J. KSIAM Vol.22, No.3, 155–162, 2018 http://dx.doi.org/10.12941/jksiam.2018.22.155

A PARALLEL IMPLEMENTATION OF A RELAXED HSS PRECONDITIONER FOR SADDLE POINT PROBLEMS FROM THE NAVIER-STOKES EQUATIONS

HO-JONG JANG AND KIHANG YOUN†

DEPARTMENT OF MATHEMATICS, RESEARCH INSTITUTE FOR NATURAL SCIENCES, HANYANG UNIVERSITY, SEOUL 04763, KOREA

E-mail address: khyoun@hanyang.ac.kr

ABSTRACT. We describe a parallel implementation of a relaxed Hermitian and skew-Hermitian splitting preconditioner for the numerical solution of saddle point problems arising from the steady incompressible Navier-Stokes equations. The equations are linearized by the Picard iteration and discretized with the finite element and finite difference schemes on two-dimensional and three-dimensional domains. We report strong scalability results for up to 32 cores.

1. INTRODUCTION

We consider the parallel solution of the saddle point problems arising from the discretization and linearization of the incompressible Navier-Stokes equations. Recall that the steady-state Navier-Stokes equation is

$$\begin{aligned} -\nu\Delta\mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p &= \mathbf{f}, \\ \nabla\cdot\mathbf{u} &= 0, \\ \mathbf{u} &= \mathbf{g}. \end{aligned}$$

The problem is posed on an open and bounded domain Ω in $\mathbb{R}^d$ $(d = 2, 3)$, with suitable boundary conditions specified on $\partial\Omega$. Here $\mathbf{u}$ denotes the fluid velocity vector field and p the pressure scalar field, $\mathbf{f}$ is a known forcing term, $\mathbf{g}$ is some type of boundary operator (e.g. Dirichlet boundary data), and $\nu > 0$ represents viscosity. Due to the convection term $(\mathbf{u}\cdot\nabla)\mathbf{u}$, the Navier-Stokes equation is nonlinear. They can be linearized by the Picard iteration, which leads to a sequence of Oseen problems,

$$\begin{aligned} -\nu\Delta\mathbf{u} + (\mathbf{v}\cdot\nabla)\mathbf{u} + \nabla p &= \mathbf{f}, \\ \nabla\cdot\mathbf{u} &= 0, \\ \mathbf{u} &= \mathbf{g}. \end{aligned}$$

Received by the editors May 29 2018; Accepted September 10 2018; Published online September 15 2018.
2000 *Mathematics Subject Classification.* 65F10, 65N22, 65Y05, 76D07.
Key words and phrases. Saddle point problems; Preconditioning; Krylov subspace methods; Multicores.
† Corresponding author.

Here the fluid vector field $\mathbf{v}$ is the approximation of $\mathbf{u}$ from the previous the Picard iteration. Several approximation techniques can be used to discretize the Oseen problem leading to a generalized saddle point system [3] of the form

$$\begin{bmatrix} A & B^T \\ -B & C \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}, \quad \text{or} \quad \mathcal{A}x = b, \tag{1.1}$$

where now u, p, f, and g are finite-dimensional counterparts of the functions denoted by the equal symbols, $A \in \mathbb{R}^{n\times n}$ is a block diagonal matrix consisting of a set of d (for problems in d dimensions) independent discrete convection-diffusion operators, $B^T \in \mathbb{R}^{n\times m}$ is the discrete gradient, B represents a discrete divergence operator, and $C \in \mathbb{R}^{m\times m}$ is a stabilization matrix. Here we assume that a div-stable method (like MAC) is being used, so that no stabilization is required and we can take $C = 0$.

The efficient solution of (1.1) calls for rapidly convergent iterative methods. Much work has been done on the developments of efficient preconditioners for Krylov subspace methods applied to this problem; see, e.g., [1, 4, 6]. A promising technique for preconditioning the linear system (1.1) is the Hermitian and skew-Hermitian splitting(HSS) preconditioner introduced in [3], and some modifications of the HSS preconditioner including its relaxed adaptations have proposed in [7, 11].

In this paper we evaluate the potential of a relaxed HSS preconditioner within GMRES for solving large problems in a parallel environment. Our parallel implementation is largely based on public solver utilities and domain software, mostly from the PETSc [2].

The paper is organized as follows. Section 2 gives the brief description of a relaxed HSS preconditioner. The specifics of the parallel implementation are given in Section 3. Numerical results are given in Section 4. Finally, some concluding remarks are contained in Section 5.

2. THE RELAXED HSS PRECONDITIONER

2.1. **HSS iteration for the saddle point problem.** From the structure of the saddle point system (1.1), write $\mathcal{A} = \mathcal{H} + \mathcal{S}$, where

$$\mathcal{H} = \frac{1}{2}(\mathcal{A} + \mathcal{A}^T) = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}, \quad \mathcal{S} = \frac{1}{2}(\mathcal{A} - \mathcal{A}^T) = \begin{bmatrix} 0 & B^T \\ -B & 0 \end{bmatrix}$$

are the symmetric and skew-symmetric part of $\mathcal{A}$, respectively. Let $\alpha > 0$ be a parameter. Consider the following two splittings of $\mathcal{A}$:

$$\mathcal{A} = (\alpha\mathcal{I} + \mathcal{H}) - (\alpha\mathcal{I} - \mathcal{S}) \quad \text{and} \quad \mathcal{A} = (\alpha\mathcal{I} + \mathcal{S}) - (\alpha\mathcal{I} - \mathcal{H}).$$

Here $\mathcal{I}$ denotes the identity matrix, and both of the matrices $(\alpha\mathcal{I} + \mathcal{H})$ and $(\alpha\mathcal{I} + \mathcal{S})$ are nonsingular. Alternating between these two splittings leads to the stationary HSS iteration:

$$\begin{cases} (\alpha\mathcal{I} + \mathcal{H})x^{k+\frac{1}{2}} &= (\alpha\mathcal{I} - \mathcal{S})x^k + b \\ (\alpha\mathcal{I} + \mathcal{S})x^{k+1} &= (\alpha\mathcal{I} - \mathcal{H})x^{k+\frac{1}{2}} + b \end{cases}$$

($k = 0, 1, \dots$). It has been studied in [3] that for all $\alpha > 0$, the HSS iteration is convergent unconditionally to the unique solution of the saddle point problem (1.1).

2.2. **HSS preconditioner and its variants.** Benzi and Golub [3] introduced a Krylov subspace preconditioning strategy based on the symmetric/skew-symmetric splitting of the coefficient matrix $\mathcal{A}$. Let

$$\widehat{\mathcal{P}} = \frac{1}{2\alpha}(\alpha\mathcal{I} + \mathcal{H})(\alpha\mathcal{I} + \mathcal{S})$$

be a left-preconditioner for solving (1.1), then solving

$$\widehat{\mathcal{P}}^{-1}\mathcal{A}x = c,$$

where $c = \widehat{\mathcal{P}}^{-1}b$, using GMRES will always converge to the unique solution of the augmented linear system $\mathcal{A}x = b$. Since the factor $\frac{1}{2\alpha}$ has no effect on the preconditioned system, the HSS preconditioner can be written as

$$\mathcal{P}_H = \frac{1}{\alpha}(\alpha\mathcal{I} + \mathcal{H})(\alpha\mathcal{I} + \mathcal{S}) = \frac{1}{\alpha}\begin{bmatrix} A + \alpha I & 0 \\ 0 & \alpha I \end{bmatrix}\begin{bmatrix} \alpha I & B^T \\ -B & \alpha I \end{bmatrix} = \begin{bmatrix} A + \alpha I & B^T + \frac{1}{\alpha}AB^T \\ -B & \alpha I \end{bmatrix}. \tag{2.1}$$

It follows from (1.1) and (2.1) that the difference between the HSS preconditioner $\mathcal{P}_H$ and the coefficient matrix $\mathcal{A}$ is

$$\mathcal{R}_H = \mathcal{P}_H - \mathcal{A} = \begin{bmatrix} \alpha I & \frac{1}{\alpha}AB^T \\ 0 & \alpha I \end{bmatrix}. \tag{2.2}$$

We see that, as α tends to zero, the diagonal blocks in $\mathcal{R}_H$ tend to zero while the nonzero off-diagonal block becomes unbounded. Hence, it is sought an appropriate α to balance the weight of both parts. We refer readers to [7] for a detailed analysis of these preconditioners. Recently, Salkuyeh et. al [11] introduced a new relaxed version of the HSS (RH) preconditioner as

$$\mathcal{P}_{RH} = \begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix}\begin{bmatrix} I & B^T \\ -B & \alpha I \end{bmatrix} = \begin{bmatrix} A & AB^T \\ -B & \alpha I \end{bmatrix}. \tag{2.3}$$

In this case, the difference between the RH preconditioner $\mathcal{P}_{RH}$ and the coefficient matrix $\mathcal{A}$ is

$$\mathcal{R}_{RH} = \mathcal{P}_{RH} - \mathcal{A} = \begin{bmatrix} 0 & (A - I)B^T \\ 0 & \alpha I \end{bmatrix}. \tag{2.4}$$

As α tends to zero the (2, 2) block of $\mathcal{R}_{RH}$ tends to zero and in contrast with $\mathcal{R}_H$ in (2.2) the (1, 2) block remains bounded. In [11], it has been shown that the RH preconditioner is in general superior to the HSS preconditioner and its variant in [7].

Next, we consider the parallel implementation of the RH preconditioner.

3. Parallel Implementation

The parallel implementation is built upon the PETSc framework and is written in C. We demonstrate the strategy used to parallelize the solver in the RH preconditioner case. Here we briefly describe the PETSc packages and libraries we use in our code.

- Vec(Vectors). Vector operations required for setting up and solving large-scale linear and nonlinear problems. Includes easy-to-use parallel scatter and gather operations, as well as special-purpose code for handling ghost points for regular data structures.
- Mat(Operators). A large suite of data structures and code for the guidance of parallel sparse matrices. Includes four different parallel matrix data structures, each appropriate for a different class of problems.
- KSP(Krylov Subspace Methods). Parallel implementations of many popular Krylov subspace iterative methods, including GMRES, CG, CGS, BiCG-Stab, two variants of TFQMR, CR, and LSQR. All are coded so that they are immediately usable with any preconditioners and any matrix data structures, including matrix-free methods.
- PC(Preconditioners). A collection of sequential and parallel preconditioners, including (sequential) ILU(k), LU, and (both sequential and parallel) block Jacobi, overlapping additive Schwarz methods and structured MG.
- MatMPIAIJSetPreallocation. Preallocates memory for a sparse parallel matrix in AIJ format (the default parallel PETSc format). For good matrix assembly performance we should preallocate the matrix storage by setting the parameters the number of nonzeros in rows of the diagonal or off-diagonal. By setting these parameters accurately, performance can be increased by more than a factor of 50.
- VecAssembly, MatAssembly. Communicates Vector/Matrix values between different processes.
- PCFactorSetMatSolverPackage, MATSOLVERSUPERLU_DIST. A solver package providing solvers LU and ILU for sequential matrices via the external package SuperLU.

The most important works, and also the one requiring the enormous amount of effort to run efficiently, are defining the matrix-vector multiplication and preconditioner application in the RH preconditioner setting. To make efficient use of the PETSc library, the correct use of the PETSc data parallel structures for Matrices and Vectors is crucial for performance. Most of the work necessary to use PETSc was contained in constructing and populating these data structures. The PETSc library has specific data structures for sparse matrices. Since the matrix $\mathcal{A}$ is sparse, the MPIAIJ format is used to store it. In order to use this format in parallel MatMPIAIJSetPreallocation is called to create and allocate the data structure. We use PETSc calls to create a matrix object. We can call MatMPIAIJSetPreallocation to create a parallel matrix. After the matrix has been created, we call MatSetValues to insert values. Filling in the elements of a matrix or a vector in parallel takes little care: we want each element to be constructed only once, so a process needs to know which vector elements and matrix rows it owns. For this purpose, calls such as MatGetOwnershipRange. Once all of the values have been inserted with MatSetValues, we must call MatAssemblyBegin and MatAssemblyEnd to perform any needed message passing of nonlocal components in order to allow the overlap of calculation and communication.

After all this step, all vectors and matrices are partitioned as required. The C code to create a parallel matrix and vector and store saddle-point matrix and vectors is listed in Table 1.

TABLE 1

```
MatCreate(PETSC_COMM_WORLD,&A);
MatSetSizes(A,PETSC_DECIDE,PETSC_DECIDE,N,N);
MatMPIAIJSetPreallocation(A,D_nz,NULL,O_nz,NULL);
MatGetOwnershipRange(A,&Istart,&Iend);
for (i = 0 ; i < A_nnz; i++) {
   if ( Istart ≤ A_rowind[i] < Iend ){
      MatSetValues(A,1,&A_rowind[i],1,& A_colind[i],& A_val[i],ADD_VALUES);
}}
MatAssemblyBegin(A,MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A,MAT_FINAL_ASSEMBLY);
VecCreate(PETSC_COMM_WORLD,&b);
VecSetSizes(b,PETSC_DECIDE,N);
for (i = 0 ; i < b_nnz; i++) {
   if ( Istart ≤ b_rowind[i]) < Iend){
      VecSetValues(b,1,&b_rowind[i],&b_val[i],ADD_VALUES);
}}
VecAssemblyBegin(b);
VecAssemblyEnd(b);
```

To make RH preconditioner in the matrix-matrix multiplication form, we make $(\alpha\mathcal{I} + \mathcal{H})$ and $(\alpha\mathcal{I} + \mathcal{S})$ matrices using the previous routine and make RH preconditioner using parallel matrix-matrix multiplication routine MatMatMult. The C code to create the RH preconditioner is as in Table 2.

TABLE 2

```
MatCreate(PETSC_COMM_WORLD,&M_hss);
MatSetSizes(M_hss,PETSC_DECIDE,PETSC_DECIDE,N,N);
MatMatMult(M1_hss,M2_hss,MAT_INITIAL_MATRIX,PETSC_DEFAULT,&M_hss);
MatDiagonalSet(M_hss,zero,ADD_VALUES);
```

We nowhere combine GMRES methods and RH preconditioner to solve the saddle point system (1). We can implement this preconditioned GMRES solver by using KSP and PC modules within the PETSc package. The first step to make a solver for a linear system with KSP is to create a solver context with the KSPCreate, and we set the right-hand side and solution vectors by calling the routine KSPSolve. To employ a preconditioning method, we can set the preconditioner type and method with the subroutine PCSetType and PCFactorSetMatSolverPackage. During solution of preconditioned Krylov method, convergence is decided the relative decrease of the residual norm. This parameter and the maximum number of iteration steps can be set with the KSPSetTolerances. The C code to create the KSP context and to perform the solution is as in Table 3.

TABLE 3

```
KSPCreate(PETSC_COMM_WORLD,&ksp);
KSPSetType(ksp,KSPGMRES);
KSPGMRESSetRestart(ksp, 50);
KSPSetTolerances(ksp,1e-24,1e-6,PETSC_DEFAULT,500*50);
KSPGetPC(ksp,&pc);
KSPSetOperators(ksp,A,M_hss);
PCSetType(pc,PCLU);
PCFactorSetMatSolverPackage(pc,MATSOLVERSUPERLU_DIST);
PCSetUp(pc);
KSPSolve(ksp,b,x);
KSPGetResidualNorm(ksp,&rnorm);
KSPGetIterationNumber(ksp,&its);
```

4. NUMERICAL EXPERIMENTS

In this section, we present some numerical experiments on a compute cluster. The cluster consists of ten Dell FC430 nodes, and each node has a 20-core Intel 2.3 GHz E6-2650v3 Xeon processor and 64 GB of GDDR3 RAM.

The first example problem is a strong scalability and preconditioner effectiveness study for the steady 2D Oseen equations for the lid-driven cavity problem generated using the IFISS package [8]. We also show numerical experiments to demonstrate the effectiveness of the RH preconditioner $\mathcal{P}_{RH}$ for the saddle point problem (1.1) in terms of iteration counts and CPU time in seconds. The value of the viscosity is $\nu = 0.01$, and α is chosen experimentally to minimize the GMRES iteration counts. Using Q2-Q1 finite elements on the uniform 256×256 grid results in a $148,739 \times 148,739$ saddle point matrix with 16,836,552 nonzero entries. The stopping criterion for GMRES is a reduction of the initial residual norm by 6 orders of magnitude. The restarted GMRES with restarting frequency 50, i.e., GMRES(50), is applied. The initial guess is the zero vector. GMRES iteration counts and CPU times using 2, 4, 8, 16 and 32 cores are shown in Table 4. The iteration counts are nearly constant as the number of CPU cores grows, and the CPU times show a quite good scalability for up to 32 cores. As in [11], RH preconditioner $\mathcal{P}_{RH}$ is more effective than the HSS preconditioner $\mathcal{P}_H$ in terms of the iteration counts and CPU times.

Next, we present the parallel results for a 3D Oseen problem discretized by the stable Marker-and-Cell (MAC) method [10] finite difference method with viscosity $\nu = 0.01$, and $\alpha = 0.01$. Brief information of the test problems, including $n, m, nnz(A)$ and $nnz(B)$ are presented in Table 5 where $nnz(A)$ denotes the number of the nonzero elements of the matrix A. GMRES iteration counts and CPU times are shown in Table 6 when the problem is discretized on the $32 \times 32 \times 32$ and $64 \times 64 \times 64$ grids. The initial guess and stopping criterion are as in the previous example.

TABLE 4. GMRES(50) iterations and CPU times with HSS and RH Preconditioners (2D Oseen, Q2-Q1, uniform 256×256 grid, $\nu = 0.01$, $\alpha = 10^{-2}$)

Precond.		No. of cores 2	4	8	16	32
$\mathcal{P}_H$	Iterations	73	68	64	59	54
	Set-up time	11.59	9.94	8.70	10.37	10.55
	Iter time	635.33	390.48	257.61	200.06	140.40
	Total time	646.92	400.41	266.30	210.42	150.95
$\mathcal{P}_{RH}$	Iterations	3	3	3	3	3
	Set-up time	10.82	9.17	8.74	9.60	10.40
	Iter time	26.10	17.05	11.94	10.18	7.57
	Total time	36.92	26.22	20.68	19.78	17.97

TABLE 5. The size of the matrices A and B for different grids

Grid	Lid-driven cavity problem n	m	$nnz(A)$	$nnz(B)$
$32 \times 32 \times 32$	95,232	32,768	648,576	190,464
$64 \times 64 \times 64$	774,144	262,144	5,346,048	1,548,288

TABLE 6. GMRES(50) iterations and CPU times with RH Preconditioner (3D Oseen, MAC, $\nu = 0.01$, $\alpha = 10^{-2}$)

No. of cores	2	4	8	16	32
$32 \times 32 \times 32$	2	2	2	2	2
Set-up time	118.94	51.04	30.76	20.99	16.37
Iter time	54.18	34.56	20.20	17.03	10.81
Total time	173.13	85.60	50.95	38.03	27.18
$64 \times 64 \times 64$	2	2	2	2	2
Set-up time	5555.82	1904.60	1037.84	618.87	518.36
Iter time	1154.51	736.43	408.57	338.95	210.96
Total time	6710.33	2641.02	1446.42	957.82	729.32

5. CONCLUSIONS

In this paper we investigated the parallel performance of GMRES with a new relaxed version of the HSS preconditioner for the solution of linear systems arising from various discretizations of the steady Oseen equations in two-dimensional and three-dimensional. Parallelization of the code was achieved within PETSc utilities and MPICH. The results show good scalability and performance of the preconditioned Krylov subspace iteration method with respect to the problem size and the number of cores.

REFERENCES

[1] Z. -Z. Bai, G. H. Golub and M. K. Ng, *Hermitian and skew-Hermitian splitting methods for non-Hermitian positive definite linear systems*, SIAM J. Matrix Anal. Appl., **24** (2003), 603–626.

[2] S. Balay, S. Abhyankar, F. Adams, J. Brown and P. Brune et al., *PETSc Users Manual*, Argonne National Laboratory, ANL-95/11 - Revision 3.9 (2018).

[3] M. Benzi and G. H. Golub, *A preconditioner for generalized saddle point problems*, SIAM J. Matrix Anal. Appl., **26** (2004), 20–41.

[4] M. Benzi, G. H. Golub and J. Liesen, *Numerical solution of saddle point problems*, Acta Numer, **14** (2005), 1–137.

[5] M. Benzi and J. Liu, *An efficient solver for the incompressible Navier-Stokes equations in rotation form*, SIAM J. Sci. Comput., **29** (2007), 1959–1981.

[6] M. Benzi and Z. Wang, *A parallel implementation of the modified augmented Lagrangian preconditioner for the incompressible Navier-Stokes equations*, Numer Algor, **64** (2013), 73–84.

[7] Y. Cao, L. Q. Yao, M. Q. Jiang and Q. Niu, *A relaxed HSS preconditioner for saddle point problems from meshfree discretization*, J. Comput. Math, **31** (2013), 398–421.

[8] H. C. Elman, A. Ramage and D. J. Silvester, *IFISS: a Matlab toolbox for modelling incompressible flow*, ACM Trans. Math. Software, **33** (2007), Article 14.

[9] H. C. Elman, D. J. Silvester and A. J. Wathen, *Finite Elements and Fast Iterative Solvers: With Applications in Incompressible Fluid Dynamics*, Oxford University Press, Oxford, UK, 2005.

[10] F. H. Harlow, J. E. Welch, *Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface*, Phys. Fluids, **8**(1965), 2182–2189.

[11] M. H. Salkuyeh and M. Masoudi, *A new relaxed HSS preconditioner for saddle point problems*, Numer. Algor., **74** (2017), 781–795.