Received: 12 July 2022 | Revised: 5 April 2023 | Accepted: 12 May 2023

DOI: 10.1002/nla.2512

RESEARCH ARTICLE

WILEY

Convergence acceleration of preconditioned conjugate gradient solver based on error vector sampling for a sequence of linear systems

Takeshi Iwashita[1] | **Kota Ikehara**[2] | **Takeshi Fukaya**[1] | **Takeshi Mifune**[3]

[1]Information Initiative Center, Hokkaido University, Sapporo, Japan

[2]Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Japan

[3]Graduate School of Engineering, Kyoto University, Kyoto, Japan

Correspondence
Takeshi Iwashita, Information Initiative Center, Hokkaido University, N 11 W 5, Sapporo, Japan.
Email: iwashita@iic.hokudai.ac.jp

Funding information
Japan Society for the Promotion of Science, Grant/Award Numbers: JP19H04122, JP19H05662, JP20K21782, JP23H00462

Abstract

In this article, we focus on solving a sequence of linear systems that have identical (or similar) coefficient matrices. For this type of problem, we investigate subspace correction (SC) and deflation methods, which use an auxiliary matrix (subspace) to accelerate the convergence of the iterative method. In practical simulations, these acceleration methods typically work well when the range of the auxiliary matrix contains eigenspaces corresponding to small eigenvalues of the coefficient matrix. We develop a new algebraic auxiliary matrix construction method based on error vector sampling in which eigenvectors with small eigenvalues are efficiently identified in the solution process. We use the generated auxiliary matrix for convergence acceleration in the following solution step. Numerical tests confirm that both SC and deflation methods with the auxiliary matrix can accelerate the solution process of the iterative solver. Furthermore, we examine the applicability of our technique to the estimation of the condition number of the coefficient matrix. We also present the algorithm of the preconditioned conjugate gradient method with condition number estimation.

KEYWORDS

condition number estimation, conjugate gradient method, deflation, subspace correction, vector sampling

1 | INTRODUCTION

A preconditioned conjugate gradient (CG) solver is widely used to solve a linear system of equations of a symmetric positive-definite (s.p.d.) matrix that arises in various applications. The computational time to solution is mostly given by the product of the number of iterations for convergence and the computational time per iteration. High performance and parallel computing techniques are effective for reducing the computational time per iteration, and the convergence acceleration of the solver is also an important topic. The convergence rate of the CG solver is affected by the condition number or the eigenvalue distribution of the coefficient matrix. In practical simulations, the coefficient matrix often has a few small isolated eigenvalues, which lead to a significant decline in convergence. For these problems, subspace correction (SC)[1] and deflation[2] methods are widely used to improve the convergence rate of the iterative solver.

The procedures of SC and deflation involve using an auxiliary matrix to specify a certain subspace, in which errors are efficiently removed. Therefore, an appropriate setting of the auxiliary matrix (subspace) is key to making these acceleration methods work well. For example, when the range of the matrix contains the eigenspaces corresponding to small isolated eigenvalues, the convergence rate of the solver is expected to be improved by the acceleration methods. However, it is difficult to identify these eigenspaces. Accordingly, in practical simulations, an effective auxiliary matrix is often derived from information about the problem. For example, coarse grid correction in the multigrid method,[3,4] which is regarded as one of the most successful SC methods, uses the characteristics of discretized PDE problems. Other examples of the auxiliary matrix or the subspace that is determined based on physics or models can be seen in the literature.[5-9] However, there are many cases in which the eigenvector with a small eigenvalue cannot be easily identified from information about the problem. For these problems, an automatic (algebraic) auxiliary matrix construction method that does not use special knowledge of the problem has been investigated.

In this article, we introduce an algebraic auxiliary matrix construction method for a problem that involves a sequence of linear systems to be solved. When the coefficient matrices are identical, it is often called a multiple right-hand side problem. In our method, we construct an auxiliary matrix to specify the subspace using the sampling of error vectors in the preceding iterative solution process. The idea is based on the expectation that the error that is not efficiently removed in the solution process contains useful information about the eigenvectors associated with small eigenvalues.[10] Although error vector sampling during the solution process may seem difficult, we can implement it by sampling the approximate solution vectors for the targeted problem. When the solution process is complete, we can easily calculate the corresponding error vectors. We apply the Rayleigh–Ritz method using the subspace spanned by these error vectors to obtain (approximate) eigenvectors associated with small eigenvalues. In our technique, sampling plays a key role in saving the additional memory footprint and computations for SC and deflation, which is essential for many practical applications.

In this paragraph, we describe related works on algebraic auxiliary matrix construction for convergence acceleration methods. Many related works can be found in the context of recycling the Krylov subspace, deflation, the augmented Krylov subspace, subspace recycling, and spectral preconditioning. After some early activities on deflation and augmentation in a GMRES solver,[11-13] Morgan proposed the GMRES-DR method. In the method, basis vectors generated in the Arnoldi process in the restart period are used to determine the subspace used for deflation.[14] Morgan et al. also introduced some variants of the GMRES-DR method which includes an application to the flexible GMRES method.[15,16] Carpenter described five major methods to specify the subspace (enrichment vectors) in the context of solvers based on the GMRES method.[17] For CG solvers, Saad et al. introduced the deflated Lanczos algorithm and developed the deflated-CG method.[18] In this method, the vectors (subspace) used for deflation are based on A-orthogonal basis vectors and are updated in multiple linear system solution steps. Abdel-Rehim et al. introduced the deflated restarted Lanczos algorithm.[19] The techniques mentioned above were enhanced for nonlinear application problems, for example, in further research.[20,21] As a recently published study, we refer to the paper[22] in which Daas et al. introduced a method based on singular value decomposition. Moreover, we note that a block Krylov method can be used together with convergence acceleration methods, although it is a popular technique for a multiple right-hand side problem in itself.[23] Finally, we refer to a recent survey paper by Soodhalter et al.[24] The paper provides a comprehensive review of subspace recycling techniques and possibly covers most of works related to our research. For other related works that were not introduced in Reference 24, we refer to convergence acceleration techniques for AMG solvers.[25-29] These techniques intelligently identify (approximate) near kernel vectors using coarse grids and use for the convergence acceleration.

To the best of our knowledge, the above related papers do not explicitly discuss our approach, which is based on error vector sampling, especially for a preconditioned CG solver. For example, the methods introduced in References 30 and 31 focus on the GMRES method and are different from our method, though they use approximate error vectors. In this article, we describe the auxiliary matrix construction method based on vector sampling for subspace preconditioning and the deflation method. We also introduce a cost model for convergence acceleration. Finally, we report the numerical results using test matrices in various application areas that we derived from the SuiteSparse Matrix Collection,[32] although, in our preliminary analyses, we only considered two computational electromagnetic problems.[33] The numerical results confirmed the effectiveness of our method in terms of convergence (# iterations) and computational time. The numerical tests also verified our cost model and demonstrated how the small eigenvalues were captured. Furthermore, we demonstrated that our method can be used for condition number estimation without significant additional computations in the iterative solution process.

This article is structured as follows: In Section 2, we introduce the target problem in this article. In Section 3, we introduce SC preconditioning and the deflation method to accelerate the convergence of iterative solvers. In Section 4, we explain our auxiliary matrix construction method using error vector sampling. We also introduce the cost models for the convergence acceleration technique and auxiliary matrix construction. In Section 5, we discuss the numerical results. In Section 6, we provide a summary of this study.

2 | PROBLEM DEFINITION

In this article, we consider solving a sequence of n-dimensional linear systems:

$$A_k \boldsymbol{x}_k = \boldsymbol{b}_k, (k = 1, 2, \dots, k_t), \tag{1}$$

where the coefficient matrix $A_k \in \mathbb{R}^{n\times n}$ is a real s.p.d. matrix. We assume that the right-hand side vector $\boldsymbol{b}_k \in \mathbb{R}^n$ depends on the previous solution vectors. Consequently, we solve the linear systems sequentially. In this article, we discuss the case in which the coefficient matrices are all identical:

$$A_k = A, (k = 1, 2, \dots, k_t). \tag{2}$$

However, we expect that the technique introduced in the following sections will work when the coefficient matrix changes, but not dramatically. More precisely, when the coefficient matrices have identical eigenvectors associated with small eigenvalues, the technique is possibly effective. We solve the linear system of equations (1) using a preconditioned CG solver.

3 | CONVERGENCE ACCELERATION FOR ITERATIVE LINEAR SOLVERS

3.1 | Convergence acceleration methods

In an iterative linear solver, its convergence rate directly affects the solution time. We focus on convergence acceleration methods that use a (user-specified) subspace different from the subspace designated by the coefficient matrix, such as the Krylov subspace. In these methods, the dimension of the subspace used is typically much smaller than n, and the error component involved in the subspace is efficiently removed using a special procedure. A multigrid method can be regarded as a typical example of this type of convergence acceleration method. In this article, we discuss SC and deflation methods, both of which use a user-specified subspace to accelerate convergence.

3.2 | Subspace correction method

SC is a generalized version of the coarse grid correction of the multigrid method. We describe its procedure for an n-dimensional linear system; $A\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{x} \in \mathbb{R}^n$ is the unknown vector, and $\boldsymbol{b} \in \mathbb{R}^n$ is the right-hand-side vector.

In the SC method, an approximate solution vector $\tilde{\boldsymbol{x}}$ is updated as follows:

1. Step 1: Compute $\boldsymbol{f} = W^\top(\boldsymbol{b} - A\tilde{\boldsymbol{x}})$.
2. Step 2: Solve $(W^\top AW)\boldsymbol{u} = \boldsymbol{f}$.
3. Step 3: Update $\tilde{\boldsymbol{x}} \leftarrow \tilde{\boldsymbol{x}} + W\boldsymbol{u}$.

W is the auxiliary matrix used to designate the user-specified subspace. The number of columns of W is typically much less than n.

When we use the SC method together with a Krylov subspace method, we construct the preconditioner based on the correction similar to the multigrid (two-level) preconditioning.[3] SC preconditioning[1] can be combined with any other (standard) preconditioning technique in the additive/multiplicative Schwarz preconditioning manner. When the stand-alone preconditioner is denoted by M^{-1}, the additive Schwarz SC preconditioner M_{sc}^{-1} is given by

$$M_{sc}^{-1} = M^{-1} + W(W^\top AW)^{-1}W^\top. \tag{3}$$

When only subspace preconditioning is used, M is given by the identity matrix I.

3.3 | Deflation method

In this section, we describe the procedure of the deflated CG method[18] for $A\boldsymbol{x} = \boldsymbol{b}$. In the deflation method, we use the projector given by

$$P = I - W(W^{\top}AW)^{-1}(AW)^{\top}. \tag{4}$$

P decomposes n-dimensional space $\mathbb{R}^n$ into two A-orthogonal spaces $\mathcal{W}$ and $\mathcal{W}^{\perp}$. Using the projector, we can split solution vector $\boldsymbol{x}$ into two components:

$$\boldsymbol{x} = \boldsymbol{y} + \boldsymbol{z},\ \ \boldsymbol{y} = (I - P)\boldsymbol{x},\ \ \boldsymbol{z} = P\boldsymbol{x}. \tag{5}$$

In the deflation method, we calculate two vector components $\boldsymbol{y}$ and $\boldsymbol{z}$ individually. Vector $\boldsymbol{y}$ is in lower-dimensional space $range(W)$ and is given by

$$\boldsymbol{y} = (I - P)\boldsymbol{x} = W(W^{\top}AW)^{-1}W^{\top}\boldsymbol{b}. \tag{6}$$

Because $P^{\top}A(I - P) = O$ holds, we compute the second component $\boldsymbol{z}$ by solving the deflated system

$$P^{\top}A\boldsymbol{z} = P^{\top}\boldsymbol{b}. \tag{7}$$

In this article, we solve the deflated system with a semi-positive definite coefficient matrix (7) using a preconditioned CG solver. Algorithm 1 shows the algorithm of the deflated CG method. We note that projector P is not explicitly constructed in practical implementations.

Algorithm 1. Deflated PCG method

Input: $A, \boldsymbol{b}, M, W, P, \boldsymbol{x}_0, \varepsilon$

$\boldsymbol{r}_0 = P^{\top}(\boldsymbol{b} - A\boldsymbol{x}_0)$

$\boldsymbol{p}_0 = M^{-1}\boldsymbol{r}_0$

for $i = 0, 1, 2, \ldots$ **until** $\|\boldsymbol{r}_i\|_2 \le \varepsilon\|\boldsymbol{b}\|_2$ **do**

  $\alpha_i = \dfrac{(M^{-1}\boldsymbol{r}_i, \boldsymbol{r}_i)}{(\boldsymbol{p}_i, P^{\top}A\boldsymbol{p}_i)}$

  $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \alpha_i\boldsymbol{p}_i$

  $\boldsymbol{r}_{i+1} = \boldsymbol{r}_i - \alpha_i P^{\top}A\boldsymbol{p}_i$

  $\beta_i = -\dfrac{(M^{-1}\boldsymbol{r}_{i+1}, \boldsymbol{r}_{i+1})}{(M^{-1}\boldsymbol{r}_i, \boldsymbol{r}_i)}$

  $\boldsymbol{p}_{i+1} = M^{-1}\boldsymbol{r}_{i+1} + \beta_i\boldsymbol{p}_i$

end for

$\boldsymbol{x} = P\boldsymbol{x}_i + W(W^{\top}AW)^{-1}W^{\top}\boldsymbol{b}$

Output: $\boldsymbol{x}$

4 | AUXILIARY MATRIX CONSTRUCTION METHOD BASED ON ERROR VECTOR SAMPLING

4.1 | Auxiliary matrix based on eigenvectors

In SC and deflation methods, the key to convergence acceleration is the proper setting of the auxiliary matrix W. Typically, when the range of W contains eigenspaces corresponding to small eigenvalues of the coefficient matrix, the methods work. In practical problems, a coefficient matrix often has a few isolated small eigenvalues, which worsens the convergence of the iterative solver. These eigenvalues typically arise from the physical property of the targeted problem.

For a simple example, we consider the case that W is an $n \times 1$ matrix and $W = [\boldsymbol{u}_s]$, where $\boldsymbol{u}_s$ is the eigenvector associated with the smallest eigenvalue λ_s. We assume that $\lambda_s \ll 1$ and is isolated. We also assume that the coefficient matrix has an eigenvalue close to or larger than 1. In this case, SC preconditioning M_{sc}^{-1} with $M = I$ only shifts the eigenvalue λ_s to λ_s+1; that is, the preconditioned coefficient matrix has an eigenvalue of λ_s+1 and $n-1$ eigenvalues that are identical to those of A and larger than λ_s. Consequently, the condition number of the preconditioned coefficient matrix is better than that of A, which results in better convergence for the preconditioned system.

When we use the deflation method with the above setting for W, λ_s is removed in the coefficient matrix of (7), $P^\top A$. $P^\top A$ has a zero eigenvalue which is associated with $\boldsymbol{u}_s$, and other eigenvalues and eigenvectors are the same as A. The (preconditioned) CG method can be applied to (7) because $P^\top \boldsymbol{b}$ is involved in range($P^\top A$), and its convergence rate is improved from that for the original linear system, $A\boldsymbol{x} = \boldsymbol{b}$.

The above discussion is straightforwardly extended to the case that W consists of multiple eigenvectors associated with small eigenvalues. However, calculation of eigenvalues and eigenvectors typically requires more computational efforts than solving the linear system itself. Consequently, in practical simulations, the knowledge of the problem is often used for identifying the eigenvectors associated with small eigenvalues and constructing a proper auxiliary matrix. But, there are problems in which the origin of the small eigenvalue is unclear from the viewpoint of physics or simulation models. In this article, we focus on a problem of solving a sequence of linear systems, and intend to develop an automatic auxiliary matrix construction method for the problem.

4.2 | Auxiliary matrix construction method based on error vector sampling

In this section, we describe our auxiliary matrix construction method based on error vector sampling for a sequence of linear systems (1). During the first iterative solution process for $A\boldsymbol{x}_1 = \boldsymbol{b}_1$, we preserve m approximate solution vectors $\tilde{\boldsymbol{x}}_1^{(s)} (s = 1, 2, \dots, m)$. Typically, m is much smaller than n. When the solution process is complete, we obtain the solution vector $\boldsymbol{x}_1$. Consequently, we can calculate the exact error vectors that correspond to $\tilde{\boldsymbol{x}}_1^{(s)}$ using

$$\boldsymbol{e}^{(s)} = \boldsymbol{x}_1 - \tilde{\boldsymbol{x}}_1^{(s)} \ (s = 1, 2, \dots, m). \tag{8}$$

Applying the Gram–Schmidt process to these error vectors, we obtain the mutually orthogonal $\overline{m}(\leq m)$ normal basis vectors:

$$\overline{\boldsymbol{e}}^{(1)}, \ \overline{\boldsymbol{e}}^{(2)}, \ \dots, \ \overline{\boldsymbol{e}}^{(\overline{m})}. \tag{9}$$

In our technique, we use the Rayleigh–Ritz method based on the space spanned by $\overline{\boldsymbol{e}}^{(s)}$ to identify approximate eigenvectors associated with small eigenvalues of A.

The auxiliary matrix construction method is given as follows:

Step 1: Solve the $\overline{m}$-dimensional eigenvalue problem [2]:

$$E^\top A E \boldsymbol{t} = \lambda \boldsymbol{t}, \tag{10}$$

where

$$E = [\overline{\boldsymbol{e}}^{(1)} \ \overline{\boldsymbol{e}}^{(2)} \ \cdots \ \overline{\boldsymbol{e}}^{(\overline{m})}]. \tag{11}$$

Step 2: When the Ritz value λ is less than the preset threshold θ, adopt Ritz vector $E\boldsymbol{t}$ as a column vector of W. The number of Ritz values less than θ is denoted by $\tilde{m}$, and the Ritz vector that corresponds to each small Ritz value is written as $E\boldsymbol{t}_i (i = 1, 2, \dots, \tilde{m})$. Finally, the auxiliary matrix W is given by

$$W = [E\boldsymbol{t}_1 \ E\boldsymbol{t}_2 \ \cdots \ E\boldsymbol{t}_{\tilde{m}}]. \tag{12}$$

The threshold is typically much less than 1; that is, $(0 < \theta \ll 1)$ when the coefficient matrix is diagonally (or properly) scaled.

4.3 | Selection method for stored approximate solution vectors

In practical analyses, to avoid an excessive additional cost (in memory space and computations), the number of stored vectors, m, should be substantially small. We use a selection method based on "sampling." We store approximate solution vectors with a certain interval in the solution process. Considering the difficulty of predicting the number of iterations for convergence, we use the following two methods for sampling. In sampling method A, we use the algorithm shown in Appendix A. When we set m to 4 and the (preconditioned) CG solver attains convergence at the 1000th iteration, the sampling method preserves the approximate solution vectors at 256, 384, 512, and 768th iterations. The other method (sampling method B) is based on the relative residual norm. We take a sample of approximate solution vectors when the relative residual norm first reaches $10^{-s\alpha/(m+1)}$, $(s = 1, 2, \dots, m)$, when the convergence criterion is given by $10^{-\alpha}$. Based on the preliminary test results, we use sampling method A when we do not explicitly mention the sampling method.

4.4 | Computational cost for subspace correction preconditioning and deflation

In this section, we discuss the additional computational cost for two convergence acceleration techniques. We split the computational time per iteration of preconditioned CG solver T into two parts:

$$T = T_{\text{pre}} + T_{\text{cg}}, \tag{13}$$

where T_{pre} and T_{cg} are the computational time for the preconditioning and CG solver parts, respectively. Because the total data amount for matrices and vectors is typically larger than the cache memory in practical simulations, most of the computational kernels of the solver become memory bound. Consequently, we estimate the computational time using the amount of transferred data from the main memory. In the analysis, we use double precision floating point numbers for matrices and vectors. The main part of the CG solver is a sparse matrix vector multiplication (SpMV) kernel. We estimate the amount of transferred data for SpMV as $20n + 12nnz$, where nnz is the number of nonzero elements of A and the unit is byte. Although the cache hit ratio for elements of the source vector depends on the nonzero pattern of A, we use a relatively optimistic estimation. We estimate the transferred data for other parts that consist of inner products and vector updates as $56n$. When the effective memory bandwidth is denoted by b_m Byte/s, T_{cg} is estimated as

$$T_{\text{cg}} = (76n + 12nnz)/b_m. \tag{14}$$

When we use IC preconditioning, the transferred data for preconditioning is almost the same as that for SpMV. Finally, the computational time for an incomplete Cholesky CG (ICCG) iteration that is denoted by T_{iccg} is approximately given by

$$T_{\text{iccg}} = (100n + 24nnz)/b_m. \tag{15}$$

When we consider SC preconditioning, the additional cost for $W(W^\top AW)^{-1}W^\top$ should be taken into account. In the estimation, we ignore the cost for $(W^\top AW)^{-1}$ because the dimension $\tilde{m}$ is much smaller than n for the setting of $m \ll n$. The additional transferred data for the SC preconditioning is mainly for the $n \times \tilde{m}$ dense matrix W, and we estimate it as $16\tilde{m}n + 16n$. When we use SC preconditioning together with IC preconditioning, we estimate the computational time for an SC-ICCG iteration that is denoted by T_{sciccg} as

$$T_{\text{sciccg}} = (116n + 16\tilde{m}n + 24nnz)/b_m. \tag{16}$$

From (15) and (16), we can (roughly) estimate the ratio of the computational cost per iteration for two solvers, SC-ICCG and ICCG, which is denoted by γ_{sciccg}, as follows:

$$\gamma_{\text{sciccg}} = (116 + 16\tilde{m} + 24nnz_{\text{av}})/(100 + 24nnz_{\text{av}}), \tag{17}$$

where nnz_{av} is the average number of nonzero elements per row. When the number of iterations of SC-ICCG is less than $1/\gamma_{\text{sciccg}}$ that of ICCG, we expect SC-ICCG to outperform ICCG. More details of the cost models are given in Appendix B.

TABLE 1 Matrix information for the test problems.

Data set	Problem type	Dimension	# nonzero	nnz_{av}
Queen_4147	2D/3D problem	4,147,110	316,548,962	76.3
Bump_2911	2D/3D problem	2,911,419	127,729,899	43.9
G3_circuit	Circuit simulation problem	1,585,478	7,660,826	4.8
Flan_1565	Structural problem	1,564,794	114,165,372	73.0
Hook_1498	Structural problem	1,498,023	59,374,451	40.0
StocF-1465	Computational fluid dynamics problem	1,465,137	21,005,389	14.3
Geo_1438	Structural problem	1,437,960	60,236,322	41.9
Serena	Structural problem	1,391,349	64,131,971	46.1
thermal2	Thermal problem	1,228,045	8,580,313	7.0
ecology2	2D/3D problem	999,999	4,995,991	5.0
bone010	Model reduction problem	986,703	47,851,783	48.5
ldoor	Structural problem	952,203	42,493,817	44.6
audikw_1	Structural problem	943,695	77,651,847	82.3
Emilia_923	Structural PROBLEM	923,136	40,373,538	43.7
boneS10	Model Reduction problem	914,898	40,878,708	44.7
PFlow_742	2D/3D problem	742,793	37,138,461	50.0
tmt_sym	Electromagnetics problem	726,713	5,080,961	7.0
apache2	Structural problem	715,176	4,817,870	6.7
Fault_639	Structural problem	638,802	27,245,944	42.7
parabolic_fem	Computational fluid dynamics problem	525,825	3,674,625	7.0
bundle_adj	Computer vision problem	513,351	20,207,907	39.4
af_shell8	Subsequent structural problem	504,855	17,579,155	34.8
af_shell4	Subsequent structural problem	504,855	17,562,051	34.8
af_shell3	Subsequent structural problem	504,855	17,562,051	34.8
af_shell7	Subsequent structural problem	504,855	17,579,155	34.8
inline_1	Structural problem	503,712	36,816,170	73.1
af_0_k101	Structural problem	503,625	17,550,675	34.8
af_4_k101	Structural problem	503,625	17,550,675	34.8
af_3_k101	Structural problem	503,625	17,550,675	34.8
af_2_k101	Structural problem	503,625	17,550,675	34.8

Next, we consider the deflation method. When we use the deflation method, the additional cost is in calculating $P^{\top}A$. We estimate the data transferred for $P^{\top}A$ to be almost the same as that for SC preconditioning because both AW and W are dense matrices of identical size. Consequently, we can use (17) for the ICCG solver with deflation.

Based on our expectation for the reduction of the iteration count and (17), we can set the number of sample vectors, m. For example, when we expect a 40% reduction as a result of using the convergence acceleration method for the problem of $nnz_{\text{av}} = 30$, $\tilde{m}(\leq m)$ should be less than 20.

Next, we discuss the setup cost for the auxiliary matrices. The dominant part of the cost is given by the Gram–Schmidt process, the $\tilde{m}$ times sparse matrix-vector multiplication for $\boldsymbol{AE}$, the dense matrix-matrix product of $\boldsymbol{E}^{\top}$ and $(\boldsymbol{AE})$, and the dense matrix-vector product for (12). Because $\tilde{m}$ is typically much smaller than n, the cost to solve the $\tilde{m}$-dimensional eigenvalue problem is negligible compared with the computational costs for the above four kernels. Because the kernel

TABLE 2 Numerical results (sequential solver, $\boldsymbol{b} = (1, 1, \ldots, 1)^{\top}$).

		Queen_4147			Bump_2911			G3_circuit			Flan_1565			Hook_1498		
Solver	θ	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t
ICCG		-	3128	2763	-	1551	584	-	898	44.8	-	3124	996	-	1617	287
ES-SC-ICCG	10^{-3}	20	995	1039	20	526	249	18	705	70.1	20	1082	398	20	472	108
	10^{-4}	19	2041	2121	18	824	382	9	707	54.8	19	1212	449	13	676	144
	10^{-5}	7	2816	2667	5	1118	445	1	887	49.6	8	1766	596	5	1080	209
ES-D-ICCG	10^{-3}	20	993	1036	20	459	218	18	702	70.1	20	942	347	20	469	108
	10^{-4}	19	2044	2120	18	821	381	9	706	54.1	19	1213	443	13	675	144
	10^{-5}	7	2818	2670	5	1117	449	1	887	49.8	8	1762	595	5	1078	209

		StocF-1465			Geo_1438			Serena			thermal2			ecology2		
Solver	θ	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t
ICCG		-	56,109	4741	-	443	79.6	-	301	55.7	-	2281	141	-	1823	49.7
ES-SC-ICCG	10^{-3}	20	14,780	2011	15	248	58.2	7	243	49.6	20	849	89	20	813	50.1
	10^{-4}	20	14,775	2001	2	387	72.5	0	-	-	17	994	99	15	902	48.5
	10^{-5}	20	14,775	1998	0	-	-	0	-	-	4	1523	111	5	1329	50.3
ES-D-ICCG	10^{-3}	20	14,731	1992	15	248	55.9	7	242	49.5	20	847	89	20	808	49.9
	10^{-4}	20	14,717	1988	2	386	72.9	0	-	-	17	992	99	15	899	48.3
	10^{-5}	20	14,717	2001	0	-	-	0	-	-	4	1519	111	5	1328	49.7

		bone010			ldoor			audikw_1			Emilia_923			boneS10		
Solver	θ	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t
ICCG		-	4162	801	-	2160	293	-	2629	583	-	462	53.6	-	8532	1275
ES-SC-ICCG	10^{-3}	20	943	213	20	658	111	20	745	185	20	218	32.2	20	2688	486
	10^{-4}	18	967	216	16	1073	174	9	1138	265	19	266	38.9	20	2688	487
	10^{-5}	13	1302	280	3	1663	238	4	1521	343	5	373	46.5	20	2688	488
ES-D-ICCG	10^{-3}	20	935	211	20	655	110	20	756	188	20	201	29.7	20	2682	485
	10^{-4}	18	962	215	16	1072	173	9	1084	253	19	265	38.9	20	2682	485
	10^{-5}	13	1293	278	3	1662	237	4	1586	358	5	374	46.8	20	2682	486

		PFlow_742			tmt_sym			apache2			Fault_639			parabolic_fem		
Solver	θ	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t
ICCG		-	33,076	3357	-	1252	35.9	-	768	16.5	-	2187	177	-	1131	18.9
ES-SC-ICCG	10^{-3}	20	10,359	1299	20	507	27.0	19	359	16.0	20	806	83	18	671	22.4
	10^{-4}	20	10,360	1299	15	613	29.3	12	429	15.7	15	1366	134	7	862	20.7
	10^{-5}	20	10,359	1299	3	1013	34.7	2	663	17.1	4	1905	164	0	-	-
ES-D-ICCG	10^{-3}	20	10,269	1287	20	501	26.5	19	360	16.3	20	798	82	18	670	22.3
	10^{-4}	20	10,269	1287	15	610	29.1	12	428	15.8	15	1364	133	7	861	20.7
	10^{-5}	20	10,269	1285	3	1011	33.9	2	662	16.8	4	1901	164	0	-	-

TABLE 2 Continued

		bundle_adj			af_shell8			af_shell4			af_shell3			af_shell7		
Solver	θ	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t
ICCG		-	42,809	2275	-	1048	52.0	-	1048	52.0	-	1048	52.3	-	1048	53.0
ES-SC-ICCG	10^{-3}	20	11,705	824	18	483	31.4	18	481	31.1	18	481	31.4	18	483	31.5
	10^{-4}	18	11,533	793	9	614	35.8	9	615	35.3	9	615	35.7	9	614	35.5
	10^{-5}	17	11,460	781	0	-	-	0	-	-	0	-	-	0	-	-
ES-D-ICCG	10^{-3}	20	9740	686	18	481	31.4	18	479	31.0	18	479	31.4	18	481	31.5
	10^{-4}	18	10,117	698	9	613	35.4	9	615	35.5	9	615	35.7	9	613	35.6
	10^{-5}	17	10,532	717	0	-	-	0	-	-	0	-	-	0	-	-
		inline_1			**af_0_k101**			**af_4_k101**			**af_3_k101**			**af_2_k101**		
Solver	θ	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t
ICCG		-	8487	879	-	12,953	636	-	9993	489	-	8519	423	-	13,092	648
ES-SC-ICCG	10^{-3}	20	2573	311	20	4153	276	20	3093	204	20	2632	176	20	4194	279
	10^{-4}	20	2572	310	20	4153	276	20	3094	204	20	2633	176	20	4194	279
	10^{-5}	19	2573	309	20	4153	275	20	3094	204	20	2632	176	20	4194	278
ES-D-ICCG	10^{-3}	20	2570	311	20	4150	275	20	3085	203	20	2624	175	20	4189	279
	10^{-4}	20	2571	311	20	4150	276	20	3086	203	20	2629	175	20	4189	278
	10^{-5}	19	2571	309	20	4150	275	20	3086	204	20	2624	175	20	4189	279

of the Gram–Schmidt process is computationally bounded, its computational time T_{GS} is estimated by

$$T_{GS} = 2nm^2/f, \tag{18}$$

where f is the FLOPS of the processing core. The matrix-vector multiplication kernel is typically memory bound. Accordingly, the computational time for the SpMV is estimated by

$$T_{AE} = (12n + 12nnz)\tilde{m}/b_m. \tag{19}$$

Moreover, the computational time for (12) is estimated by

$$T_W = (8n + 8\tilde{m}n)/b_m. \tag{20}$$

The matrix-matrix multiplication kernel is computationally bound, and its computational time is estimated by

$$T_{E^\top AE} = 2n\tilde{m}^2/f. \tag{21}$$

Consequently, the computational time for the auxiliary matrix setup, T_{AM}, is estimated by

$$T_{AM} = T_{GS} + T_{AE} + T_W + T_{E^\top AE} \tag{22}$$

$$= 2n(\tilde{m}^2 + m^2)/f + (8n + 20\tilde{m}n + 12\tilde{m} \cdot nnz)/b_m. \tag{23}$$

On a recent computer system, the BYTE/FLOPS ratio ($= b_m/f$) is typically less than 0.1. For example, the ratio for the system used in the numerical test was 0.087. Consequently, we assume that $f = 10b_m$. Moreover, for simplicity, we assume that $\tilde{m} = m = 20$ and $nnz_{\rm av} = 30$. From (15) and (23), for these settings, the computational cost for the setup is comparable with that for ten ICCG iterations. Because it is not rare that the number of iterations exceeds several hundred for a practical engineering problem and k_t is typically not small, the setup cost can be amortized in the following solution steps.

TABLE 3 Numerical results (sequential solver, $\boldsymbol{b}$: random vector).

		Queen_4147			Bump_2911			G3_circuit			Flan_1565			Hook_1498		
Solver	θ	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t
ICCG	-	-	3140	2776	-	1544	564	-	926	46.1	-	3196	1010	-	1613	282
ES-SC-ICCG	10^{-3}	20	2546	2648	20	906	428	19	865	88.8	20	1013	372	20	554	127
	10^{-4}	19	2566	2652	17	921	422	10	891	70.6	19	1048	382	13	672	143
	10^{-5}	7	2783	2626	5	1114	441	0	-	-	9	1524	518	5	1076	208
ES-D-ICCG	10^{-3}	20	2542	2652	20	900	424	19	863	88.2	20	1011	372	20	553	126
	10^{-4}	19	2561	2654	17	917	418	10	890	70.2	19	1048	383	13	671	141
	10^{-5}	7	2779	2630	5	1113	439	0	-	-	9	1523	517	5	1075	206

		StocF-1465			Geo_1438			Serena			thermal2			ecology2		
Solver	θ	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t
ICCG		-	55,799	4714	-	441	81.3	-	299	58.1	-	2261	141	-	1902	51.7
ES-SC-ICCG	10^{-3}	20	29,693	4001	15	252	54.7	7	242	49.1	20	959	101	20	853	52.5
	10^{-4}	20	29,693	4011	2	385	71.9	0	-	-	17	1020	101	16	933	51.5
	10^{-5}	20	29,693	4007	0	-	-	0	-	-	4	1526	112	5	1268	47.8
ES-D-ICCG	10^{-3}	20	29,600	3990	15	251	55.1	7	241	49.6	20	957	100	20	850	52.2
	10^{-4}	20	29,600	3993	2	384	72.3	0	-	-	17	1019	101	16	930	51.2
	10^{-5}	20	29,600	4002	0	-	-	0	-	-	4	1524	111	5	1267	47.4

		bone010			ldoor			audikw_1			Emilia_923			boneS10		
Solver	θ	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t
ICCG		-	4189	804	-	2143	293	-	2420	533	-	459	54	-	8515	1274
ES-SC-ICCG	10^{-3}	20	996	225	20	1230	208	19	858	214	20	266	39	20	2733	492
	10^{-4}	17	1060	234	16	1259	204	8	1220	284	18	276	40	20	2733	494
	10^{-5}	13	1288	277	3	1649	236	4	1604	364	5	371	46	20	2733	494
ES-D-ICCG	10^{-3}	20	989	221	20	1227	206	19	861	214	20	267	40	20	2728	490
	10^{-4}	17	1053	231	16	1256	203	8	1207	280	18	276	40	20	2728	490
	10^{-5}	13	1281	273	3	1648	234	4	1579	356	5	370	47	20	2728	490

		PFlow_742			tmt_sym			apache2			Fault_639			parabolic_fem		
Solver	θ	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t
ICCG		-	32,971	3311	-	1256	35.8	-	770	16.8	-	2172	176	-	1208	20.1
ES-SC-ICCG	10^{-3}	20	14,148	1774	20	562	29.8	20	342	15.7	20	1601	162	18	835	27.6
	10^{-4}	20	14,148	1772	15	617	29.5	12	445	16.4	16	1629	159	9	889	22.8
	10^{-5}	20	14,148	1769	3	1002	33.9	2	653	16.7	4	1899	162	0	-	-
ES-D-ICCG	10^{-3}	20	14,042	1758	20	556	29.8	20	341	15.7	20	1595	163	18	833	27.6
	10^{-4}	20	14,042	1758	15	614	29.5	12	444	16.4	16	1623	159	9	888	22.6
	10^{-5}	20	14,042	1754	3	1000	33.8	2	653	16.6	4	1896	162	0	-	-

TABLE 3 Continued

		bundle_adj			af_shell8			af_shell4			af_shell3			af_shell7		
Solver	θ	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t
ICCG		-	43,578	2325	-	1038	51.2	-	1039	51.7	-	1039	51.1	-	1038	51.6
ES-SC-ICCG	10^{-3}	20	11,997	846	18	510	32.8	18	513	33.5	18	513	33.2	18	510	33.2
	10^{-4}	19	11,394	795	9	606	34.9	9	602	34.9	9	602	34.7	9	606	35.2
	10^{-5}	19	11,394	795	0	-	-	0	-	-	0	-	-	0	-	-
ES-D-ICCG	10^{-3}	20	10,110	711	18	508	33.0	18	511	33.4	18	511	33.1	18	508	33.1
	10^{-4}	19	10,030	700	9	605	34.9	9	601	34.9	9	601	34.7	9	605	35.1
	10^{-5}	19	10,030	698	0	-	-	0	-	-	0	-	-	0	-	-

		inline_1			af_0_k101			af_4_k101			af_3_k101			af_2_k101		
Solver	θ	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t
ICCG		-	8464	870	-	12,961	641	-	9974	495	-	8501	420	-	12,970	641
ES-SC-ICCG	10^{-3}	20	2686	324	20	5657	372	20	3582	238	20	2680	177	20	5413	361
	10^{-4}	20	2686	324	20	5657	372	20	3582	238	20	2680	177	20	5413	360
	10^{-5}	19	2686	322	20	5657	372	20	3582	238	20	2680	177	20	5413	360
ES-D-ICCG	10^{-3}	20	2683	324	20	5649	376	20	3575	238	20	2676	177	20	5409	356
	10^{-4}	20	2683	324	20	5649	376	20	3575	238	20	2676	177	20	5409	356
	10^{-5}	19	2684	322	20	5649	376	20	3575	237	20	2676	177	20	5409	355

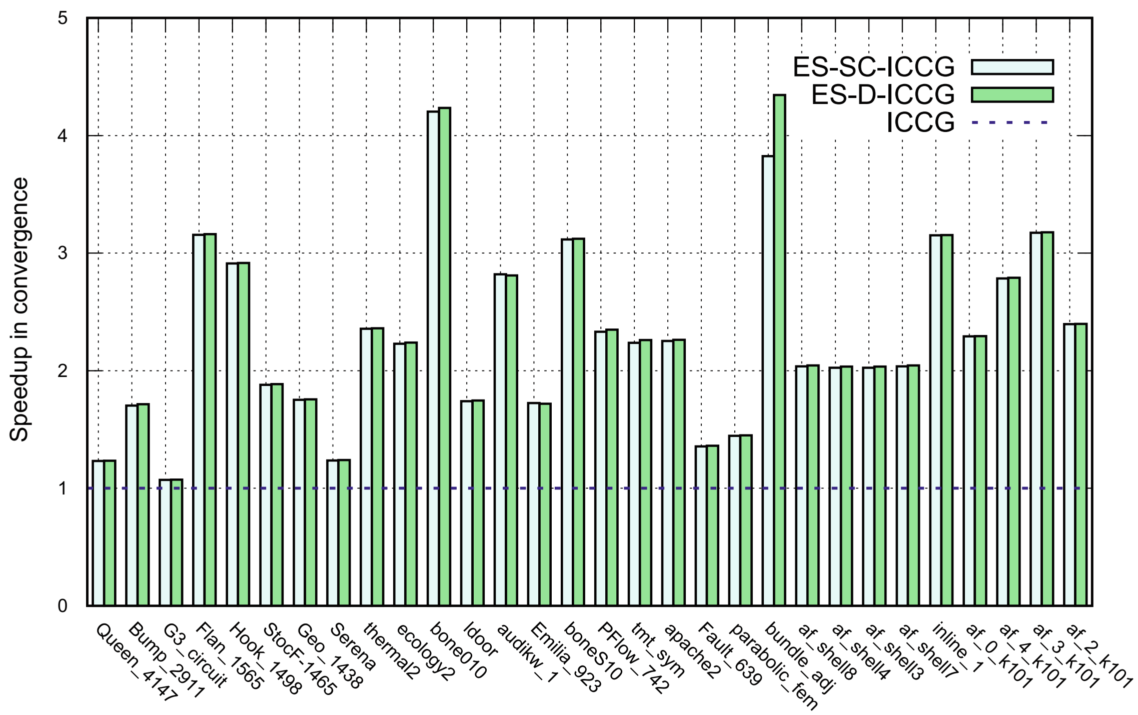

FIGURE 1 Speedup in convergence of ES-SC-ICCG and ES-D-ICCG over ICCG ($\boldsymbol{b}$: random vector).

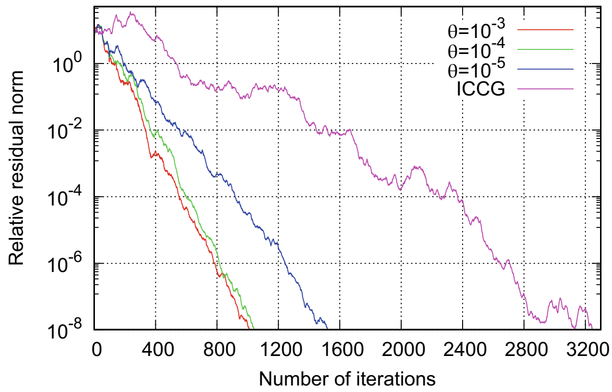

FIGURE 2 Convergence behavior of ES-SC-ICCG (dataset: Flan_1565).

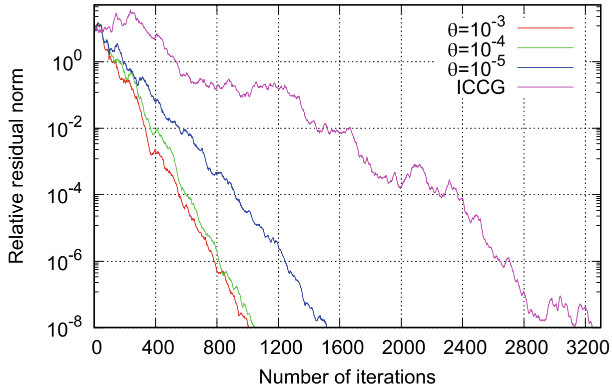

FIGURE 3 Convergence behavior of ES-D-ICCG (dataset: Flan_1565).

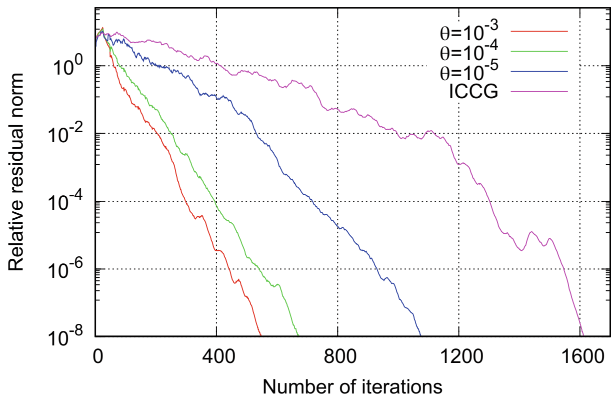

FIGURE 4 Convergence behavior of ES-SC-ICCG (dataset: Hook_1498).

5 | NUMERICAL RESULTS

5.1 | Test conditions

We conducted numerical tests to examine the effect of convergence acceleration methods (SC and deflation) based on our algebraic auxiliary matrix generation method. For the test matrix, we downloaded 30 relatively large matrices from the SuiteSparse Matrix Collection[32] and applied the diagonal scaling to them. We selected s.p.d. matrices that were mainly

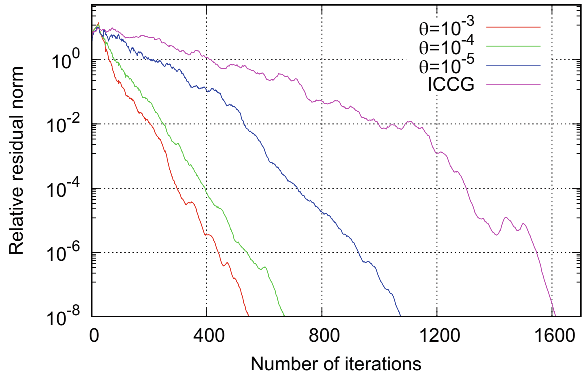

FIGURE 5 Convergence behavior of ES-D-ICCG (dataset: Hook_1498).

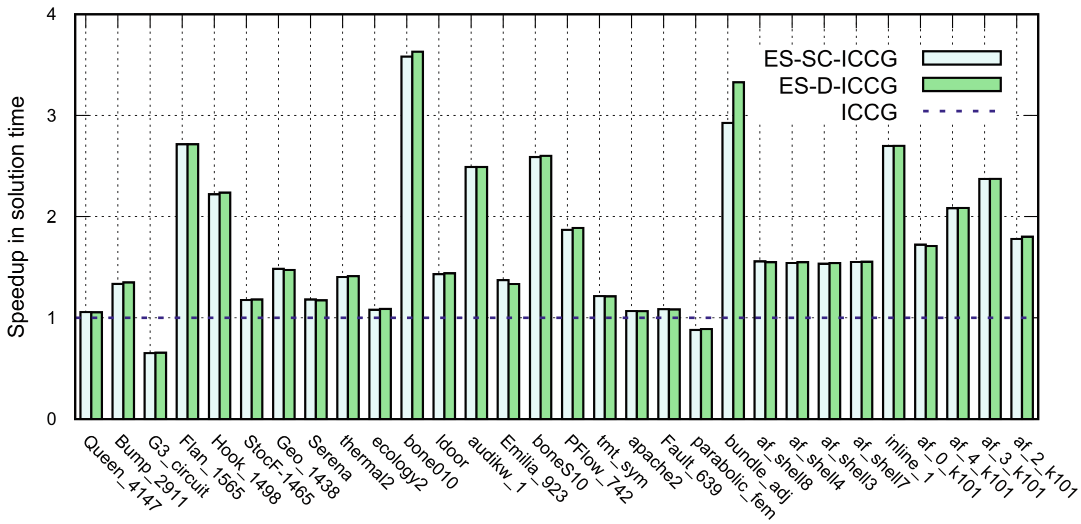

FIGURE 6 Speedup in the computational time of ES-SC-ICCG and ES-D-ICCG over ICCG ($\boldsymbol{b}$: random vector).

derived from computational science or engineering problems. Table 1 shows the properties of the test matrices. For each coefficient matrix, we solved a linear system of equations six times. The convergence criterion was that the relative residual 2-norm was less than 10^{-8}. When the first solution process was complete, we generated the auxiliary matrix and used it in the following five solution processes, in which we evaluated the solver performance. For the right-hand side vector, we used two types of vectors: a vector of ones and a random vector. In the former case, the linear systems used for the auxiliary matrix generation and the evaluation were identical. When we used random vectors, we solved linear systems of different right-hand side vectors. In this article, we report the results when we set the number of sampled vectors, m, to 20.

We conducted numerical tests on a computational node of Fujitsu CX2550 (M4) at the Information Initiative Center, Hokkaido University. The node is equipped with two Intel Xeon (Gold6148, Skylake) processors, each of which has 20 cores, and 384 GB memory. The program code was written in C and OpenMP directives were used for multi-threading. Intel C compiler version 19.1.3.304 was used with the option of "-O3 -qopenmp -ipo -xCORE-AVX512." In the tests for parallel multithreaded solvers, we used 40 threads.

5.2 | Numerical results for the sequential solver

5.2.1 | Performance evaluation

Table 2 lists the numerical results for the standard ICCG solver and its variants with the introduced convergence acceleration techniques when we used a vector of ones for the right-hand side. ES-SC-ICCG denotes the CG solver with IC and

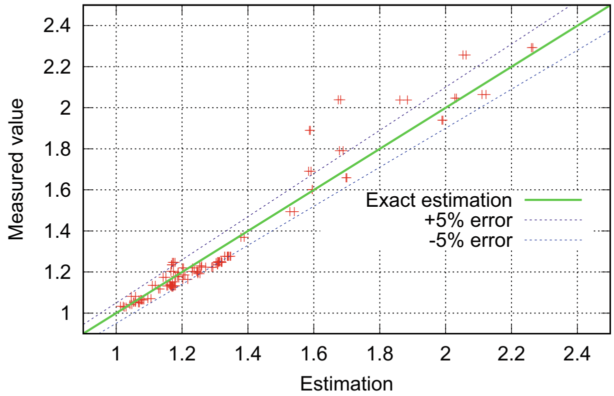

FIGURE 7 Comparison of the estimated and measured values of ratio of the computational time of an ES-SC-ICCG or ES-D-ICCG iteration to that of an ICCG iteration.

TABLE 4 Solver performance using sampling method B (sequential solver, $\boldsymbol{b} = (1, 1, \ldots, 1)^{\mathrm{T}}$).

		Flan_1565			Hook_1498		
Solver	θ	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t
ES-SC-ICCG	10^{-3}	20	1584	586	15	1075	233
	10^{-4}	15	1927	690	7	1157	229
	10^{-5}	7	2094	706	4	1208	230
ES-D-ICCG	10^{-3}	20	1579	585	15	1072	232
	10^{-4}	15	1925	687	7	1156	229
	10^{-5}	7	2093	704	5	1207	229

SC preconditioning based on the proposed error vector sampling method. ES-D-ICCG denotes the deflated ICCG solver using our technique. The table shows the average computational time (s) for five solution steps, which is denoted by T_t. Table 3 shows the results when we used random vectors for the right-hand side. The table shows the average number of iterations and computational time for five solution steps. The numerical results indicate that both solvers based on the proposed method achieved convergence acceleration for all 60 test cases (30 datasets × 2 types of right-hand side vectors). The convergence acceleration was significant for some datasets. In the numerical tests using the vector of ones, the acceleration method attained a more than three-fold speedup in convergence for 16 out of 30 datasets. Even when we used random right-hand side vectors, convergence was more than twice as fast as that of the ICCG solver for 20 out of 30 datasets, as shown in Figure 1.

Figures 2–5 show the convergence behaviors of the ES-SC-ICCG and ES-D-ICCG solvers in the second solution step for the Flan_1565 and Hook_1498 datasets when we used a random vector for the right-hand side. The figures also confirm the effectiveness of SC and deflation based on our technique. The numerical results imply that the larger θ typically leads to larger $\tilde{m}$ and better convergence. This characteristic is confirmed by the results listed in Tables 2 and 3. Figures 2–5 show that the convergence behaviors of the two solvers were identical, although each solver shifts small eigenvalues in a different way.[37] We examined the convergence behavior of the residual norm for all test cases and observed that the convergence properties of the two solvers were almost the same for most test cases. This result indicates that the effects of SC preconditioning and deflation are similar when the coefficient matrix is diagonally scaled and identical subspaces that correspond to eigenvectors associated with small eigenvalues are used.

Next, we examine the computational time to solution. Table 2 shows that the solution time reduced in 28 out of 30 cases in the tests using the right-hand side vector of ones. For 16 datasets, the computational time of the solvers using our technique (ES-SC-ICCG and ES-D-ICCG) reduced to less than half of that of the normal ICCG solver. The performance difference between the two solvers ES-SC-ICCG and ES-D-ICCG was marginal. In the numerical test using random vectors, the computational time also reduced in 28 out of 30 cases. Table 3 and Figure 6 show the effectiveness of our technique in the random vector test. In the tests, we did not attain performance improvement on the G3_circuit and parabolic_fem

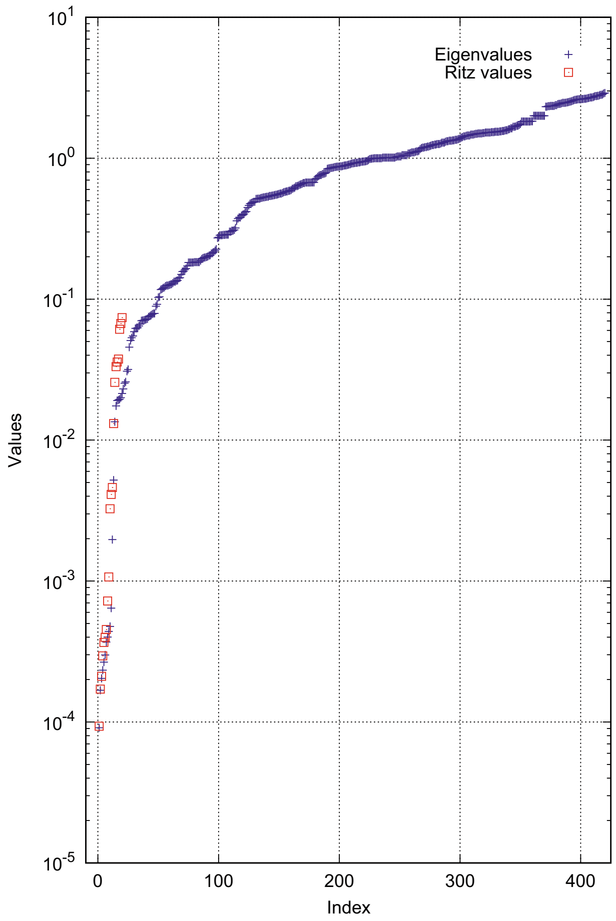

FIGURE 8 Comparison of eigenvalues and Ritz values (dataset: bcsstik06, n=420, m=20).

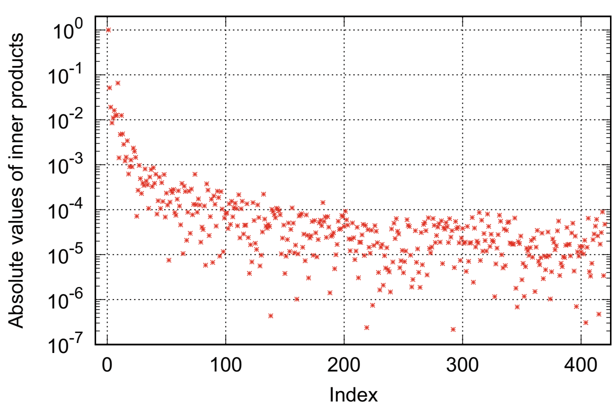

FIGURE 9 Absolute values of inner products, $\|(\tilde{\boldsymbol{v}}_1, \boldsymbol{v}_{ir})\|, (ir = 1, \ldots, 420)$.

datasets, which have relatively small nnz_{av} values. In (17), γ_{sciccg} enlarged when nnz_{av} decreased. This means that it becomes difficult to obtain performance improvement in the solution time using SC preconditioning and the deflation method; that is, for a dataset with a small nnz_{av} value, the convergence rate should be substantially improved by the limited number of sample vectors to achieve solver performance improvement. In the numerical test, the ES-SC-ICCG and ES-D-ICCG solvers obtained their best results for 12 out of 30 datasets when $\tilde{m}$ was equal to m (=20). For these datasets, an increase in the number of sample vectors, m, possibly improves solver performance.

TABLE 5 Numerical results of deflated CG solver in Reference 18.

Queen_4147		Bump_2911		G3_circuit		Flan_1565		Hook_1498	
#Ite.	T_t	#Ite.	T_t	#Ite.	T_t	#Ite.	T_t	#Ite.	T_t
3118	2607	1534	532	879	47.7	3112	923	1599	271
StocF-1465		**Geo_1438**		**Serena**		**thermal2**		**ecology2**	
#Ite.	T_t	#Ite.	T_t	#Ite.	T_t	#Ite.	T_t	#Ite.	T_t
56,175	4759	2083	150	282	50.2	2263	141	2148	110
bone010		**ldoor**		**audikw_1**		**Emilia_923**		**boneS10**	
#Ite.	T_t	#Ite.	T_t	#Ite.	T_t	#Ite.	T_t	#Ite.	T_t
6277	1254	2141	306	2369	486	453	55.9	9993	1574
PFlow_742		**tmt_sym**		**apache2**		**Fault_639**		**parabolic_fem**	
#Ite.	T_t	#Ite.	T_t	#Ite.	T_t	#Ite.	T_t	#Ite.	T_t
33,064	3161	1447	64.4	781	20.9	2168	171	1161	24.0
bundle_adj		**af_shell8**		**af_shell4**		**af_shell3**		**af_shell7**	
#Ite.	T_t	#Ite.	T_t	#Ite.	T_t	#Ite.	T_t	#Ite.	T_t
48,942	2805	1066	52.2	1064	56.7	1064	67.3	1066	52.0
inline_1		**af_0_k101**		**af_4_k101**		**af_3_k101**		**af_2_k101**	
#Ite.	T_t	#Ite.	T_t	#Ite.	T_t	#Ite.	T_t	#Ite.	T_t
8474	817	13,540	678	9976	472	8498	429	13,077	619

5.2.2 | Verification of the model for computational time per iteration

The application of SC or deflation typically leads to an increase in the computational cost per iteration. In this section, we examine the performance model for the iteration cost introduced in Section 4.4. In Figure 7, we plot the measured and estimated values for the ratio of the computational time of an ES-SC-ICCG or ES-D-ICCG iteration to that of an ICCG iteration. The estimated values for the two solvers are given in (17). Figure 7 shows the results for all test cases, although we plot only one mark for identical $\tilde{m}$. For most test cases, Equation (17) obtained a good estimation of the ratio, and the error of the estimation was within ±5%. Consequently, (17) can be used for the estimation of the additional cost for SC or deflation. However, in some test cases, particularly when the measured value was over 2.0, we observed a relatively large estimation error. These results arose for the G3_circuit, ecology2, and apache2 datasets. The coefficient matrices of these datasets commonly had a small number of nonzero elements per row (nnz_{av}) and a relatively structured nonzero element pattern; that is, these matrices were derived from relatively simple problems and (15) tended to overestimate the ratio for such problems. Moreover, (17) implies that the influence of the additional cost of the convergence acceleration method on the computational time per iteration tends to enlarge when nnz_{av} is small. Accordingly, we recommend that the number of sampling vectors m (the upper bound of $\tilde{m}$) should be small for a problem with small nnz_{av}.

5.2.3 | Discussions (other factors that affect solver performance)

Sampling method

In preliminary analyses, we compared two sampling methods: A and B. Table 4 shows the results of the solver using sampling method B on Flan_1565 and Hook_1498. In the comparison of Tables 2 and 4, sampling method A obtained better convergence acceleration than method B. Because we observed this tendency for other test datasets, we decided to mainly use sampling method A in our numerical tests. Moreover, the numerical test implied that the additional sampling

TABLE 6 Numerical results (parallel solver, $\boldsymbol{b} = (1, 1, \dots, 1)^{\top}$).

		Queen_4147			Bump_2911			G3_circuit			Flan_1565			Hook_1498		
Solver	θ	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t
ICCG		-	4663	215	-	3455	73.4	-	1461	5.40	-	4911	86.6	-	2312	23.4
ES-SC-ICCG	10^{-3}	20	1532	89	20	1062	30.8	20	468	3.81	20	1504	31.3	19.0	808	11.6
	10^{-4}	20	1532	88	17	1814	51.1	13	1067	7.22	18	1777	37.3	13.0	1036	13.9
	10^{-5}	6	4258	218	2	2977	68.1	2	1392	5.67	9	2415	47.2	5.0	1562	18.8
ES-D-ICCG	10^{-3}	20	1530	91	20	1062	31.3	20	468	3.90	20	1502	33.0	19.0	807	12.1
	10^{-4}	20	1530	93	17	1811	51.4	13	1066	7.08	18	1776	38.2	13.0	1035	14.3
	10^{-5}	6	4252	216	2	2977	68.7	2	1392	5.73	9	2409	46.6	5.0	1561	18.9

		StocF-1465			Geo_1438			Serena			thermal2			ecology2		
Solver	θ	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t
ICCG		-	66,348	329	-	904	9.62	-	628	6.77	-	3583	12.0	-	2131	3.29
ES-SC-ICCG	10^{-3}	20	16,453	157	14	549	7.26	8	546	6.36	20	1128	8.1	20	885	4.40
	10^{-4}	20	16,453	154	2	779	8.24	0	-	-	17	1555	10.7	15	1039	4.19
	10^{-5}	20	16,453	157	0	-	-	0	-	-	4	2506	10.4	4	1656	3.95
ES-D-ICCG	10^{-3}	20	16,452	148	14	548	7.27	8	545	6.39	20	1126	8.0	20	882	4.38
	10^{-4}	20	16,452	147	2	778	8.40	0	-	-	17	1554	10.2	15	1037	4.33
	10^{-5}	20	16,452	150	0	-	-	0	-	-	4	2504	10.8	4	1654	4.05

		bone010			ldoor			audikw_1			Emilia_923			boneS10		
Solver	θ	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t
ICCG		-	7838	76.9	-	5227	38.0	-	2635	30.0	-	6542	42.1	-	14,690	119
ES-SC-ICCG	10^{-3}	20	2141	29.6	20	1503	14.8	20	816	11.4	20	1893	17.3	20	5166	55
	10^{-4}	18	2207	28.6	18	2199	20.7	7	1549	19.1	19	2991	27.5	20	5164	56
	10^{-5}	12	2925	35.7	3	4040	29.6	3	1798	21.2	7	4829	36.0	19	5446	58
ES-D-ICCG	10^{-3}	20	2138	28.9	20	1504	14.8	20	813	11.6	20	1895	17.8	20	5162	57
	10^{-4}	18	2203	28.9	18	2196	20.9	7	1545	19.3	19	2989	29.1	20	5161	56
	10^{-5}	12	2921	36.3	3	4036	30.8	3	1796	21.9	7	4823	37.1	19	5446	59

		PFlow_742			tmt_sym			apache2			Fault_639			parabolic_fem		
Solver	θ	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t	$\tilde{m}$	**#Ite.**	T_t
ICCG		-	37,485	214	-	1576	2.26	-	1056	1.31	-	5083	26.2	-	2125	1.58
ES-SC-ICCG	10^{-3}	20	11,633	95	20	638	2.40	19	408	1.51	20	1496	9.6	19	1419	3.50
	10^{-4}	20	11,633	95	14	777	2.33	12	494	1.27	18	3075	19.4	8	1326	1.89
	10^{-5}	20	11,633	99	3	1259	2.09	3	816	1.31	2	4735	22.6	0	-	-
ES-D-ICCG	10^{-3}	20	11,617	100	20	636	2.44	19	408	1.53	20	1495	9.6	19	1417	4.03
	10^{-4}	20	11,617	97	14	776	2.35	12	494	1.39	18	3074	18.8	8	1325	2.27
	10^{-5}	20	11,617	102	3	1257	2.30	3	815	1.43	2	4739	22.2	0	-	-

TABLE 6 Continued

Solver	θ	bundle_adj $\tilde{m}$	bundle_adj #Ite.	bundle_adj T_t	af_shell8 $\tilde{m}$	af_shell8 #Ite.	af_shell8 T_t	af_shell4 $\tilde{m}$	af_shell4 #Ite.	af_shell4 T_t	af_shell3 $\tilde{m}$	af_shell3 #Ite.	af_shell3 T_t	af_shell7 $\tilde{m}$	af_shell7 #Ite.	af_shell7 T_t
ICCG		-	64,356	797	-	1575	5.13	-	1575	5.31	-	1575	5.07	-	1575	4.91
ES-SC-ICCG	10^{-3}	20	14,407	208	20	537	2.58	20	539	2.59	20	539	2.62	20	537	2.56
	10^{-4}	20	14,407	207	11	764	3.08	11	764	3.10	11	764	3.17	11	764	3.07
	10^{-5}	19	14,104	200	0	-	-	0	-	-	0	-	-	0	-	-
ES-D-ICCG	10^{-3}	20	13,547	196	20	537	2.64	20	537	2.76	20	537	2.65	20	537	2.59
	10^{-4}	20	13,547	196	11	764	3.25	11	763	3.33	11	763	3.24	11	764	3.20
	10^{-5}	19	13,611	194	0	-	-	0	-	-	0	-	-	0	-	-

Solver	θ	inline_1 $\tilde{m}$	inline_1 #Ite.	inline_1 T_t	af_0_k101 $\tilde{m}$	af_0_k101 #Ite.	af_0_k101 T_t	af_4_k101 $\tilde{m}$	af_4_k101 #Ite.	af_4_k101 T_t	af_3_k101 $\tilde{m}$	af_3_k101 #Ite.	af_3_k101 T_t	af_2_k101 $\tilde{m}$	af_2_k101 #Ite.	af_2_k101 T_t
ICCG		-	23,064	115	-	16,157	48.6	-	12,458	45.5	-	10,595	34.9	-	16,249	57.6
ES-SC-ICCG	10^{-3}	20	6393	44	20	5026	24.7	20	4230	20.7	20	3567	17.1	20	4924	24.6
	10^{-4}	20	6393	43	20	5026	23.9	20	4230	19.7	20	3567	16.9	20	4924	23.7
	10^{-5}	18	8969	60	20	5026	24.1	20	4230	19.5	20	3567	16.8	20	4924	22.8
ES-D-ICCG	10^{-3}	20	6390	46	20	5018	24.8	20	4237	21.6	20	3574	18.0	20	4920	24.9
	10^{-4}	20	6390	46	20	5018	24.7	20	4237	20.8	20	3574	17.6	20	4920	24.5
	10^{-5}	18	8964	62	20	5018	25.2	20	4237	20.4	20	3574	17.5	20	4920	24.4

of the approximation vector when the residual norm increased or stagnated was effective for the improvement of the convergence acceleration effect. Because it is not straightforward to mathematically interpret the phenomenon, we intend to investigate the behavior of the error in the solution process in future work based on numerical tests.

Sampling of residual vectors

In this article, we consider the sampling of a relatively small number of vectors because it is practically important to save additional memory space and computational cost. Considering other related techniques, the sampling of residual vectors might be of interest. We have an intuitive perspective on the comparison of the sampling of error vectors and residual vectors. Because $A\boldsymbol{e}_s = \boldsymbol{r}_s$ holds, the components along eigenvectors corresponding small eigenvalues involved in $\boldsymbol{e}_s$ are numerically reduced in $\boldsymbol{r}_s$ by the multiplication of A, where $\boldsymbol{e}_s$ and $\boldsymbol{r}_s$ are the sampled error and residual vectors, respectively. Consequently, we expect that error vector sampling will be superior to residual vector sampling to capture (approximate) eigenvectors that correspond to small eigenvalues, which will lead to a better preconditioning effect for convergence. To verify our perspective, we conducted additional numerical tests of the solver using residual vector sampling. In the numerical test on Flan_1565 and Hook_1498, the results demonstrated that we could not obtain a small Ritz value less than 10^{-1} and the convergence acceleration of SC and deflation did not work well. The numerical results imply that error vector sampling outperforms residual vector sampling to construct an effective mapping operator for subspaces used in the convergence acceleration techniques.

Verification of the Ritz vector

In this section, we attempt to examine the property of the Ritz vector calculated by our technique using a small dataset (bccstk06: a 420×420 matrix). Figure 8 shows the eigenvalue distribution of the coefficient matrix and the Ritz values obtained by our method applied to a non-preconditioned CG solver. We confirmed that some small eigenvalues, including the smallest eigenvalue, were well approximated by the obtained Ritz values. Moreover, we checked the orthogonality of the normalized Ritz vector that corresponds to the smallest Ritz value, $\tilde{\boldsymbol{v}}_1$, to the normalized eigenvectors of A denoted by $\boldsymbol{v}_{ir}, (ir = 1, \dots, 420)$, where ir represents the index of eigenvalues in ascending order. Figure 9 shows the absolute value of the inner product $(\boldsymbol{v}_r, \boldsymbol{v}_{ir})$. The magnitude of $|(\tilde{\boldsymbol{v}}_1, \boldsymbol{v}_1)|$ is close to 1 and substantially larger than those of other inner products, most of which are less than 10^{-3}.

TABLE 7 Numerical results (parallel solver, $\boldsymbol{b}$: random vector).

		Queen_4147			Bump_2911			G3_circuit			Flan_1565			Hook_1498		
Solver	θ	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t
ICCG		-	4684	215	-	3437	73.9	-	1455	5.27	-	4906	83.6	-	2309	24.3
ES-SC-ICCG	10^{-3}	20	3762	217	20	1844	53.9	20	1157	9.54	20	1684	36.8	19.0	858	12.7
	10^{-4}	19	3760	217	17	1929	54.6	13	1203	8.15	18	1768	37.8	13.0	1027	13.6
	10^{-5}	6	4166	212	2	2967	67.8	2	1388	5.74	9	2411	47.3	5.0	1557	18.3
ES-D-ICCG	10^{-3}	20	3761	220	20	1842	53.5	20	1159	9.71	20	1684	37.7	19	857	12.4
	10^{-4}	19	3758	218	17	1927	53.8	13	1202	8.24	18	1766	38.5	13	1026	13.9
	10^{-5}	6	4164	212	2	2964	67.0	2	1388	5.94	9	2410	48.2	5.0	1556	18.6

		StocF-1465			Geo_1438			Serena			thermal2			ecology2		
Solver	θ	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t
ICCG		-	51,167	258	-	901	9.37	20	625	6.49	-	3569	12.5	-	2225	3.48
ES-SC-ICCG	10^{-3}	20	37,038	341	14	548	7.26	8	546	6.53	20	1343	9.4	20	937	4.67
	10^{-4}	20	37,038	341	2	774	8.49	0	-	-	17	1597	10.4	17	1045	4.74
	10^{-5}	20	37,038	342	0	-	-	0	-	-	4	2466	10.6	5	1471	3.69
ES-D-ICCG	10^{-3}	20	37,036	342	14	547	7.22	8	545	6.69	20	1342	9.1	20	935	4.55
	10^{-4}	20	37,036	340	2	773	8.57	0	-	-	17	1596	10.2	17	1044	4.64
	10^{-5}	20	37,036	337	0	-	-	0	-	-	4	2465	10.8	5	1469	3.99

		bone010			ldoor			audikw_1			Emilia_923			boneS10		
Solver	θ	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t
ICCG			8190	84.1	-	5198	36.2	-	2633	30.2	-	6507	43.0	-	14,637	119
ES-SC-ICCG	10^{-3}	20	2077	28.3	20	2222	22.0	20	1031	14.5	20	3989	35.4	20	5422	60
	10^{-4}	19	2100	28.0	18	2320	21.9	7	1541	19.4	19	4027	36.7	20	5422	59
	10^{-5}	12	2883	35.6	3	4019	30.1	3	1791	21.7	7	4798	36.8	19	5432	59
ES-D-ICCG	10^{-3}	20	2074	27.9	20	2220	22.6	20	1028	14.9	20	3986	37.7	20	5419	61
	10^{-4}	19	2096	28.4	18	2319	22.1	7	1536	19.9	19	4026	37.9	20	5419	62
	10^{-5}	12	2870	37.1	3	4016	31.2	3	1789	21.8	7	4797	36.9	19	5428	60

		PFlow_742			tmt_sym			apache2			Fault_639			parabolic_fem		
Solver	θ	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t
ICCG		-	37,486	221	-	1569	2.13	-	1055	1.34	-	5047	22.7	-	2583	1.89
ES-SC-ICCG	10^{-3}	20	15,380	127	20	673	2.53	19	454	1.68	20	3828	24.9	19	1798	4.33
	10^{-4}	20	15,380	127	14	784	2.51	12	499	1.37	18	3872	24.3	9	1885	2.80
	10^{-5}	20	15,380	130	3	1256	2.12	3	810	1.21	2	4710	22.4	0	-	-
ES-D-ICCG	10^{-3}	20	15,354	131	20	671	2.55	19	454	1.64	20	3830	24.9	19	1797	4.84
	10^{-4}	20	15,354	130	14	782	2.53	12	498	1.51	18	3869	24.6	9	1885	3.42
	10^{-5}	20	15,354	133	3	1255	2.30	3	809	1.38	2	4708	23.0	0	-	-

TABLE 7 Continued

		bundle_adj			af_shell8			af_shell4			af_shell3			af_shell7		
Solver	θ	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t
ICCG		-	55,336	701	-	1572	5.07	-	1572	5.06	-	1572	4.95	-	1572	5.04
ES-SC-ICCG	10^{-3}	20	13,873	199	20	557	2.69	20	556	2.71	20	556	2.68	20	557	2.75
	10^{-4}	20	13,873	200	11	760	3.05	11	760	3.04	11	760	3.04	11	760	3.03
	10^{-5}	19	13,947	200	0	-	-	0	-	-	0	-	-	0	-	-
ES-D-ICCG	10^{-3}	20	13,287	192	20	555	2.76	20	555	2.78	20	555	2.77	20	555	2.75
	10^{-4}	20	13,287	193	11	759	3.23	11	759	3.21	11	759	3.19	11	759	3.22
	10^{-5}	19	13,796	199	0	-	-	0	-	-	0	-	-	0	-	-
		inline_1			**af_0_k101**			**af_4_k101**			**af_3_k101**			**af_2_k101**		
Solver	θ	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t	$\tilde{m}$	#Ite.	T_t
ICCG		-	23,054	124	-	16,121	51.5	-	12,425	39.9	-	10,584	33.7	-	16,237	50.9
ES-SC-ICCG	10^{-3}	20	8738	60	20	6977	33.1	20	4327	20.8	20	3877	19.6	20	6526	31.0
	10^{-4}	20	8738	61	20	6977	33.1	20	4327	20.9	20	3877	18.8	20	6526	31.1
	10^{-5}	18	9090	62	20	6977	33.5	20	4327	20.5	20	3877	18.5	20	6526	31.3
ES-D-ICCG	10^{-3}	20	8732	61	20	6966	34.0	20	4322	21.2	20	3873	19.2	20	6523	32.1
	10^{-4}	20	8732	62	20	6966	34.6	20	4322	21.2	20	3873	19.3	20	6523	31.7
	10^{-5}	18	9086	65	20	6966	34.5	20	4322	21.3	20	3873	19.3	20	6523	32.2

Comparison with other solvers

For a further examination of our technique, we implemented the well-known deflated CG solver proposed in Reference 18 and performed a numerical test. In the numerical test, we solved the deflated system using the ICCG method and set the maximum number of deflated vectors to 20. We used a vector of ones for the right-hand side. The solver is denoted by D-CG in this article. Table 5 shows the number of iterations and the solution time of the D-CG solver for 30 datasets. Our solver based on error vector sampling outperformed the D-CG solver for 29 out of 30 datasets. Moreover, the solution time of our solver was less than half that of D-CG for 22 datasets. Although the numerical result implies that our technique is effective, we consider that further investigation is required. As shown in Reference 24, many methods exist to (algebraically) determine the subspace or deflation vectors. However, because solver performance significantly depends on properties of the target problem, it may be difficult to develop the best method for a wide variety of problems. Therefore, in our future work, we will compare our method with other related techniques in a specific problem domain.

5.3 | Numerical results for the parallel solver

In this section, we report the results for the parallel (multithreaded) solver. The parallelization of the CG solver is relatively straightforward. However, the IC preconditioning step that consists of forward and backward substitutions is not naturally parallelized. Various parallel processing methods exist. We used a simple but popular method, that is, block Jacobi IC preconditioning.[38] The parallelization of SC preconditioning and the deflation method is relatively easy. The computationally dominant part of these methods is dense matrix vector multiplication, which can be straightforwardly parallelized. Because $\tilde{m}$ is typically tiny, we sequentially solve the linear system having an $\tilde{m} \times \tilde{m}$ coefficient matrix $W^\top AW$ that is involved in the methods.

Tables 6 and 7 list the numerical results of the parallel ICCG solver and its variants using the proposed techniques when a vector of ones and random vectors were used for the right-hand side vectors, respectively. From the viewpoint of convergence, the results for the parallel solver were similar to those of the sequential solver. For all 60 test cases (30 datasets × 2 types of right-hand side vectors), the proposed method attained convergence acceleration. When we used a vector of ones, convergence was more than twice as fast as that of the parallel ICCG solver for 27 out of 30 datasets.

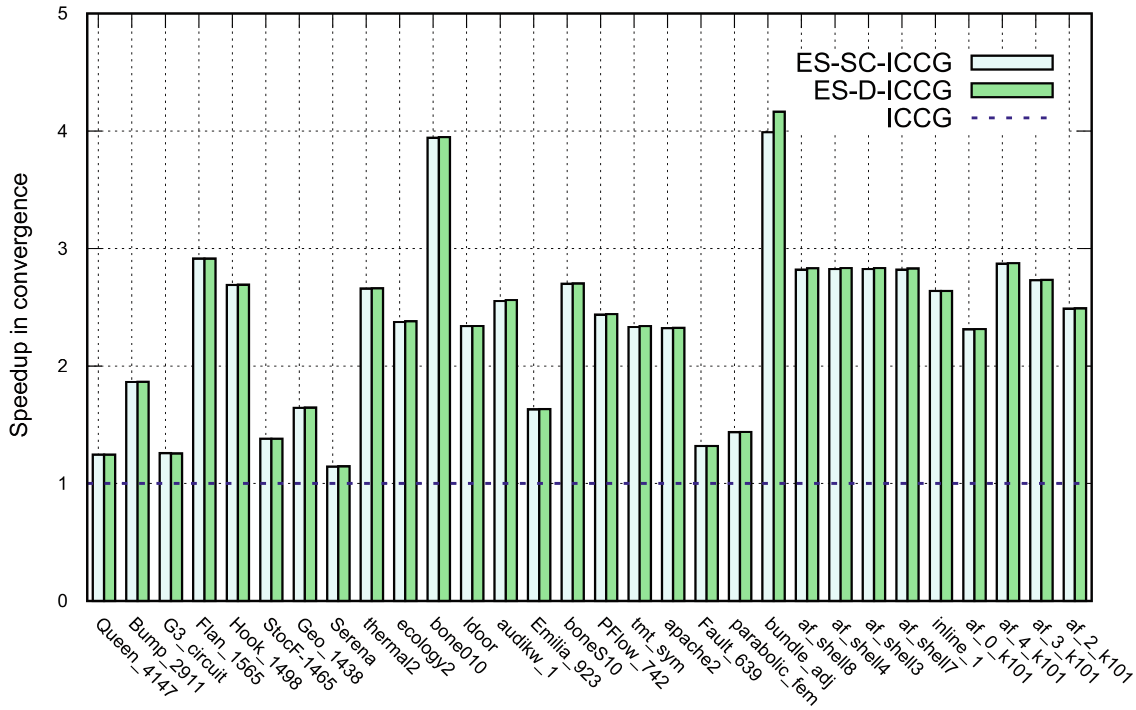

FIGURE 10 Speedup in convergence of ES-SC-ICCG and ES-D-ICCG over ICCG (parallel multithreaded solver, ***b***: random vector).

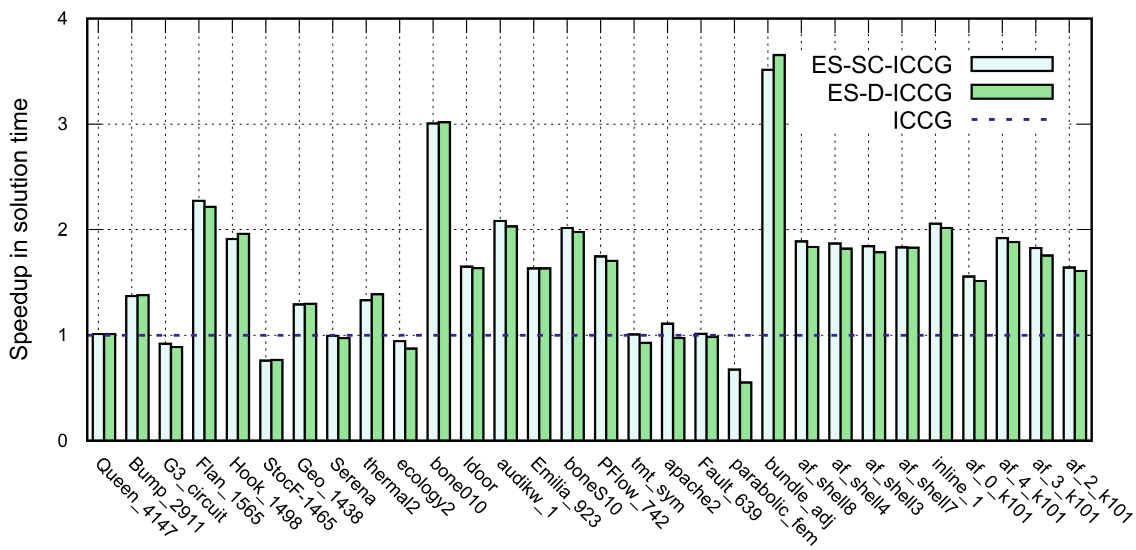

FIGURE 11 Speedup in computational time of ES-SC-ICCG and ES-D-ICCG over pICCG (parallel multithreaded solver, ***b***: random vector).

Figure 10 shows the speedup in convergence of the parallel solver based on the proposed technique against the parallel ICCG solver when random vectors are used for the right-hand side vectors. In the test using random vectors, the proposed method attained more than two-fold speedup in convergence compared with the parallel ICCG solver for 21 out of 30 datasets.

Next, we examine the computational time. In the test using a vector of ones, the proposed method reduced the solution time for 28 out of 30 datasets. For the bundle_adj dataset, the parallel deflated ICCG solver based on our technique attained a more than four-fold speedup compared with the parallel ICCG solver. The test using random vectors also indicated that our technique was effective for reducing the computational time for most test datasets (25 out of 30). In block Jacobi IC

TABLE 8 Condition number estimation based on error vector sampling.

	Estimation			LAPACK		
Dataset	λ_{max}	λ_{min}	κ	λ_{max}	λ_{min}	κ
bcsstk07	2.89	9.67×10^{-5}	2.99×10^{4}	2.90	9.11×10^{-5}	3.18×10^{4}
msc01440	3.62	3.08×10^{-4}	1.18×10^{4}	3.62	2.86×10^{-4}	1.27×10^{4}
494_bus	2.00	2.59×10^{-5}	7.71×10^{4}	2.00	2.53×10^{-5}	7.90×10^{4}
bcsstk06	2.89	9.67×10^{-5}	2.99×10^{4}	2.90	9.11×10^{-5}	3.18×10^{4}

preconditioning, the computational cost for a PCG iteration reduces as the number of threads increases. Consequently, the impact of the additional cost for convergence acceleration (SC preconditioning or deflation) on the preconditioned solver is substantially enlarged in the parallel execution by many threads. In other words, the ratio of the iteration costs is enlarged from (17). Because we used a number of threads (=40) in our numerical tests, it became difficult to reduce the solution time compared with the sequential solver. However, Figure 11 indicates that our convergence acceleration technique accelerated the solution process for most test problems.

5.4 | Condition number estimation

Figure 8 implies that our technique based on error vector sampling is a useful tool for the estimation of the smallest eigenvalue. Because the estimation of the largest eigenvalue is relatively easy, the technique can be used for the estimation of the condition number of the coefficient matrix. Algorithm 2 shows the proposed procedure of the PCG method with condition number estimation. The largest eigenvalue is estimated by the power method, which is combined with the procedure of CG method. We estimate the smallest eigenvalue using our technique. We conducted numerical tests using four relatively small matrices downloaded from the SuiteSparse matrix collection to examine our technique. Diagonal scaling was applied to the matrices before the tests. Table 8 shows the estimation for the largest eigenvalue $\lambda_{\max}$, smallest eigenvalue $\lambda_{\min}$, and condition number κ compared with the results calculated using the LAPACK library. Table 8 implies that the technique based on error sampling is effective for the estimation of the condition number. It is noted that when ES-SC and ES-D methods are used, the condition number (the smallest eigenvalue) cannot be estimated. This is because these techniques efficiently remove the error component involved in the eigenspace that corresponds to the smallest eigenvalue and the sampled error vectors might be orthogonal to the eigenvector corresponding to the smallest eigenvalue.

Because the number of sample vectors is much smaller than n ($m \ll n$), the additional computational cost for the calculation of the smallest eigenvalue (Ritz value) is typically much smaller than the iterative solution cost. Although the power method requires an additional SpMV operation, it is combined with SpMV for the CG method. In this case, the matrix data transferred from main memory are efficiently used for two vectors. The additional cost (time) for the power method, T_l, is estimated as

$$T_l = (16nN_{ite} + 20n + 12nnz)/b_m. \tag{24}$$

The cost for calculating the smallest eigenvalue, T_s, is equal to the auxiliary matrix setup cost except for the cost for (12) and is estimated as follows:

$$T_s = 2n(\tilde{m}^2 + m^2)/f + (12\tilde{m}n + 12\tilde{m}\cdot nnz)/b_m. \tag{25}$$

Finally, the additional cost for calculating the condition number estimation, $T_{\rm con}$, is given by

$$T_{\rm con} = T_l + T_s \tag{26}$$

$$= 2n(\tilde{m}^2 + m^2)/f + \{16nN_{ite} + 20n + 12\tilde{m}n + 12(\tilde{m}+1)nnz\}/b_m. \tag{27}$$

Algorithm 2. PCG method with condition number estimation

Input: $A, \boldsymbol{b}, M, \boldsymbol{x}_0, \varepsilon, m$

$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
$\boldsymbol{v}_0 \leftarrow$ Initialization (by a nonzero vector e.g., a random vector)
for $i = 1, 2, \dots$ **do**
  $\boldsymbol{z}_{i-1} = M^{-1}\boldsymbol{r}_{i-1}$
  $\rho_{i-1} = (\boldsymbol{r}_{i-1}, \boldsymbol{z}_{i-1})$
  if i=1 **then**
    $\boldsymbol{p}_1 = \boldsymbol{z}_0$
  else
    $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
    $\boldsymbol{p}_i = \boldsymbol{z}_{i-1} + \beta_{i-1}\boldsymbol{p}_{i-1}$
  end if
  $(\boldsymbol{q}_i\ \boldsymbol{v}_i) = A(\boldsymbol{p}_i\ \boldsymbol{v}_{i-1})$ // (SpMV)
  $\boldsymbol{v}_i = \boldsymbol{v}_i/\|\boldsymbol{v}_i\|_2$
  $\alpha_i = \rho_{i-1}/(\boldsymbol{p}_i, \boldsymbol{q}_i)$
  $\boldsymbol{x}_i = \boldsymbol{x}_{i-1} + \alpha_i\boldsymbol{p}_i$
  $\boldsymbol{r}_i = \boldsymbol{r}_{i-1} - \alpha_i\boldsymbol{q}_i$
  if $\|\boldsymbol{r}_i\|_2 \leq \varepsilon\|\boldsymbol{b}\|_2$ **then**
    break
  end if
  if Sampling condition is satisfied **then**
    $\tilde{\boldsymbol{x}}^{(s)} = \boldsymbol{x}_i,\ s \in \{1, 2, \dots, m\}$
  end if
end for
$\tilde{E} = (\boldsymbol{x}_i - \tilde{\boldsymbol{x}}^{(1)}\ \boldsymbol{x}_i - \tilde{\boldsymbol{x}}^{(2)}\ \dots\ \boldsymbol{x}_i - \tilde{\boldsymbol{x}}^{(m)})$
Apply the Gram–Schmidt method to $\tilde{E}$ and obtain E
Solve an eigenvalue problem: $E^{\top}AE\boldsymbol{t} = \lambda\boldsymbol{t}$ and obtain the smallest Ritz value $\lambda_{\min}$
$\lambda_{\max} = (\boldsymbol{v}_i, A\boldsymbol{v}_i)$
$\kappa = \lambda_{\max}/\lambda_{\min}$

Output: $\boldsymbol{x}_i, \kappa$

For a typical setting, $\tilde{m} = m = 20$, $nnz_{\mathrm{av}} = 30, f = 10b_m$, and $N_{ite} = 500$, the additional cost for estimating the condition number is equivalent to 6% of one ICCG solution step cost.

Most iterative solvers, such as the CG solver, typically use a convergence criterion based on a (relative) residual norm. If the estimation of the condition number of the coefficient matrix is given with the solution vector by the iterative solver, it can be a useful tool to evaluate the accuracy of the solution vector. The proposed solver provides this function without a large amount of additional computations. However, our technique is only useful for the case that the linear system equation must be solved. When we only calculate the condition number estimation, other methods, for example, Lanczos method may be a better choice.

6 | CONCLUSION

In this article, we introduced an algebraic auxiliary matrix construction method that can be used for the SC preconditioning and the deflation method. We focused on the problem in which a sequence of linear systems with identical coefficient matrices is solved. In our method, we sample the approximate solution vectors in the first iterative solution step, and calculate the error vectors corresponding to the sample vectors after the solution step is completed. Then, we perform the Rayleigh-Ritz method using a subspace spanned by these error vectors to identify (approximate) eigenvectors associated with small eigenvalues. Finally, we construct the auxiliary matrix using the Ritz vectors associated with small Ritz values. We also presented a cost model of SC preconditioning and the deflation method. Numerical tests using 30 coefficient matrices were conducted to verify our technique. The test results confirmed that the proposed convergence acceleration technique efficiently reduced both the number of iterations for convergence and the solution time of the serial and parallel preconditioned CG solvers. Moreover, additional numerical tests indicated that the proposed technique can be used for condition number estimation.

Currently, we are examining the effectiveness of the technique for a linear system that has an unsymmetric coefficient matrix. Because the preliminary results demonstrate its effectiveness, we will report it in the future. We are also investigating the application of the technique to other problems. Particularly, we are examining its effectiveness in parallel-in-time simulations, which often involve the solution process of multiple linear systems of coefficient matrices with a common property. We are also interested in the combination of our technique with the AMG method or preconditioning techniques suitable for GPU computing. In the future, we will examine our technique in various scenarios of computational science or engineering problems.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their helpful comments. This work was supported by JSPS KAKENHI Grant Numbers JP19H04122, JP19H05662, JP20K21782, and JP23H00462.

CONFLICT OF INTEREST STATEMENT

This study does not have any conflicts to disclose.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

ENDNOTES

[1]In this article, we use the term "SC preconditioning," which appears in references.[34,35] Preconditioning based on the same concept is often called two-level preconditioning, or spectral preconditioning,[36] particularly when the subspace is associated with eigenspaces.

[2]In this article, we describe the method to identify eigenvectors with relatively small eigenvalues of the coefficient matrix. However, it is possible to consider identifying eigenvectors with relatively small eigenvalues of the preconditioned matrix. In this case, we should use the preconditioned matrix instead of A in (10).

ORCID

Takeshi Iwashita https://orcid.org/0000-0003-1938-1723

REFERENCES

1. Xu J. Iterative methods by space decomposition and subspace correction. SIAM Rev. 1992;34:581–613.
2. Nicolaides RA. Deflation of conjugate gradients with applications to boundary value problems. SIAM J Numer Anal. 1987;24(2):355–65.
3. Trottenberg U, Oosterlee CW, Schüller A. Multigrid. San Diego, CA: Elsevier; 2001.
4. Wesseling P. Multigrid algorithms. An introduction to multigrid methods. Hoboken, NJ: John Wiley & Sons Ltd; 1992 Corrected Reprint., R. T. Edwards, Inc., 2004.
5. Vuik C, Segal A, Meijerink JA. An efficient preconditioned CG method for the solution of a class of layered problems with extreme contrasts in the coefficients. J Comput Phys. 1999;152:385–403.
6. Vuik C, Frank J. Deflated ICCG method applied to problems with extreme contrasts in the coefficients. Proceedings of the 16th IMACS World Congress; 2000.
7. De Gersem H, Hameyer K. A deflated iterative solver for magnetostatic finite element models with large differences in permeability. Eur Phys J Appl Phys. 2001;13:45–9.
8. Mifune T, Moriguchi S, Iwashita T, Shimasaki M. Convergence acceleration of iterative solvers for the finite element analysis using the implicit and explicit error correction methods. IEEE Trans Magn. 2009;45(3):1438–41.
9. Igarashi H, Watanabe K. Deflation techniques for computational electromagnetism: theoretical considerations. IEEE Trans Magn. 2011;47(5):1438–41.
10. Iwashita T, Kawaguchi S, Mifune T, Matsuo T. Automatic mapping operator construction for subspace correction method to solve a series of linear systems. JSIAM Lett. 2017;9:25–8.
11. Kharchenko SA, Yeremin AY. Eigenvalue translation based preconditioners for the GMRES (k) method. Numer Linear Algebra Appl. 1995;2(1):51–77.
12. Morgan RB. A restarted GMRES method augmented with eigenvectors. SIAM J Matrix Anal Appl. 1995;16(4):1154–71.
13. Erhel J, Burrage K, Pohl B. Restarted GMRES preconditioned by deflation. J Comput Appl Math. 1996;69(2):303–18.
14. Morgan RB. GMRES with deflated restarting. SIAM J Sci Comput. 2002;24(1):20–37.

15. Morgan RB, Wilcox W. Deflated iterative methods for linear equations with multiple right-hand sides. arXiv preprint arXiv:math-ph/0405053. 2004.
16. Giraud L, Gratton S, Pinel X, Vasseur X. Flexible GMRES with deflated restarting. SIAM J Sci Comput. 2010;32(4):1858–78.
17. Carpenter MH, Vuik C, Lucas P, van Gijzen M, Bijl H. A general algorithm for reusing Krylov subspace information. I. Unsteady Navier-Stokes. Hampton, VA: Langley Research Center; 2010.
18. Saad Y, Yeung M, Erhel J, Guyomarc'h F. A deflated version of the conjugate gradient algorithm. SIAM J Sci Comput. 2000;21(5): 1909–26.
19. Abdel-Rehim AM, Morgan RB, Nicely DA, Wilcox W. Deflated and restarted symmetric Lanczos methods for eigenvalues and linear equations with multiple right-hand sides. SIAM J Sci Comput. 2010;32(1):129–49.
20. Kilmer ME, De Sturler E. Recycling subspace information for diffuse optical tomography. SIAM J Sci Comput. 2006;27(6): 2140–66.
21. Gosselet P, Rey C, Pebrel J. Total and selective reuse of Krylov subspaces for the resolution of sequences of nonlinear structural problems. Int J Numer Methods Eng. 2013;94:60–83.
22. Daas HA, Grigori L, Hénon P, Ricoux P. Recycling Krylov subspaces and truncating deflation subspaces for solving sequence of linear systems. ACM Trans Math Softw. 2021;47(2):1–30.
23. Morgan RB. Restarted block-GMRES with deflation of eigenvalues. Appl Numer Math. 2005;54(2):222–36.
24. Soodhalter KM, de Sturler E, Kilmer ME. A survey of subspace recycling iterative methods. GAMM Mitt. 2020;43(4):e202000016.
25. Brezina M, Falgout R, MacLachlan S, Manteuffel T, McCormick S, Ruge J. Adaptive algebraic multigrid. SIAM J Sci Comput. 2006;27(4):1261–86.
26. Brandt A, Brannick J, Kahl K, Livshits I. Bootstrap AMG. SIAM J Sci Comput. 2011;33(2):612–32.
27. Nomura N, Fujii A, Tanaka T, Nakajima K, Marques O. Performance analysis of SA-AMG method by setting extracted near-kernel vectors. In: Dutra I, Camacho R, Barbosa J, Marques O, editors. Proceedings of the International Conference on Vector and Parallel Processing. Cham: Springer; 2017. p. 52–63.
28. D'ambra P, Filippone S, Vassilevski PS. BootCMatch: a software package for bootstrap AMG based on graph weighted matching. ACM Trans Math Softw. 2018;44:1–25.
29. D'Ambra P, Vassilevski PS. Improving solve time of aggregation-based adaptive AMG. Numer Linear Algebra Appl. 2019;26(6): e2269.
30. Baker AH, Jessup ER, Manteuffel T. A technique for accelerating the convergence of restarted GMRES. SIAM J Matrix Anal Appl. 2005;26(4):962–84.
31. Imakura A, Li RC, Zhang SL. Locally optimal and heavy ball GMRES methods. Jpn J Ind Appl Math. 2016;33:471–99.
32. Davis TA, Hu Y. The university of Florida sparse matrix collection. ACM Trans Math Softw. 2011;38:1–25.
33. Iwashita T, Kawaguchi S, Mifune T, Matsuo T. Acceleration of transient non-linear electromagnetic field analyses using an automated subspace correction method. IEEE Trans Magn. 2019;55(6):1–4.
34. Mihajlovic MD, Mijalkovic S. A component decomposition preconditioning for 3D stress analysis problems. Numer Linear Algebra Appl. 2002;9:567–83.
35. Ovtchinnikov EE, Xanthis LS. The discrete Korn's type inequality in subspaces and iterative methods for thin elastic structures. Comput Methods Appl Mech Eng. 1998;160:23–37.
36. Carpentieri B, Giraud L, Gratton S. Additive and multiplicative two-level spectral preconditioning for general linear systems. SIAM J Sci Comput. 2007;29(4):1593–612.
37. Zhao T. A spectral analysis of subspace enhanced preconditioners. J Sci Comput. 2016;66(1):435–57.
38. Saad Y. Iterative methods for sparse linear systems. 2nd ed. Philadelphia, PA: SIAM; 2003.

How to cite this article: Iwashita T, Ikehara K, Fukaya T, Mifune T. Convergence acceleration of preconditioned conjugate gradient solver based on error vector sampling for a sequence of linear systems. Numer Linear Algebra Appl. 2023;30(6):e2512. https://doi.org/10.1002/nla.2512

APPENDIX A. SELECTION METHOD FOR APPROXIMATION VECTORS

Algorithm 3 shows sampling method A for the approximate solution vector.[33] In the algorithm, i is the iteration count and m is the number of sample vectors. We set parameter $l_{\max}$ to satisfy $m^{l_{\max}} > N_{\max}$, where $N_{\max}$ is the preset maximum iteration count of the solver.

Algorithm 3. Selection of approximate solution vectors

$h = 1$
for $i = 1, 2, \ldots$ **do**
  Solver part
  Convergence check
  if $(\mathrm{mod}(i, h) == 0)$ **then**
    $i_t = \sum_{l=0}^{l_{\max}} (-1)^l \lfloor (i-1)/m^l \rfloor$
    $s = \mathrm{mod}(i_t, m) + 1$
    $\tilde{\boldsymbol{x}}^{(s)} = \tilde{\boldsymbol{x}}_i$
    if $(i == h * m)$ **then**
      $h = h * 2$
    end if
  end if
end for

APPENDIX B. CONSTANTS IN THE COST MODEL OF THE ICCG AND SC-ICCG SOLVERS

In this appendix, we describe some details of the cost models of the ICCG and SC-ICCG solvers, which are based on the amount of data transferred from main memory. In the solvers, we use 32-bit integers and double precision (64-bit) floating-point arithmetic and data.

B.1 CG method

The main kernel of the CG solver is a SpMV kernel. In this study, we implement the SpMV kernel using the standard compressed sparse row format. In this implementation, the data transferred for the coefficient matrix consist of an integer array of size n for the row pointers, an integer array of size nnz for the column indices, and a floating-point array of size nnz for the matrix element values. Consequently, $4n + 12nnz$ Bytes of data are transferred from memory for the coefficient matrix. Regarding the amount of data transferred for the source vector, we use an optimistic estimation. Namely, we assume full utilization of the cache memory for the vector. Accordingly, each element of the source vector is loaded from main memory only once. Moreover, each element of the resultant vector is stored in main memory. Consequently, the amount of data transferred for the source and resultant vectors is estimated to be $16n$ Bytes. In total, the amount of data transferred for the SpMV is estimated to be $20n + 12nnz$.

We estimate the amount of data transferred for other parts of the CG solver that consist of inner products and vector updates as $56n$, though it depends on the implementation. Accordingly, the cost for one CG iteration is given by $76n + 12nnz$ Bytes.

B.2 IC preconditioning

The additional cost for IC preconditioning in each iteration is given by the forward and backward substitutions. When we use IC(0) preconditioning, the number of nonzero elements is the same as that of the coefficient matrix. Accordingly, the amount of data transferred for the preconditioning matrix in the substitutions is almost the same as that for SpMV. However, the kernel of the substitutions needs one more row-pointer array, because the lower and upper triangular matrices are separately processed. Considering this factor ($4n$ Bytes), the cost of an IC preconditioning step is estimated to be $24n + 12nnz$. Consequently, the cost for one ICCG iteration is given by $100n + 24nnz$ Bytes.

B.3 SC preconditioning

The dominant part of the computational cost of an SC preconditioning step is given by two dense matrix vector products using $\boldsymbol{W}$ and $\boldsymbol{W}^\top$. Because the size of $\boldsymbol{W}$ is $n \times \tilde{m}$, the amount of data transferred for these matrices is given by $16\tilde{m}n$ Bytes. Considering the data needed for the source and resultant vectors, the cost for an SC preconditioning step is estimated to be $16n + 16\tilde{m}n$. Consequently, the cost of an SC-ICCG iteration is given by $116n + 16\tilde{m}n + 24nnz$ Bytes.