore solvers for reservoir simulation. *European Conference on the Mathematics of Oil Recovery*, 2010.
- [29] X.H. Wei, W.S. Li, H.L. Tian, H.L. Li, H.X. Xu, and T.F. Xu. THC-MP: High performance numerical simulation of reactive transport and multiphase flow in porous media. *Computers & Geosciences*,

- 80:26–37, 2015.
- [30] A. Wilkins, C.P. Green, and J. Ennis-King. An open-source multiphysics simulation code for coupled problems in porous media. *Computers & Geosciences*, 154:104820, 2021.
 - [31] S.H. Wu, J.C. Xu, C.S. Feng, C.S. Zhang, Q.Y. Li, S. Shu, B.H. Wang, X.B. Li, and H. Li. A multi-level preconditioner and its shared memory implementation for a new generation reservoir simulator. *Petroleum Science*, 11:540–549, 2014.
 - [32] S.H. Wu, B.H. Wang, Q.Y. Li, J.C. Xu, C.S. Zhang, and C.S. Feng. Cost-effective parallel reservoir simulation on shared memory. In *SPE Asia Pacific Oil & Gas Conference and Exhibition*, volume SPE-182367-MS. OnePetro, 2016.
 - [33] B. Yang, H. Liu, and Z.X. Chen. Accelerating linear solvers for reservoir simulation on GPU workstations. *Society for Computer Simulation International*, 1:1–8, 2016.
 - [34] H.J. Yang, S.Y. Sun, Y.T. Li, and C. Yang. Parallel reservoir simulators for fully implicit complementarity formulation of multicomponent compressible flows. *Computer Physics Communications*, 244:2–12, 2019.
 - [35] J.A. Trangenstein and J.B. Bell. Mathematical structure of the black-oil model for petroleum reservoir simulation. *SIAM Journal on Applied Mathematics*, 49(3):749–783, 1989.
 - [36] C.S. Feng, S. Shu, J.C. Xu, and C.S. Zhang. Numerical study of geometric multigrid methods on CPU-GPU heterogeneous computers. *Advances in Applied Mathematics and Mechanics*, 6(1):1–23, 2014.
 - [37] V.E. Bulgakov. Multi-level iterative technique and aggregation concept with semi-analytical preconditioning for solving boundary-value problems. *Communications in Numerical Methods in Engineering*, 9(8):649–657, 1993.
 - [38] A. Napov and Y. Notay. An algebraic multigrid method with guaranteed convergence rate. *SIAM Journal on Scientific Computing*, 34(2):A1079–A1109, 2012.
 - [39] X.Z. Hu, P.S. Vassilevski, and J.C. Xu. Comparative convergence analysis of nonlinear AMLI-Cycle multigrid. *SIAM Journal on Numerical Analysis*, 51(2):1349–1369, 2013.
 - [40] M.A. Christie and M.J. Blunt. Tenth SPE comparative solution project: A comparison of upscaling techniques. *SPE Reservoir Evaluation & Engineering*, 4(04):308–317, 2001.
 - [41] Rock Flow Dynamics. Reservoir simulator tNavigator user manual. 2020.