A Multi-Stage Preconditioner for the Black Oil Model and Its OpenMP Implementation

Chunsheng Feng[1], Shi Shu[2], Jinchao Xu[3], and Chen-Song Zhang[4]

1 Introduction

A significant portion of our energy needs is met using oil and gas, and mathematical models of flow through porous media play an important role in developing and managing oil and gas reservoirs. Highly sophisticated mathematical and computational methods that describe compressible multi-phase multi-component fluid flow in reservoirs are crucial for optimizing oil reservoir development. Numerical solutions of these highly nonlinear coupled partial differential equations (PDEs) require moderate to sophisticated algorithms and computing platforms.

When a reservoir's pressure drops below bubble-point pressure, the hydrocarbon phase splits into a liquid (oil) phase and a gaseous (gas) phase at the thermodynamical equilibrium. Under these conditions, the flow in the porous media is of the black oil type: the water phase does not exchange mass with the other phases, and the liquid and gaseous phases exchange mass with each other. This model is referred to as the black oil model and is often applied in primary and secondary oil recovery. In this paper, we will consider a numerical solution of the black oil model, although the methods discussed here can be extended to other models.

We propose an algorithm for solving the Jacobian system $Ax = b$ arising from the fully implicit method, which is the most popular method for the black oil model (see [8]). The proposed method constructs an efficient preconditioner using the framework in [14]. We will focus on the multithread implementation of this method in modern multicore computer environments. In order to facilitate the discussion and emphasize the main points, we will use a simplified version of the algorithm.

Obtaining a solution of a large-scale reservoir simulation is challenging. The Jacobian system resulting from the Newton linearization is usually large, sparse, highly nonsymmetric, and ill-conditioned. However, the Krylov subspace methods, such as BiCGstab and GMRes, are efficient iterative methods for these linear systems (see [21]). In order to solve a linear algebraic system of equations efficiently, a preconditioner is often necessary to accelerate a Krylov subspace method. A preconditioner is an approximation to A^{-1}, and its action on a vector should be easy to compute. The preconditioners used in reservoir simulators mainly fall into two

School of Mathematics and Computational Science, Xiangtan University, China. Email: spring@xtu.edu.cn · Hunan Key Laboratory for Computation and Simulation in Science and Engineering, Xiangtan University, China. Email: shushi@xtu.edu.cn · Department of Mathematics, The Pennsylvania State University, USA. Email: xu@math.psu.edu · NCMIS & LSEC, Academy of Mathematics and System Sciences, China. Email: zhangcs@lsec.cc.ac.cn.

categories: (i) purely algebraic preconditioners and (ii) preconditioners based on the different properties of the variables.

Category (i) includes block incomplete lower-upper factorization (BILU) methods [17, 10], nest factorization [3, 4], and SVD-reduction methods [24]. Category (ii), on the other hand, includes methods based on the understanding that pressure variables and saturation variables differ from each other in regard to analytic properties; representative examples are the combinative method [6], the constrained pressure residual (CPR) method [23], and several multi-stage methods [2, 16, 15, 22]. As a key component of these preconditioners, algebraic multigrid (AMG) methods [7, 20, 12] have also been applied.

There is a trend toward using multicore processors, which helps CPU designers to avoid the high power-consumption problem that comes with increasing chip frequency. As CPU speeds rise into the 3–4 GHz range, the amount of electrical power required is prohibitive. Hence, the trend toward multicore processors started and will continue into the foreseeable future. OpenMP is an application program interface that can be used to explicitly direct multicore (shared memory) parallelism. It is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify shared memory parallelism in Fortran and C/C++ programs.

Several difficulties can arise when using multithread implementation for preconditioned Krylov subspace methods: (i) Some preconditioners use sequential algorithms, like Gauss-Seidel; (ii) OpenMP programs sometimes require more memory space than their corresponding sequential versions do. When a numerical algorithm is implemented in OpenMP or any other multithread computer language, it is important to maintain the convergence rate of the corresponding sequential algorithm. However, this is not always possible as many numerical algorithms are sequential in nature. When working with sparse matrices in compressed formats, like the Compressed Sparse Row format, we sometimes need to introduce auxiliary memory space. This becomes an increasingly heavy burden as the number of threads increases. We will analyze the parallel interpolation and coarse-grid operators in the setup phase of AMG based on the fact that the coefficient matrices A we consider are banded. Our results will offer a basis for reducing memory costs.

The rest of the paper is organized as follows: In Section 2, we describe the widely used black oil model and its fully implicit discretization. In Section 3, we introduce a simplified version of the preconditioner studied in [14] for the black oil model and show how this method relates to a few well-known methods such as the CPR method. In Section 4, we give the implementation details of the proposed preconditioner in the shared-memory architecture using OpenMP. Finally, in Section 5, we report the results of some numerical experiments conducted in a typical multicore computing environment.

2 The black oil model

The black oil model is developed based on the assumptions that (i) the reservoir is isothermal, (ii) the flow in porous media has three phases (liquid, gaseous, water) and three components (oil, gas, water), (iii) mass transfer occurs between the oil and gas phase, and (iv) no mass transfer occurs between the water phase and either the gas or the oil phases. We use lower- and upper-case subscripts to indicate three phases—water, oil (the liquid phase), and gas (the gaseous phase)—and the component of each—water, oil, and gas, respectively.

Let ϕ and k denote the porosity and permeability, respectively, of the porous medium $\Omega \subset \mathbb{R}^3$. For the α-phase ($\alpha = w, o, g$), let S_α, μ_α, p_α, u_α, B_α, ρ_α, and $k_{r\alpha}$ be the saturation, viscosity, pressure, volumetric velocity, formation volume factor (FVF), density, and relative permeability, respectively. Moreover, we use R_{so} to denote the gas solubility, and we use Q_{Ws}, Q_{Os}, and Q_{Gs}[1] to denote the volumetric production rate of water, oil, and gas, respectively. The mass conservation equations of the black oil model can be written as follows:

$$\frac{\partial}{\partial t}\left(\phi \frac{S_w}{B_w}\right) + \nabla \cdot \left(\frac{1}{B_w} u_w\right) = \frac{Q_{Ws}}{B_w}, \tag{1}$$

$$\frac{\partial}{\partial t}\left(\phi \frac{S_o}{B_o}\right) + \nabla \cdot \left(\frac{1}{B_o} u_o\right) = \frac{Q_{Os}}{B_o}, \tag{2}$$

$$\frac{\partial}{\partial t}\left[\phi \left(\frac{S_g}{B_g} + \frac{R_{so} S_o}{B_o}\right)\right] + \nabla \cdot \left(\frac{1}{B_g} u_g + \frac{R_{so}}{B_o} u_o\right) = \frac{Q_{Gs}}{B_g} + \frac{R_{so} Q_{Os}}{B_o}, \tag{3}$$

where

$$u_\alpha = -\frac{k k_{r\alpha}}{\mu_\alpha}\left(\nabla P_\alpha - \rho_\alpha \mathrm{g} \nabla z\right), \qquad \alpha = w, o, g \tag{4}$$

$$S_w + S_o + S_g = 1. \tag{5}$$

Equations (1)–(3) describe the mass conservation of the water, oil, and gas components, respectively; (4) is the Darcy's law for porous media; and (5) represents the phase saturation balance. Throughout this paper, we assume that the capillary pressure between each phase is zero, i.e., $P_w = P_o = P_g = P$.

Among the many possible discretization methods for the above model, we consider only the Fully Implicit method (FIM) [11] in which the Newton linearization is combined with first-order upstream-weighting finite difference spatial discretization; for details, see [8, Chapter 8]. For the sake of simplicity and clarity, we make two more assumptions:

- All three phases are present during the whole simulation period of the black oil model; i.e., the transition between the two-phase and the three-phase regions is ignored.

[1] The subscript s indicates that these variables are at the standard conditions instead of reservoir conditions.

- The well flow rate constraints are modeled by the Peaceman model (see [19]), and they are treated explicitly; i.e., the well constraints do not contribute to the Jacobian system.

Remark 1 (Phase transition and implicit wells). We note that these two assumptions are made only so that we can the main ideas of the method as clearly as possible. In practical implementation, none of these assumptions is applicable: (i) When only two phases are present in a reservoir grid-cell, we add another primary variable—the gas solubility R_{so} or the bubble-point pressure P_b—besides oil pressure and saturation as many other simulators do. (ii) Treating well constraints implicitly is important to obtain accurate simulation results in a more stable fashion. When implicit well constraints are present, we get a bordered coefficient matrix; details on how to treat them can be found in [14].

We eliminate S_g from (1)–(4) using (5) and plug (4) into (1)–(3). Moreover, we choose the increments δP, δS_w, and δS_o as the main solution variables[2] and give the rest of the variables in terms of these main solution variables. In each Newton iteration, this discretization method gives a Jacobian system of the following type:

$$A = \begin{bmatrix} A_{1P} & A_{1S_w} & \\ A_{2P} & A_{2S_w} & A_{2S_o} \\ A_{3P} & A_{3S_w} & A_{3S_o} \end{bmatrix}, \tag{6}$$

where A_{1P} is the pressure block of the water mass conservation equation; the block matrix

$$\begin{bmatrix} A_{2S_w} & A_{2S_o} \\ A_{3S_w} & A_{3S_o} \end{bmatrix}$$

is the saturation block; and A_{1S_w}, A_{2P}, and A_{3P} are the blocks that couple the pressure with the non-pressure variables.

The coefficient matrix A of the Jacobian system is often large and sparse, and it is stored in the block compressed sparse row (BCSR)[3] format. From this point on, N_P is used to refer to the total number of pressure unknowns and N_{S_w} and N_{S_o} are the numbers of the water and oil saturation unknowns, respectively. We further define $N_S = N_{S_w} + N_{S_o}$ and $N = N_P + N_S$.

Remark 2 (Decoupling strategies). The decoupling technique is a preprocessing step designed to weaken the coupling between different unknowns. There are many possible options for decoupling, such as Householder transformations, the IMPES-type method, and the BSD method based on the least square method. Details regarding the performance of each and a comparison between them can be found in [2, 16], for example. For the present study, we apply the alternative block factorization (ABF) strategy introduced by Bank et al. [5] due to its simplicity and reasonable decoupling effects. Investigating efficient and robust decoupling strategies is beyond the scope of this paper.

[2] We denote the solution variable as $x := [\delta P, \delta S_w, \delta S_o]^T$.

[3] This data structure is similar to the compressed sparse row (CSR) format, but each nonzero entry is a 3×3 sub-matrix in BCSR.

3 A multi-stage preconditioner for FIM

It is natural to introduce auxiliary or fictitious problems for different physical unknowns and use them to construct a multi-stage (multiplicative) preconditioner. Assume that we have the transfer operators Π_P and Π_S from x to the pressure variable P and the saturations, respectively. Let R be a relaxation or smoother for A. A multi-stage preconditioner can be defined in Algorithm 1.

Algorithm 1: A multiplicative preconditioner for the black oil model

```
Step 0. Given an initial guess $x$
Step 1. $x \leftarrow x + \Pi_S B_S \Pi_S^T (b - Ax)$
Step 2. $x \leftarrow x + \Pi_P B_P \Pi_P^T (b - Ax)$
Step 3. $x \leftarrow x + R(b - Ax)$
```

It is easy to see that this algorithm defines a preconditioner B such that

$$I - BA = (I - RA)(I - \Pi_P B_P \Pi_P^T A)(I - \Pi_S B_S \Pi_S^T A). \tag{7}$$

The choice of auxiliary problems and their corresponding solvers is crucial to the overall performance of the preconditioner B. The auxiliary problems should preserve the property of the governing equations of each unknown. We expect A_{1P} to preserve the ellipticity of the pressure equation, and we expect multilevel solvers like AMG to solve this auxiliary problem efficiently.

To facilitate our discussion on OpenMP implementation in the next section, we will use a simple version of Algorithm 1, in which we define

$$\Pi_P = \begin{bmatrix} I_P \\ 0 \end{bmatrix} \in \mathbb{R}^{N \times N_P} \quad \text{and} \quad \Pi_S = \begin{bmatrix} 0 \\ I_S \end{bmatrix} \in \mathbb{R}^{N \times N_S},$$

where $I_P \in \mathbb{R}^{N_P \times N_P}$ and $I_S \in \mathbb{R}^{N_S \times N_S}$ are identity matrices corresponding to the pressure variables and the saturation variables, respectively. We use one classical AMG V-cycle [20] as the subspace solver B_P, and we apply the block Gauss-Seidel (GS) method as the subspace solver B_S and the relaxation R. For the multithreaded version, the usual GS method is replaced by the hybrid GS method.[4] This preconditioner is referred to as B_{MSP} in the rest of this paper.

Remark 3 (CPR preconditioner). One well-known special case of Algorithm 1 is the constrained pressure residual (CPR) preconditioner [23], which can be presented in the following algebraic form:

$$B_{\text{CPR}} = R(I - AM) + M, \tag{8}$$

where

[4] The standard GS sweep is applied in each thread, and parallel (simultaneous) updating is used across multiple threads.

$$M = \begin{bmatrix} B_P & 0 \\ 0 & 0 \end{bmatrix} \in \mathbb{R}^{N\times N} \quad \text{and} \quad B_P \approx A_{1P}^{-1} \text{ is constructed using AMG.} \tag{9}$$

The smoother R is usually defined by the Line SOR smoother or the Incomplete Factorization methods. B_P can often be replaced by one or more AMG cycles. If we choose $\Pi_P = [I_P, 0, 0]^T$, then we can rewrite the CPR preconditioner as

$$I - B_{\text{CPR}}A = (I - RA)\Big(I - \Pi_P B_P \Pi_P^T A\Big), \tag{10}$$

which has the exact same form of (7) as for A.

Remark 4 (Block triangular preconditioner). Another simple way to construct an efficient preconditioner is to choose $R = 0$ in (7). In this case, the resulting preconditioner B_{TRIG} can be viewed as a block upper triangular preconditioned with B_P as an approximated A_{1P}^{-1} and B_S as an approximation of $[A_{2S_w}, A_{2S_o}; A_{3S_w}, A_{3S_o}]^{-1}$. The preconditioner, therefore, is an inexact version of the block GS method.

4 Implementation details in OpenMP

In this section, we discuss an OpenMP implementation of the proposed auxiliary space preconditioner in Algorithm 1. Using a shared-memory paradigm can greatly simplify the programming task compared to message-passing implementations. OpenMP parallel programs are relatively easy to implement, as each processor has a global view of the entire memory. Parallelism can be achieved by inserting compiler directives into the code to distribute loop iterations among the processors. However, performance may suffer from the poor spatial locality of physically distributed shared data [18].

In this paper, we will not discuss general tasks such as sparse-matrix multiplications for OpenMP. Interested readers are referred to Oliker et al. [18] and references therein for related discussions. We will focus on one part of our algorithm, namely the setup stage of the classical AMG method and propose a simple but efficient algorithm for constructing standard prolongation and coarse-level operators using OpenMP. We show that if the bandwidth of the sparse coefficient matrix A is relatively small, then much less memory is needed.

Suppose $A \in \mathbb{R}^{n\times n}$ is symmetric. Let $G_A(V,E)$ denote the graph of the matrix A where V is the set of vertices (i.e., unknowns), and let E be the set of edges (i.e., connections that correspond to nonzero matrix entries). Suppose the index set of vertices V is split into a set C of coarse-level vertices and a set F of fine-level vertices, such that

$$V = C \cup F \quad \text{and} \quad C \cap F = \varnothing,$$

and we denote n_c as the cardinality of C, i.e., the number of C-vertices. Assume that F^C is the map from F-vertices to C-vertices.

We define $N_i := \{j \in V : \ A_{ij} \neq 0, \ j \neq i\}$, and for $\theta \in [0,1)$ we denote

$$S_i(\theta) := \left\{ j \in N_i : \ -A_{ij} \geq \theta \cdot \max_{k \neq i}(-A_{ik}) \right\}.$$

Let $D_i^{F,s} := S_i(\theta) \cap F$, $D_i^{C,s} := S_i(\theta) \cap C$ and $D_i^w := N_i \setminus \left(D_i^{C,s} \cup D_i^{F,s}\right)$. We can now define

$$F_i := \left\{ j \in D_i^{F,s} : \ i \text{ and } j \text{ without the same depended } C\text{-vertices} \right\}.$$

Let $\hat{A}_{ij} := 0$ if $A_{ii}A_{ij} > 0$, and let $\hat{A}_{ij} := A_{ij}$ otherwise. We denote $P = (P_{ij_c}) \in \mathbb{R}^{n \times n_c}$ as the standard prolongation matrix where entry

$$P_{ij_c} = \begin{cases} \frac{-1}{A_{ii} + \sum\limits_{k \in D_i^w \cup F_i} A_{ik}} \left(A_{ij} + \sum\limits_{k \in D_i^{F,s} \setminus F_i} \frac{A_{ik}\hat{A}_{kj}}{\sum\limits_{m \in D_i^{C,s}} \hat{A}_{km}} \right), & i \in F, \, j \in D_i^{C,s}, \, j_c = F^C[j], \\ 1.0, & i \in C, \ j_c = F^C[i], \\ 0.0, & \text{otherwise.} \end{cases}$$

As the matrix P is sparse and stored in the CSR format, we need to use an auxiliary integer marker called M_P to quickly locate the column index of each non-zero entry (see for example in BoomerAMG of hypre [1]). In fact, to generate the i-th row of P, we define that, for $0 \leq j \leq n-1$,

$$M_P[j] := \begin{cases} J_{j_c}, & j \in D_i^{C,s}, \ j_c = F^C[j], \\ -2-i, & j \in D_i^{F,s} \setminus F_i, \\ -1, & \text{otherwise,} \end{cases} \tag{11}$$

where J_{j_c} is the position of P_{ij_c} entry in the column index array of the CSR storage of P. In the OpenMP implementation, we have to allocate the marker M_P for all OpenMP threads. The length of each M_P is n, and the total length of M_P for all threads is then $N_T \times n$ where N_T is the total number of OpenMP threads. When N_T is large, the memory cost for M_P is considerable.

Assume that $b_n = b_l + b_r$ is the bandwidth of A, where b_l and b_r are the left and right bandwidths for matrix A, respectively. When the parallel partition of V is continuously distributed in a balanced fashion to each OpenMP thread (i.e., the size difference between each thread does not exceed one), we can easily see that the length of M_P that is actually used is much smaller than n (Fig. 1). Taking into account that the matrix is banded, we can get the following estimates of the length L_P^t and the minimal offset $M_l^t(P)$[13]:

$$L_P^t \leq \min(n, \frac{n}{N_T} + 2b_n) \quad \text{and} \quad M_l^t(P) \geq \max\left(0, \frac{n}{N_T}(t-1) - 2b_n\right). \tag{12}$$

The coarse grid operator for the multigrid method can be built using the Galerkin relation $A_c = (A_{ij}^c)_{n_c \times n_c} := P^T A P$, where

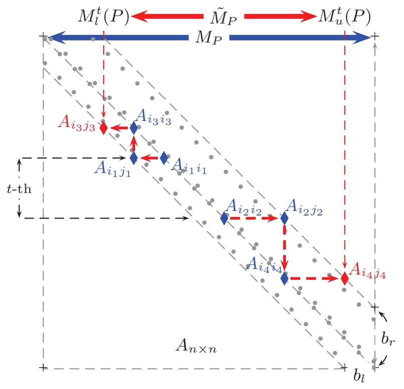

Fig. 1 Construction of the prolongation operator P for the banded sparse matrix A. Here, $M_l^t(P)$ and $M_u^t(P)$ are the lower and upper column indices, respectively, of the non-zero entries in A of the t-th OpenMP thread.

$$A_{ij}^c = \sum_{k_1}\sum_{l_1} P_{k_1 i} A_{k_1 l_1} P_{l_1 j}, \qquad i, j = 1, \cdots, n_c. \tag{13}$$

Similar to the implementation of the prolongation operator, we need to allocate two

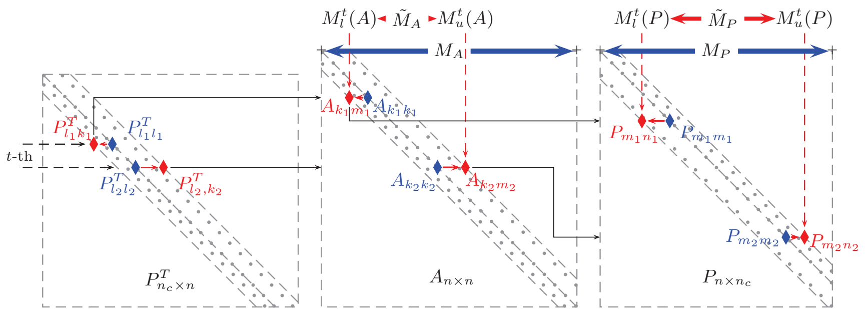

Fig. 2 Construction of the Galerkin coarse-level operator $A_c = P^T AP$. Here, $M_l^t(A)$ and $M_u^t(A)$ are the lower and upper column indices of the non-zero entries in A of the t-th OpenMP thread.

auxiliary arrays called M_A and M_P (Fig. 2). The length of M_A is n and the length of M_P is n_c. By taking into account the characteristic of the banded sparse matrices of the coarse operator, we can get the estimation formula for M_A and M_P. The actual length L_A^t and the offset $M_l^t(A)$ can be calculated using

$$L_A^t \le \min(n, \frac{n}{N_T} + 2b_n) \quad \text{and} \quad M_l^t(A) \ge \frac{n}{N_T} t - b_n. \tag{14}$$

Remark 5 (How much memory can we save?). If we do not consider the possibility that the bandwidth of A can be much smaller than n, then we will need two auxiliary arrays with length nN_T. However, as noted above, we only need two arrays of length $n+2b_nN_T$. When $n \gg b_n$ and N_T is relatively large, we can save a lot of memory by using these improved estimates. In fact, this will reduce not only storage cost but also the time needed to allocate and initialize memory.

5 Numerical experiments

In this section, we design several numerical experiments and analyze the performance of OpenMP implementation of the preconditioner proposed in Section 3. We use a HP desktop PC equipped with two Intel Xeon X5676 (3.07GHz, 12 cores) and 96GB RAM. The experimental environment is Cent OS 6.2 and GCC 4.4.6 (with an "–O2" optimization parameter).

Our example is adapted from the second data set of the Tenth SPE Comparative Solution Project ([9]), which is designed to compare the ability of upscaling approaches used by various participants to predict the performance of water-flooding in a simple but highly heterogeneous black oil reservoir described by a fine-scale $(60 \times 220 \times 85)$ regular Cartesian geological model. This model has a simple geometry, with no top structure or faults. The model dimensions are $1200 \times 2200 \times 170$ (ft). The top 70 ft (35 layers) represents the Tarbert formation, and the bottom 100 ft (50 layers) represents the Upper Ness formation. There is one injector in the center of the field and a producer located at each of the four corners. The total simulation time is 2,000 days. The purpose of this benchmark is to compare the models in regard to accuracy and computational cost.

For our purpose, we modify the SPE10 example as a three-phase black oil test by changing the properties of the fluid. Hence, the total number of unknowns of each Jacobian system is $N = 3.3$M and the size of the pressure equation is $n = N_P = 1.1$M. We employ the GMRes method as our iterative solver for solving linear Jacobian systems. The stopping criteria is that the relative residual in the Euclidian norm is less than 10^{-4}. In Table 1, we summarize the performance of our simulator, in which #Timesteps is the total number of time steps, #Newton is the total number of Newton iterations, #Linear is the total number of linear iterations, Solver Time is the total wall-time for the linear solution steps, Aver. Newton is the average number of Newton iterations in each time step, and Aver. Linear Iter is the average number of linear iterations in each Newton iteration.

Table 1 Performance of preconditioned GMRES for solving the three-phase SPE10 problem.

Preconditioner	#Timesteps	#Newton	#Linear	Solver Time (hour)	Aver. Newton	Aver. Linear Iter
B_{MSP}	736	997	32829	6.60	1.35	32.92
B_{CPR}	796	1253	57723	20.15	1.57	41.50
B_{TRIG}	805	2045	103249	17.47	2.54	46.34

In order to further demonstrate the performance of the proposed preconditioner, we select four typical Jacobian linear systems from different periods of the 2,000 days of simulation. They are all from the first Newton iteration in different time levels and the time step sizes are the same (each is five days). Using these examples, we test the performance of the three different preconditioners, B_{MSP}, B_{CPR}, and B_{TRIG}, given in Section 3. The proposed preconditioner in Algorithm 1 results in various preconditioners depending on the different choices of auxiliary problem solvers/smoothers. In this section, we only compare the performance of these three simple choices.

The total number of iterations and the wall-time in seconds for each of these methods is reported in Tables 2–4, in which N_T is the total number of OpenMP threads. Moreover, the respective OpenMP speedup for these methods are listed along with the wall-times. We observe that these three methods are very robust for the test problems and that their OpenMP versions can deliver about three times speedup compared with the corresponding serial versions. Furthermore, the numerical tests show that each component, B_S, B_P, and R, plays a role such that dropping any of them would result in at least 20% to 30% performance lost in CPU time. And, for more difficult problems, this drop is expected to be more severe.

Table 2 Number of iterations, wall-times (seconds), and OpenMP speedups of B_{MSP}.

	1st			2nd			3nd			4nd		
N_T	#Iter	Time	Speedup	#Iter	Time	Speedup	#Iter	Time	Speedup	#Iter	Time	Speedup
1	32	31.34	—	34	32.79	—	34	32.77	—	32	31.49	—
2	32	17.72	1.77	34	18.48	1.77	34	18.46	1.78	32	17.68	1.78
4	32	13.44	2.33	34	13.19	2.49	34	13.14	2.49	32	12.60	2.50
8	33	11.02	2.84	34	11.20	2.93	34	11.18	2.93	32	10.80	2.91
12	33	10.99	2.85	34	11.27	2.91	34	10.84	3.02	32	10.77	2.92

Table 3 Number of iterations, wall-times (seconds), and OpenMP speedups of B_{CPR}.

	1st			2nd			3nd			4nd		
N_T	#Iter	Time	Speedup	#Iter	Time	Speedup	#Iter	Time	Speedup	#Iter	Time	Speedup
1	45	39.01	—	45	38.90	—	43	37.36	—	42	36.56	—
2	45	21.95	1.78	45	21.90	1.78	43	21.00	1.78	42	20.67	1.77
4	45	15.42	2.53	45	15.44	2.52	44	15.19	2.46	42	14.56	2.51
8	45	13.12	2.97	45	13.09	2.97	44	12.86	2.90	42	12.35	2.96
12	45	13.19	2.96	45	13.18	2.95	43	12.66	2.95	42	11.93	3.07

Finally, we test the memory cost for the AMG setup stage, which is crucial in constructing B_P. As discussed in Section 4, the auxiliary arrays introduced to assist in assembling the sparse matrix could waste a lot of precious memory resources during the AMG setup stage. And, by using the improved bounds given in (12) and (14), we are able to use much shorter auxiliary arrays than the standard implementation in [1] and this can save a lot memory, especially when the bandwidth of the

Table 4 Number of iterations, wall-times (seconds), and OpenMP speedups of B_{TRIG}.

	1st			2nd			3nd			4nd		
N_T	#Iter	Time	Speedup	#Iter	Time	Speedup	#Iter	Time	Speedup	#Iter	Time	Speedup
1	49	41.69	—	49	41.48	—	48	40.96	—	44	37.75	—
2	49	23.42	1.78	48	22.93	1.81	48	22.87	1.79	44	21.25	1.78
4	49	16.67	2.50	49	16.62	2.50	48	16.30	2.51	44	15.37	2.46
8	49	14.30	2.91	48	13.94	2.98	48	13.91	2.95	44	12.92	2.92
12	48	14.00	2.98	48	13.99	2.97	47	13.58	3.02	44	12.99	2.91

sparse matrix A is small or the number of OpenMP threads is large. Let Length(M_P) be the total length of M_P, and let Length(M_A) be the total length of M_A. We compare these two auxiliary arrays (M_A and M_P) on the finest level as an example in Table 5. Numerical results show that this simple improvement can save about 87% storage when 12 threads are used on the finest level.

Table 5 Auxiliary memory storage on the finest level of the AMG setup for the pressure equation.

	Length(M_P)			Length(M_A)		
N_T	$N_T \times n$	L_P	Saving (%)	$N_T \times n$	L_A	Saving (%)
2	2,188,844	1,200,022	45.1	2,188,844	1,147,222	47.6
4	4,377,688	1,305,622	70.2	4,377,688	1,252,822	71.3
6	6,566,532	1,411,222	78.5	6,566,532	1,358,422	79.3
8	8,755,376	1,516,822	82.7	6,566,532	1,464,022	83.3
12	13,133,064	1,728,022	86.8	13,133,064	1,675,222	87.2

Acknowledgements The authors appreciate the anonymous referee for his or her suggestions which led to a better presentation of our method. The authors would like to thank RIPED, PetroChina, for providing the modified SPE10 test. Feng is partially supported by NSFC Grant 11201398. Shu is partially supported by NSFC Grants 91130002 and 11171281 and by the Scientific Research Fund of the Hunan Provincial Education Department of China #12A138. Xu is partially supported by NSFC Grant 91130011. Zhang is partially supported by the Dean's Startup Fund, Academy of Mathematics and System Sciences and by NSFC Grant 91130011.

References

1. hypre: A scalable linear solver library. URL `https://computation.llnl.gov/casc/linear_solvers/sls_hypre.html`
2. Al-Shaalan, T., Klie, H., Dogru, A., Wheeler, M.: Studies of Robust Two Stage Preconditioners for the Solution of Fully Implicit Multiphase Flow Problems. In: SPE Reservoir Simulation Symposium (2009)
3. Appleyard, J., Cheshire, I.: Nested factorization. In: SPE Reservoir Simulation Symposium (1983)
4. Appleyard, J., Cheshire, I., Pollard, R.: Special techniques for fully implicit simulators. In: Proceedings of the European Symposium on Enhanced Oil Recovery, Bournemouth, England, pp. 395–408 (1981)

5. Bank, R.E., Chan, T.F., Coughran Jr., W.M., Smith, R.K.: The alternate-block-factorization procedure for systems of partial differential equations. BIT **29**(4), 938–954 (1989)
6. Behie, A., Vinsome, P.: Block iterative methods for fully implicit reservoir simulation. Old SPE Journal **22**(5), 658–668 (1982)
7. Brandt, A., McCormick, S., Ruge, J.: Algebraic multigrid (AMG) for sparse matrix equations. In: Sparsity and its applications (Loughborough, 1983), pp. 257–284. Cambridge Univ. Press, Cambridge (1985)
8. Chen, Z., Huan, G., Ma, Y.: Computational methods for multiphase flows in porous media, vol. 2. Society for Industrial Mathematics (2006)
9. Christie, M., Blunt, M.: Tenth SPE Comparative Solution Project: A Comparison of Upscaling Techniques. SPE Reservoir Evaluation & Engineering **4**(4), 308–317 (2001)
10. Concus, P., Golub, G., Meurant, G.: Block preconditioning for the conjugate gradient method. SIAM J. Sci. Statist. Comput **6**(1) (1985)
11. Douglas, Jr., J., Peaceman, D.W., Rachford, D.: A method for calculating multi-dimensional displacement. Transaction of American Institute of Mining, Metallurgical, and Petroleum Engineers **216**, 297–306 (1959)
12. Falgout, R.: An introduction to algebraic multigrid. Computing in Science and Engineering **8**(6), 24 (2006)
13. Feng, C., Shu, S., Yue, X.: An Improvement for the OpenMP Version BoomerAMG. In: Proceedings of CCF HPC CHINA 2012, Zhangjiajie, China, pp. 321–328 (2012)
14. Hu, X., Liu, W., Qin, G., Xu, J., Yan, Y., Zhang, C.: Development of a fast auxiliary subspace pre-conditioner for numerical reservoir simulators. In: SPE Reservoir Characterization and Simulation Conference (2011)
15. 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 (electronic) (2003)
16. Lacroix, S., Vassilevski, Y.V., Wheeler, M.F.: Decoupling preconditioners in the implicit parallel accurate reservoir simulator (IPARS). Numer. Linear Algebra Appl. **8**(8), 537–549 (2001). Solution methods for large-scale non-linear problems (Pleasanton, CA, 2000)
17. Meyerink, J.: Iterative methods for the solution of linear equations based on incomplete block factorization of the matrix. In: SPE Reservoir Simulation Symposium (1983)
18. Oliker, L., Li, X., Husbands, P., Biswas, R.: Effects of Ordering Strategies and Programming Paradigms on Sparse Matrix Computations. SIAM Review **44**(3), 373–393 (2002)
19. Peaceman, D.W.: Presentation of a horizontal well in numerical reservoir simulation. In: The 11th SPE Symposium on Reservoir Simulation, SPE 21217 (1991)
20. Ruge, J.W., Stüben, K.: Algebraic multigrid. In: Multigrid methods, *Frontiers Appl. Math.*, vol. 3, pp. 73–130. SIAM, Philadelphia, PA (1987)
21. Saad, Y.: Iterative methods for sparse linear systems, second edn. Society for Industrial and Applied Mathematics, Philadelphia, PA (2003)
22. 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. In: SPE Reservoir Simulation Symposium (2007)
23. Wallis, J.: Incomplete gaussian elimination as a preconditioning for generalized conjugate gradient acceleration. In: SPE Reservoir Simulation Symposium (1983)
24. Watts, J., Shaw, J.: A new method for solving the implicit reservoir simulation matrix equation. In: SPE Reservoir Simulation Symposium, 31 January-2 Feburary 2005, The Woodlands, Texas (2005)