A parallel generator of non-Hermitian matrices computed from given spectra

Xinzhe Wu, Serge Petiton, Yutong Lu

Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

SPECIAL ISSUE PAPER

A parallel generator of non-Hermitian matrices computed from given spectra

Xinzhe Wu*[1,2] | Serge G. Petiton[1,2] | Yutong Lu[3]

[1]Maison de la Simulation, CNRS, Gif-sur-Yvette, France
[2]CRIStAL, UMR CNRS 9189, University of Lille, Villeneuve d'Ascq, France
[3]National Supercomputing Center in Guangzhou, Sun Yat-sen University, Guangzhou, China

Correspondence
*Xinzhe Wu, Maison de la Simulation, CNRS, F-91191 Gif-sur-Yvette, France.
Email: xinzhe.wu.etu@univ-lille.fr

Summary

Iterative linear algebra methods to solve linear systems and eigenvalue problems with non-Hermitian matrices are important for both the simulation arising from diverse scientific fields and the applications related to big data, machine learning and artificial intelligence. The spectral property of these matrices has impacts on the convergence of these solvers. Moreover, with the increase of the size of applications, iterative methods are implemented in parallel on clusters. Analysis of their behaviors with non-Hermitian matrices on supercomputers is so complex that we need to generate large-scale matrices with different given spectra for benchmarking. These test matrices should be non-Hermitian and non-trivial, with high dimension. This paper highlights a scalable matrix generator that constructs large sparse matrices using the user-defined spectrum, and the eigenvalues of generated matrices are ensured to be the same as the predefined spectrum. This generator is implemented on CPUs and multi-GPUs platforms, with good strong and weak scaling performance on several supercomputers. We also propose a method to verify its ability to guarantee the given spectra. Finally, we give an example to evaluate the numerical properties and parallel performance of iterative methods using this matrix generator.

KEYWORDS:
Non-Hermitian Matrix, Matrix Generation, Spectrum, Iterative Methods, Parallel Computing, Linear System and Eigenvalue Problem

1 | INTRODUCTION

Linear system can be defined as finding $x \in \mathbb{C}^n$ with a coefficient matrix $A \in \mathbb{C}^{n\times n}$ and the Right-hand Side $b \in \mathbb{C}^n$ such that

$$Ax = b.$$

Eigenvalue problem can be defined as finding some pairs (λ, u) with $\lambda \in \mathbb{C}$ and $u \in \mathbb{C}^n$ of matrix $A \in \mathbb{C}^{n\times n}$ such that

$$Au = \lambda u.$$

In this formula, λ is an *eigenvalue* of A, and u is its corresponding *eigenvector*, λ may be complex even if A is real. The *spectrum* of A is the set of all eigenvalues of A, denoted it as $\lambda(A)$.

Eigenvalue and linear system problems occur in many areas of science and engineering, such as structural analysis[1,2], wave modes simulations[3,4] and electromagnetic applications[5,6]. Eigenvalues are also important in analysing numerical methods. In

machine learning and pattern recognition, both supervised and unsupervised learning algorithms, such as principal component analysis (PCA), Fisher discriminant analysis (FDA), and clustering, often require solving eigenvalue problems. Iterative methods are most commonly used solvers for large-scale eigenvalue and linear systems problems. An insufficient accuracy and a failure of these solvers usually result in, respectively, a poor approximation to original problems and a failure of entire algorithms. A good selection of eigenvalue and linear system solvers becomes essential. Researchers would benefit from large-scale test matrices with different spectral properties to benchmark the numerical performance and parallel efficiency of these methods.

In this paper, we present a Scalable Matrix Generator from Given Spectra (SMG2S) to benchmark the linear and eigenvalue solvers on large-scale platforms. This paper is organized as follows: Section 2 talks about the related work on providing test matrix collections and the motivation to propose a matrix generator with given spectra. Section 3 gives an overview of the mathematical framework and numerical algorithm for matrix generation. The parallel implementation of SMG2S on CPUs and multi-GPUs is introduced in Section 4. Parallel performance of SMG2S on several different supercomputers is also evaluated in this section. In Section 5, an accuracy verification method to check the ability of SMG2S to keep the given spectra is proposed based on Shifted Inverse Power method. Experimental results with different spectral distributions for the accuracy verification are also presented in Section 5. In Section 6, we give an example which uses SMG2S to evaluate several iterative solvers to solve linear systems. Finally, we conclude in Section 7.

2 | RELATED WORK AND MOTIVATION

In this section, we introduce our motivation to provide a matrix generator with given spectra by summarizing the relationship between the spectral information of the operator matrix and iterative solvers. The existing test matrices providers are also presented.

2.1 | Spectral Information and Iterative Solvers

Iterative methods approach the solution x of a linear system $Ax = b$ from an initial guess solution x_0 by a number of iterative steps. Compared with the direct methods such as LU and Gauss Jordan which need an overall computational cost of the order $\frac{2}{3}n^3$, the cost of iterative methods is the order of n^2 operations for each iteration. Iterative methods are especially competitive with direct methods in the case of large sparse matrices, to avoid the potential introduction of the dramatic fill-in by the direct methods. Krylov subspace methods are a collection of iterative solvers to solve both linear system and eigenvalue problems.

In linear algebra, the m-order Krylov subspace[7] generated by a $n \times n$ matrix $A \in \mathbb{C}^{n\times n}$ and a vector $b \in \mathbb{C}^n$ is the linear subspace spanned by the images of b under the first m powers of A, that is

$$K_m(A, b) = span(b, Ab, A^2b, \cdots, A^{m-1}b).$$

The Krylov subspace provides the ability to extract the approximations from an m-dimensional subspace K_m. These iterative methods are often restarted after a number of iterations, to avoid enormous memory and computational requirements with the increase of Krylov subspace projection number[7]. For the linear solvers, a temporary solution $\tilde{x}$ is used as a new initial guess vector, and the projection procedure is restarted with this new guess vector. For the eigenvalue solvers, they are often restarted with a linear combination of known eigenvectors. In this section, we give an introduction about the impacts of eigenvalues on the numerical performance of iterative methods.

2.1.1 | Impact of Spectral Information on Convergence of Iterative Linear Solvers

The spectral property of the coefficient matrices is one of the important factors which have impacts on the convergence of iterative methods to solve linear systems. In this section, we summarize the recent efforts to explain this relationship, which is the initial motivation for us to provide the test matrix generator with given spectra. More details about the discussions and demonstrations can be found in the book of Liesen et al.[8].

In general, for the linear systems whose coefficient matrices are *Hermitian Indefinite* or *general (quasi-)normal*, the relation between the spectra and convergence are clear, which can be summarized as follows:

(1) for Hermitian Indefinite matrix, whose eigenvalues are all real, or for general (quasi-)normal matrix, whose eigenvalues might be distributed in the real-imaginary plane, their eigenvalues close to the origin point damage the convergence rate

of iterative methods. The eigenvalues with too small Euclidean norm make the solvers difficult to obtain the convergence. If it exists an eigenvalue $\lambda \approx 0$, the iterative methods might not be able to achieve the convergence.

(2) the dominant eigenvalues of coefficient matrices with larger Euclidean norm can accelerate the convergence of iterative methods. The dominant eigenvalues are defined as the ones far from the origin point in the real-imaginary plane. In practice, an ellipse is refined in the real-imaginary plane by the dominant eigenvalues. If this refined ellipse has relatively small focal distance and length of major semi axis, but larger distance to the origin point in the real-imaginary plane, iterative methods can converge more quickly. This can be translated as the clustering of dominant eigenvalues stimulating the convergence of iterative solvers.

(3) the condition number of the coefficient matrix can also be used to qualify the convergence of iterative methods. The convergence of iterative methods for linear systems with larger condition number is slow compared with the ones with smaller condition number.

For the linear systems whose coefficient matrices are *non-normal*, the relation between their spectral properties and convergence is far more complex. There are not yet conclusions to describe this relation. However, in many practical applications, one can still observe a correlation between the eigenvalues of a general non-normal matrix A and the convergence rate of iterative linear solvers[8]. In general, linking the eigenvalues of coefficient matrices and convergence of iterative methods must always be based on additional information and arguments. They can consist of, e.g., demonstrating a special relationship between the initial residual vector (or the Right-hand Side if initial guess solution $x_0 = 0$) and the eigenvectors of coefficient matrices. Until now, several techniques are tried to be used to explain this connection for non-normal coefficient matrices, e.g., ϵ-pseudospectra[9], the field of values[10] or the polynomial numerical hull[11] are applicable for some special cases. Further efforts are necessary for the researchers to discover this link by considering other parameters. The existing of a matrix generator which can generate test matrices with users' prescribed spectra can always guide the further research.

2.1.2 | Preconditioning Techniques for Linear System and Eigenvalue Solvers

The spectral properties are important, not only for understanding the convergence behaviors of linear systems but also for constructing different preconditioners to speed up the solution.

The first kind of preconditioners is to use the preconditioning matrix M, which can be defined in many different ways. The purpose is to make it much easier to solve a newly constructed linear systems $Mx = b$ compared with the original one $Ax = b$. After the selection of M, there are three ways to apply this preconditioning matrix M to the original systems: the left, right and split preconditioning[12]. This type of preconditioning technique is generally more used for the linear systems with normal and quasi-normal matrices. They are capable of accelerating the convergence of iterative methods by changing the spectral distribution of coefficient matrices with different condition number or clustering properties, etc.

As presented in Section 2.1.1, the eigenvalues near the origin point damage the convergence of iterative methods. Thus the deflation preconditioners are constructed by removing or deflating the smallest eigenvalues, which might improve the convergence performance of solvers. If the dimension of Krylov subspace is large enough, some deflation occurs automatically[13]. But for the restarted iterative methods, the limitation of the size of Krylov subspace is not enough for these deflations, and convergence cannot be achieved accordingly. Thus deflation schemes should be constructed for each cycle of the restart.

In order to approximate a solution of linear system $Ax = b$, one approach is to get the inverse of A, denote it as A^{-1}, and then an approximate solution can be easily obtained as $\tilde{x} = A^{-1}b$. $\tilde{x}$ can be applied as a new residual vector for the subsequent restart of iterative methods, that is the polynomial preconditioning for linear solvers. In details, each cycle of restarted iterative methods can construct a Hessenberg matrix H_m by the Arnoldi reduction. Then a selected number of dominant eigenvalues are approximated from H_m. The preconditioning step is to get the best polynomial p_k using a domain Ω_k constructed by these eigenvalues to generate a new $\tilde{x}$ for the next time restart. The idea of polynomial preconditioners for linear solvers is to enlarge the Euclidean norms of dominant eigenvalues, which might stimulate the convergence.

The idea of polynomial preconditioning for the iterative solvers of eigenvalue problems is similar to the one for linear systems. In details, a selected polynomial p_k can construct by the r unwanted eigenvalues and new gotten ones from previous restart cycle, and the operator A can be replaced by $B_k = p_k(A)$. This polynomial can amplify the wanted eigenvalues of A outside Ω_k into the ones of B_k with much larger norm compared with the remaining eigenvalues, which will get much faster convergence. The eigenvalues of A can be obtained from the ones of B_k by a Galerkin projection. The polynomial p_k can be formulated either by

the Chebyshev basis with the best ellipse constructed by the unwanted eigenvalues[14], or by the refinement of a polygon with these unwanted eigenvalues[15].

Eigenvalues are critical information for researchers to construct the preconditioners. The proposition of matrix generator with given spectra can facilitate the researchers to propose and analyze their preconditioning techniques according to their applications.

2.2 | Existing Test Matrix Providers

There are already several efforts to supply test matrix collections. SPARSEKIT[16] implemented by Y. Saad contains various simple matrix generation subroutines. The Galeri package of Trilinos provides to generate simple well-known finite element matrices in parallel. Z. Bai[17] presented a collection of test matrices for developing numerical algorithms for solving non-symmetric eigenvalue problems. There are also several widely spread matrix providers, the Hawell Boeing sparse matrix collection[18], the Tim Davis[19] and Matrix Market collections[20]. They both contain many matrices from scientific fields with various mathematical characteristics. But the spectra of matrices in these collections are fixed, and cannot be customized. M.T. Chu[21] provides an overview of the inverse eigenvalues problems concerning the reconstruction of a structured matrix from prescribed spectrum without parallel implementation. Three subroutines xLATMR, xLATMS and xLATME were introduced by J. Demmel[22] in 1989 to benchmark the routines of dense matrices in LAPACK (Linear Algebra PACKage). xLATMR generates a random matrix with off-diagonal entries. It is the simplest and fastest routines in this suite, and permits no direct control over the eigenvalues of the generated matrices[22]. xLATMS generates random real symmetric and complex Hermitian matrices with given eigenvalues and bandwidth, or a random non-symmetric or complex symmetric matrix with given singular values and upper and lower bandwidth[22]. Only xLATME generates a random non-symmetric square matrix with specified eigenvalues[22]. It is able to generate a dense matrix with prescribed eigenvalues, and then reduce it into band with user-defined upper and lower bandwidth by a series of Householder transformation operations.

2.3 | Motivation to Propose a Parallel Matrix Generator with Given Spectra

Nowadays, the size of eigenvalue/linear system problems and the supercomputer systems continues to scale up. The whole ecosystem of High Performance Computing (HPC), including the linear algebra applications, should adjust to larger computing platforms. Under this background, there are five special requirements for the test matrices to evaluate the numerical algorithms:

(1) their spectra must be known and can be customized;

(2) sparse, non-Hermitian and non-trivial;

(3) a very high dimension, including the non-zero element numbers and/or the matrix dimension to evaluate the algorithms on large-scale systems, which means that the proposed matrix generator should be able to be parallelized to profit from the large-scale distributed memory supercomputers;

(4) the controllable sparsity patterns.

Due to the importance of spectral information on the eigenvalue/linear system solvers and some of their preconditioners, the test matrices with customized spectra can help to analyze and provide numerically robust solvers. Besides, some scientific communities may be interested in matrices with clustered, conjugated eigenvalues or other special spectral distributions. It is essential to develop a large set of non-Hermitian test matrices whose eigenvalues can be customized. For the test matrices, the properties of being sparse, non-Hermitian and non-trivial together can add many mathematical features. Good parallelization of proposed generation method makes it easy for the experiments with high dimensional matrices on large-scale platforms.

As far as we know, only xLATME is able to generate non-symmetric matrices with given eigenvalues. However, it is not suitable and efficient to test the solvers for the sparse matrix, since it requires $\mathcal{O}(n^3)$ time and $\mathcal{O}(n^2)$ storage even for generating a small bandwidth matrix. Moreover, it was implemented for the shared memory systems rather than larger distributed memory systems, thus it is difficult to generate large-scale test matrices targeting for extreme-scale clusters. This is the motivation for us to propose SMG2S which can generate large-scale non-Hermitian matrices with given spectra in parallel. It requires much less time and storage, and can be easily parallelized on modern distributed memory systems.

3 | MATRIX GENERATION ALGORITHM

In this section, we introduce the proposed matrix generation algorithm. First of all, in Section 3.1, we present a summary of its mathematical framework based on the preliminary theorem proposed by H. Gachlier et al.[23].

3.1 | Matrix Generation Framework

Theorem 1. Let's consider a collection of matrices $M(t) \in \mathbb{C}^{n\times n}$, $n \in \mathbb{N}^*$. If $M(t)$ verifies:

$$\begin{cases} \dfrac{dM(t)}{dt} = AM(t) - M(t)A, \\ M(t=0) = M_0. \end{cases}$$

Then $M(t)$ and M_0 are similar. $M(t)$ has the same eigenvalues as M_0.

Proof. Denote respectively $\sigma(M_0)$ and $\sigma(M_t)$ the spectra of M_0 and M_t. If M_0 is a diagonalisable matrix, $\forall\lambda \in \sigma(M_0)$, it exists an eigenvector $v \neq 0$ satisfies the relation:

$$M_0 v = \lambda v. \tag{1}$$

Denote $v(t)$ by the matrix $B \in I_n$:

$$v(t) = B_t v = e^{tA} B v. \tag{2}$$

We can get:

$$\begin{aligned} \frac{d(M_t v(t) - \lambda v(t))}{dt} &= \frac{dM_t}{dt} v(t) + M_t \frac{dv(t)}{dt} - \lambda \frac{dv(t)}{dt} \\ &= A(M_t v(t) - \lambda v(t)) + \lambda A v(t) \\ &\quad - M_t A v(t) + M_t \frac{dB_t}{dt} v - \lambda \frac{dB_t}{dt} v. \end{aligned} \tag{3}$$

With the definition of B_t in Equation (2), we have:

$$\frac{dB_t}{dt} = AB_t. \tag{4}$$

Thus the Equation (3) can be simplified as

$$\frac{d(M_t v(t) - \lambda v(t))}{dt} = A(M_t v(t) - \lambda v(t)). \tag{5}$$

The initial condition for the Equation (5) is:

$$M_t v(t) - \lambda v(t)|_{t=0} = M_0 B v - \lambda B v = M_0 v - \lambda v = 0. \tag{6}$$

Hence the solution of the differential Equation (5) is 0 and $\forall\lambda \in \sigma(M_0)$, we have $\lambda \in \sigma(M_t)$. Since $dim(M_0) = dim(M_t)$, we have $\sigma(M_0) = \sigma(M_t)$. Thus, M_0 and M_t are similar with same eigenvalues, but different eigenvectors.

□

3.2 | Matrix Generation Method

The proposed generation method can transfer a matrix M_0 with given spectra to another one $M(t)$ that satisfies *Theorem 1* and keeps the spectra of M_0. The idea may seem simple, but many parameters need to be determined to achieve our objective.

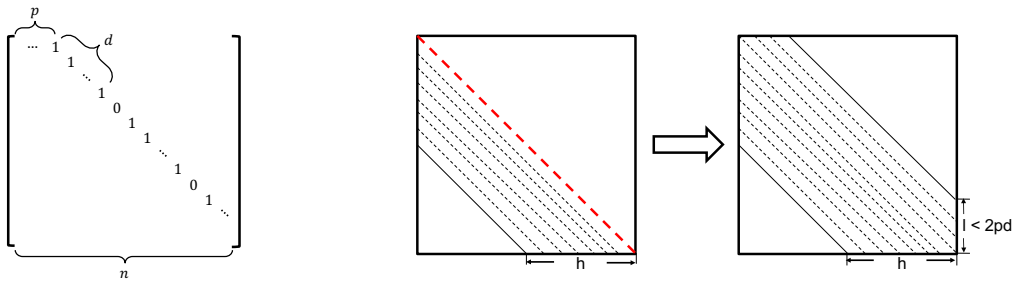
Denote a linear operator of matrix M determined by matrix A as $\widetilde{A_A} = AM - MA$, $\forall A \in \mathbb{C}^{n\times n}$, $M \in \mathbb{C}^{n\times n}$, $n \in \mathbb{N}^*$. Here AM and MA are the matrix-matrix multiplication operation of matrices A and M. By solving the differential equation in *Theorem 1*, we can firstly get the formula of $M(t)$ with the exponential operator and then extend it by the *Taylor series formula*:

$$\begin{cases} M(t) = e^{\widetilde{A_A}(M_0)t}, \\ M(t) = \displaystyle\sum_{k=0}^{\infty} \frac{t^k}{k!} (\widetilde{A_A})^k (M_0). \end{cases} \tag{7}$$

Through the loop $M_{i+1} = M_i + \frac{1}{i!}(\widetilde{A_A})^i(M_0), i \in (0, +\infty)$, a very simple initial matrix $M_0 \in \mathbb{C}^{n\times n}$ can be converted into a new sparse, non-trivial and non-Hermitian matrix $M_{+\infty} \in \mathbb{C}^{n\times n}$, which has the same spectra but different eigenvectors with M_0.

A good selection of matrix A can make $\widetilde{(A_A)}^i$ tend to $\mathbf{0}$ in limited steps. In this paper, A is defined as a nilpotent matrix, which means that there exists an integer k such that: $A^i = 0$ for all $i \geq k$. Such k is called the nilpotency of A. The selection of A will influence the sparsity pattern of the upper band of the generated matrix.

The exact shape of A is given in Fig. 1a. Inside a $n \times n$ matrix A, its entries default to 0, except for the upper diagonal of the distance p from the diagonal (this matrix is sparse, thus only the entries with non-zero values will be allocated in memory for the practical implementation). In this diagonal, its entries begin with d continuous 1 and a 0, this pattern is repeated until the end. Matrix A should be nilpotent with good choices of the parameters p, d and n. The determination of this series of matrices to be nilpotent might be difficult, but the cases that $p = 1$ or $p = 2$ are straightforward, which can completely fulfill our demands.



(a) Nilpotent matrix **(b)** Matrix Generation

FIGURE 1 (a) gives the nilpotent Matrix, with p off-diagonal offset, d number of continuous 1, and n matrix dimension; (b) shows the matrix generation example.

If $p = 1$, with $d \in \mathbb{N}^*$, or $p = 2$ with $d \in \mathbb{N}^*$ to be even, the nilpotency of A and the upper band's bandwidth of generated matrix are respectively $d + 1$ and $2pd$. Obviously, there is another constraint that the matrix size n should be greater or equal to the upper band's width $2pd$. For $p = 2$, if d is odd, the matrix A will not be nilpotent, thus we do not take it into account.

3.3 | Matrix Generation Algorithm

As shown in Algorithm 1, the procedure of SMG2S is simple. Firstly, it reads an array $Spec_{in} \in \mathbb{C}^n$, as the given eigenvalues. Then it inserts the elements in h lower diagonals of M_0 according to p and d, and sets its diagonal by $Spec_{in}$, and scales it with $(2d)!$. Meanwhile, it generates a nilpotent matrix A according to d, p and n. The final matrix M_t can be generated as $M_t = \frac{1}{(2d)!}M_{2d}$, where M_{2d} is the result after $2d$ times of loop $M_{i+1} = M_i + (\prod_{k=i+1}^{2d} k)(\widetilde{A_A})^i(M_0)$. The slight modification of the loop formula is to reduce the potential rounding errors coming from numerous division operations on computer systems.

Algorithm 1 Matrix Generation Method

function MATGEN($input$:$Spec_{in} \in \mathbb{C}^n$, p, d, h, $output$: $M_t \in \mathbb{C}^{n\times n}$)
  Insert the entries in h lower diagonals of $M_o \in \mathbb{C}^{n\times n}$
  Insert $Spec_{in}$ on the diagonal of M_0 and $M_0 = (2d)!M_0$
  Generate nilpotent matrix $A \in \mathbb{C}^{n\times n}$ with selected parameters d and p
  for $i = 0, \cdots, 2d - 1$ **do**
    $M_{i+1} = M_i + (\prod_{k=i+1}^{2d} k)(\widetilde{A_A})^i(M_0)$
  end for
  $M_t = \frac{1}{(2d)!}M_{2d}$
end function

If M_0 is a lower triangular matrix with h non zero diagonals, M_t will be a band matrix, whose bandwidth of upper triangular will be at most $2pd - 1$. Thus the maximal bandwidth of M_t is $h + 2pd - 1$, as Fig. 1b. As shown in Fig. 2, the lower diagonals of M_0 can set to be sparse, which ensures the sparsity of M_t for evaluating sparse iterative solvers. Moreover, permutation matrices can also be applied to change its sparsity. The operation complexity of SMG2S is $\max(\mathcal{O}(hdn), \mathcal{O}(d^2n))$. The worst case with large d and h would require $\mathcal{O}(n^3)$ operations and $\mathcal{O}(n^2)$ memory. However, if the required bandwidth of generated

matrix is small, or if the lower band of M_0 is sparse, it becomes an $\mathcal{O}(n)$ problem with good potential scalability and requires $\mathcal{O}(n)$ memory.

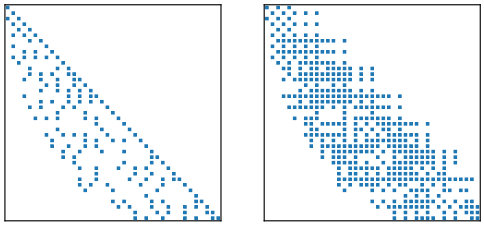


FIGURE 2 Matrix Generation Sparsity Pattern Example.

4 | PARALLEL IMPLEMENTATION

In this section, we will introduce and evaluate parallel implementations of SMG2S on homogeneous and heterogeneous clusters.

4.1 | Basic Implementation on CPUs and mutil-GPUs

The kernels of SMG2S are the sparse matrix-matrix multiplication (SpGEMM) AM and MA, and the matrix-matrix addition (AYPX operation) as $AM - MA$. SMG2S is initially implemented on CPUs with the matrix operations provided by PETSc[†]. All the sparse matrices during the generation procedure are stored in block diagonal Compressed Sparse Row (CSR) format supported in default by PETSc, which keeps the block diagonal and off-diagonal parts of matrices separately on each process into two sequence sub-matrices.

PETSc does not support SpGEMM and AXPY for multi-GPUs, so we implement them based on PETSc data structures, MPI, CUDA and cuSPARSE. The implementation of SpGEMM for multi-GPUs is similar to the one for CPUs, except the computation of sub-matrices on each MPI process is executed on its bound GPU device. The matrix operations on each GPU is supported by cuSPARSE, and final matrix can be obtained by gathering all sub-matrices from devices. This implementation for multi-GPUs was already presented by X. Wu et al.[24].

4.2 | Specific Communication-optimized Implementation on CPUs

The communication of parallel SpGEMM kernel can be specifically optimized based on the particular property of nilpotent matrix A. Denote the nilpotent matrix as $A(p, d, n)$, since it is determined by p, d and n. It is not necessary to implement it in parallel. Denote $J(i, j)$ the entry in row i and column j of matrix J; $J(i, :)$ all the entries of row i; and $J(:, j)$ all the entries of column j. As shown in Fig. 3, the right-multiplication $A(p, d, n)$ will cause all the entries of the first $n - p$ columns of M to shift right by an offset p. Denote MA the result gotten by the right-multiplying A on M. We have $MA(:, j) = M(:, j - p), \forall j \in p, \cdots, n - 1$, and $MA(:, j) = 0, \forall j \in 0, \cdots, p - 1$. Similarly, the left-multiplying $A(p, d, n)$ on M will shift up the whole entries of last $n - p$ rows by an offset p. Denote AM the matrix gotten by the left-multiplying A on M. We have $AM(i, :) = M(i + p, :), \forall i \in 0, \cdots, n - p - 1$, and $AM(i :,) = 0, \forall i \in p, \cdots, n - 1$. Moreover, the parameter d decides that $MA(:, r(d + 1)) = 0$ and $AM(r(d + 1), :) = 0$ with $r \in 1, \cdots, \lfloor \frac{n}{d+1} \rfloor$.

The communication-optimized SMG2S is implemented using MPI and C++. The sub-matrix on each process is stored in ELLPACK format, with key-value map container provided by C++. The key-value map implementation facilitates indexing and moving rows and columns. For the implementation on distributed memory systems, p, d and n are distributed across all processes. AM and MA are different from SpGEMM. Firstly, the matrix M is one-dimensional distributed by row across m MPI processes. As shown in Fig. 3b, for MA, there is no communication inter different MPI processes since the data are moved inside each row. Ensure that $\lfloor \frac{n}{m} \rfloor \geq p$, for AM, the intercommunication of MPI takes place when the MPI process k ($k \in 1, \cdots, m - 1$) should send the first p rows of their sub-matrix to the closest previous process numbering $k - 1$. The communication complexity for each process is $\mathcal{O}(np)$. When generating band matrix with low bandwidth b, it tends to be $\mathcal{O}(bp)$ with $p = 1$ or 2. The

[†]Portable, Extensible Toolkit for Scientific Computation

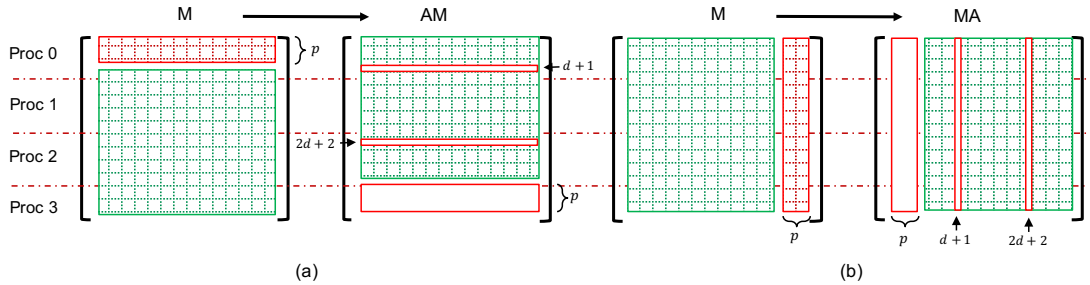


FIGURE 3 (a) AM operation; (b) MA operation.

Algorithm 2 Parallel MPI AM and MA Implementation

function AM(*input*: matrix M, matrix row number n, p, d, proc number m; *output*: matrix AM)
  Distribute t row blocks M_k of M to MPI process k
  for $p+1 \leq i < t$ **do**
    for $0 \leq j < n$ **do**
      if $M(i,j) \neq 0$ **then**
        $AM_k(i-p,j) = M_k(i,j)$
      end if
    end for
  end for
  for $0 \leq i < p$ **do**
    if $k \neq 0$ **then**
      $isend\ ith\ row\ M_k(i)$ to $k-1$
    end if
    if $k \neq m-1$ **then**
      $irecv\ ith\ row\ M_k(i)$ from $k+1$
      $AM_k(t-p+i) = M_k(i)$
    end if
  end for
end function

function MA(*input*: matrix M, matrix row number n, p, d, proc number m; *output*: matrix MA)
  Distribute t row blocks M_k of M to process k
  for $0 \leq i < t$ **do**
    for $p+1 \leq j < n$ **do**
      if $M_k(i,j) \neq 0$ **then**
        $MA_k(i,j+p) = M_k(i,j)$
      end if
    end for
  end for
end function

MPI-based optimization implementations of AM and MA are given by Algorithm 2. The communication inter MPI process is implemented by asynchronous sending and receiving functions. In this algorithm, M_k, MA_k and AM_k imply the sub-matrices on process k with t rows. The rows and columns of these sub-matrices in Algorithm 2 are all indexed by local indices.

We did not implement SMG2S with this specific optimization for multi-GPUs since its core is the data movement among computing units, which is not well suitable for the multi-GPUs architecture.

4.3 | Cost Analysis

In this section, we compare communication-optimized SMG2S with its standard version without optimization, including their memory requirement, floating-point operation count, the count of communication messages and the total size of inter-processor communication words. As we introduced in Section 2.2, the only existing generator which is able to generate sparse non-Hermitian matrices with given spectra is the xLATME routine in LAPACK. Due to its enormous memory requirement and inter-processor communication, this routine was not migrated by researchers to ScaLAPACK[‡], which is an extension of LAPACK for distributed memory platforms. In order to highlight the efficiency of our proposed method, we analyze additionly the cost of parallel implementation of xLATME with the ScaLAPACK routines. In this paper, we do not provide this practical implementation of xLATME based on ScaLAPACK, since the cost analysis in this section can already show the benefits of SMG2S compared with xLATME.

TABLE 1 Cost comparison of generators to generate $n \times n$ sparse matrix distributed over P processors in 1D fashion. We assume that p, d and h are much smaller than n, and xLATME generates test matrices which have the same bandwidth with the one generated by SMG2S. Cost of xLATEM is analyzed with the assumption that it is implemented in parallel with the routines of ScaLAPACK, but we do not provide its exact practical implementation in this paper.

	Memory	Floating-point operation count	Total word Size	Message Count
SMG2S	$\mathcal{O}(\frac{n}{P})$	$\mathcal{O}(\frac{n}{P})$	$\mathcal{O}(nP)$	$\mathcal{O}(P^2)$
SMG2S (optimized)	$\mathcal{O}(\frac{n}{P})$	$\mathcal{O}(\frac{n}{P})$	$\mathcal{O}(P)$	$\mathcal{O}(P)$
xLATME	$\mathcal{O}(\frac{n^2}{P})$	$\mathcal{O}(\frac{n^3}{P})$	$\mathcal{O}(n^2 \log P)$	$\mathcal{O}(n \log P)$

Table 1 details the cost analysis for basic SMG2S, communication-optimized SMG2S, and the parallel implementation of xLATME. In this table, assume that the $n \times n$ sparse matrix to be generated is distributed over P processors in 1D fashion. Assume also that p, d and h, the three parameters in SMG2S, are much smaller than n. As shown in Section 3, the bandwidth of generated matrix by SMG2S with parameters p, d and h is $h + 2pd - 1$. In this table, we suppose that xLATME generates test matrices which have the same bandwidth with SMG2S. Cost analysis in this section are all based on these assumptions. According to Table 1, firstly we can conclude that both two versions of SMG2S require $\mathcal{O}(\frac{n}{P})$ memory, when $n \gg p, d, h$. However, the memory requirement for xLATME will be $\mathcal{O}(\frac{n^2}{P})$, since the test matrices are initialized as dense ones, and then reduced to be banded. The two most important operations in xLATME are the dense matrix-matrix multiplication (parallel BLAS3 operation) that transforms a diagonal matrix with a given spectrum to be dense, and a long sequence of Householder transformation which reduces this matrix to be banded ($\approx 2n$ times parallel BLAS2 operations). As shown in Table 1, the parallel implementation of xLATME requires much more floating-point operations and inter-processor communication with much more communication words. For the standard SMG2S, it requires $\mathcal{O}(P^2)$ message communication, since for the parallel implementation of SpGEMM, each processor should have the communication with all the other processors. However, the specific optimized SMG2S requires only the communication between adjacent processors, which reduce the required message number to be $\mathcal{O}(P)$. Additionally, compared with basic SMG2S, the special parallel implementation scheme of optimized SMG2S reduces significantly the word size of inter-processor communications. With the cost analysis, we can firstly that both implementation of SMG2S can achieve good parallel performance on distributed memory systems. However, the scaling performance of basic SMG2S might tend to be bad with the increase of the number of processors, when the inter-processor communication becomes more and more important. Compared with basic SMG2S, specific optimized SMG2S reduces largely the floating-point operations, since it transfers the SpGEMM operation to be the data movement operation intra- and inter-processors. This conclusion can not be directly gotten from Table 1, and it introduces further speedup of specific optimized SMG2S over basic SMG2S.

4.4 | Parallel Performance Evaluation

The cost analysis of SMG2S in Section 4.3 demonstrates that SMG2S can achieve good scaling performance on distributed memory systems. In this section, its parallel performance is evaluated and validated on supercomputers *Tianhe-2* and *ROMEO*. *Tianhe-2* is installed at National Super Computer Center in Guangzhou, China, with 16000 compute nodes. Each node composes 2 Intel Ivy Bridge 12 cores @ 2.2 GHz. *ROMEO* is located at University of Reims Champagne-Ardenne, France, which is a

[‡]Scalable Linear Algebra PACKage

heterogeneous system with 130 BullX R421 nodes. Each node composes 2 Intel Ivy Bridge 8 cores @ 2.6 GHz and 2 NVIDIA Tesla K20x GPUs.

4.4.1 | Strong and Weak Scalability Results and Analysis

TABLE 2 Details for weak scaling and speedup evaluation.

(a) Matrix size for the CPU weak scaling tests on *Tianhe*-2.

CPU number	48	96	192	384	768	1536
matrix size	1×10^6	2×10^6	4×10^6	8×10^6	1.6×10^7	3.2×10^7

(b) Matrix size for the CPU weak scaling on *ROMEO*.

CPU number	16	32	64	128	256
matrix size	4×10^5	8×10^5	1.6×10^6	3.2×10^6	6.4×10^6

(c) Matrix size for the GPU weak scaling and speedup evaluation on *ROMEO*.

CPU or GPU number	16	32	64	128	256
matrix size	2×10^5	4×10^5	8×10^5	1.6×10^6	3.2×10^6

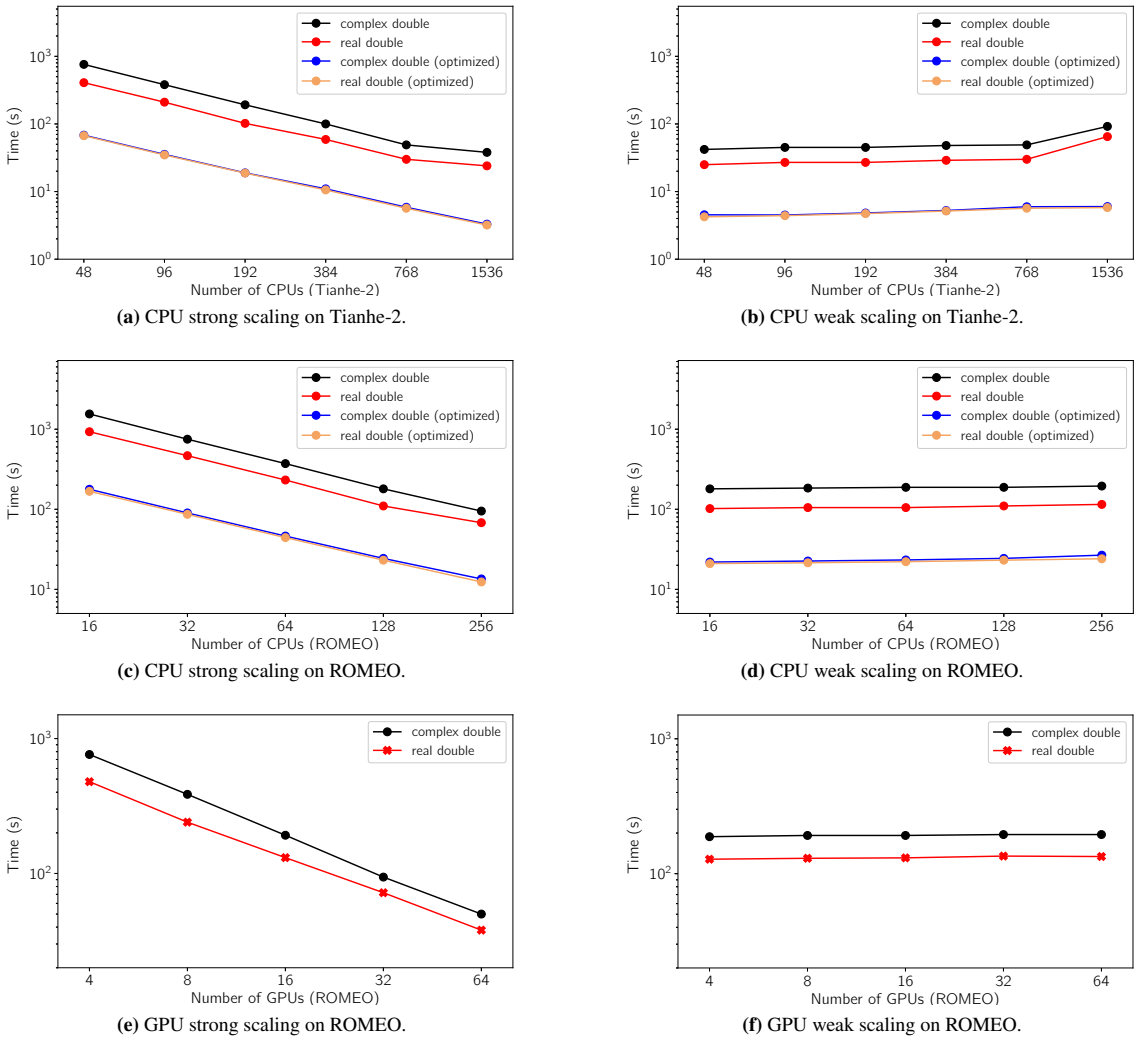
We use double-precision real and complex values to evaluate the scalability of SMG2S's different implementations on CPUs and multi-GPUs. All test matrices in this paper are generated with $h = 10$ and $d = 7$. The details of weak scaling experiments are given in Table 2. The matrix size of strong scaling experiments on *Tianhe-2* with CPUs, *ROMEO* with CPUs and *ROMEO* with multi-GPUs are respectively 1.6×10^7, 3.2×10^6 and 8.0×10^5. The results are given in Fig. 4. The weak scaling of PETSc-based SMG2S on *Tianhe-2* tends to be bad when MPI processes number is larger than 768, where the communication overhead becomes dominant for computation. But for communication-optimized SMG2S, both the strong and weak scaling perform well when the MPI process number is larger than 768. The experiments show SMG2S have good strong and weak scalability on multi-GPUs. In conclusion, SMG2S has good strong scalability when d and h are much smaller than the size of matrix, since it turns to be an $\mathcal{O}(n)$ problem. The weak scalability is good enough for most cases, since both implementations of SMG2S are not memory-bound (as shown in Table 1). The strong scalability of PETSc-based SMG2S has its drawback when the computing unit number come to be huge, where the communication overhead becomes dominant, since the inter-processor communication message number increases explosively with the increase of processor number P (the complexity of communication is $\mathcal{O}(P^2)$, as shown in Table 1). The implementation of communication-optimized SMG2S makes its strong and weak scalability better. It is also shown that the double precision complex type matrix generation takes almost 2× time over the double precision real type for PETSc-based SMG2S, but the time consumption of communication-optimized SMG2S seems similar for real and complex values. The reason is that there is no numerical values multiplication anymore in the optimized implementation of SMG2S.

4.4.2 | Speedup Results and Analysis

The speedup of PETSc-based SMG2S on multi-GPUs and communication-optimized SMG2S on CPUs compared with PETSc-based SMG2S on CPUs is also tested on *ROMEO*. We select the double precision complex values for the speedup evaluation. The details of experiments are also given in Table 2c. The results are shown in Fig. 5. We can find that PETSc-based SMG2S on multi-GPUs has almost 1.9× speedup over one on CPUs. The communication-optimized SMG2S has about 8× speedup over PETSc-based SMG2S on CPUs.

4.5 | Measurement and Profiling

After the demonstration of good scaling performance of communication-optimized SMG2S compared with PETSc-based SMG2S, we give more realistic performance measurement and profiling about SMG2S, which include its CPU usage, machine instructions, memory usage, and the trade-off between communication and computation. These measurements are made on



(a) CPU strong scaling on Tianhe-2.

(b) CPU weak scaling on Tianhe-2.

(c) CPU strong scaling on ROMEO.

(d) CPU weak scaling on ROMEO.

(e) GPU strong scaling on ROMEO.

(f) GPU weak scaling on ROMEO.

FIGURE 4 Strong and weak scaling results of SMG2S on different platforms. A base 2 logarithmic scale is used for X-axis, and a base 10 logarithmic scale for Y-axis.

the supercomputer *JURECA*, which is installed at Jülich Supercomputing Centre in Jülich, Germany, and equipped with 1872 compute nodes. Each node composes 2 Xeon E5-2680 v3 Haswell CPUs 12 cores @ 2.5 GHz.

4.5.1 | Computation vs Communication

For the implementation of applications on distributed memory systems, it is necessary to understand the trade-off between communication and computation, since that determines which configuration of the number of processors should be more suitable. In this paper, the computation and communication of communication-optimized SMG2S are measured by Score-P[§], which is a software system that provides a measurement infrastructure for profiling, event trace recording, and online analysis of HPC applications.

First of all, Fig. 6 gives an example of the event trace of communication-optimized SMG2S with 12 processors. This figure is plotted by Vampir using the trace files generated by Score-P, which details the information about functions, communication, and synchronization events. As shown in Fig. 6, it consists of a collection of rows, where each row represents a single process. Additionally, messages exchanged between two different processes are depicted as black lines. From this figure, we can conclude that the event of SMG2S is quite simple. Its communication scheme performs exactly as we expected: processor k ($k \in 1, \cdots, m-1$)

[§]Scalable Performance Measurement Infrastructure for Parallel Codes

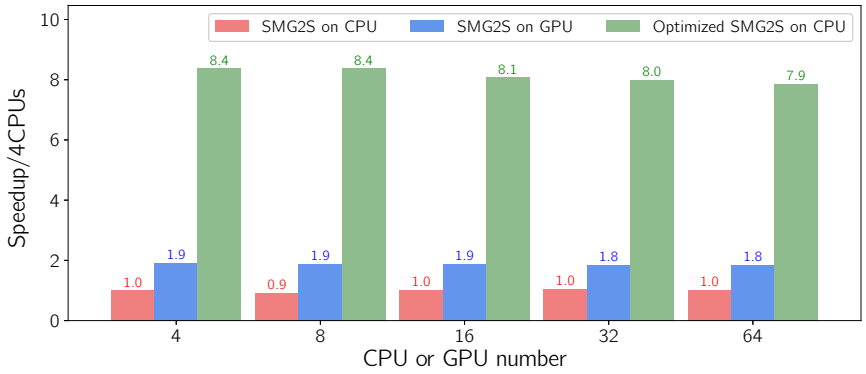


FIGURE 5 Weak scaling speedup comparison of GPUs on ROMEO.

sends messages to its closest previous processor which has the number $k-1$. Moreover, the implementation of intercommunication with MPI asynchronous communication functions removes the synchronization points, and improves the efficiency of communications.

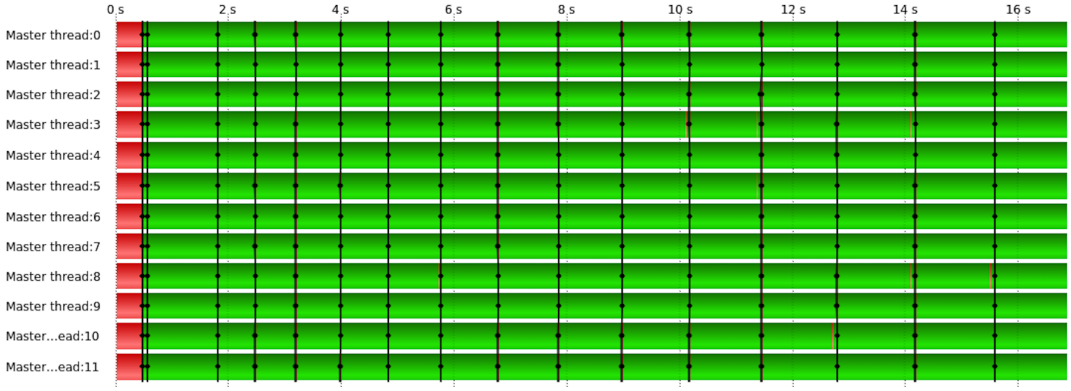


FIGURE 6 Event trace of communication-optimized SMG2S with 12 processors.

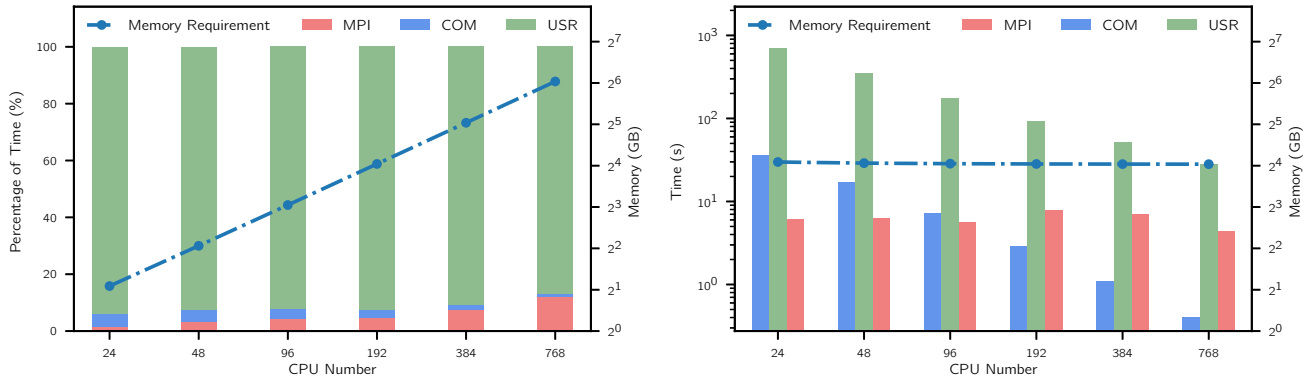
The trade-off between communication and computation of SMG2S is measured with both weak and strong scaling tests. The results are shown in Fig. 7. Generally, the functions of applications can be roughly divided into two types: *communication functions* which manage the inter-processor communications and synchronizations, and *computation functions* which focus on the computation in local without interaction with other processors. However, Score-P can detect and group the functions of pure MPI applications into three classes:

- *MPI group* (marked as red in Fig. 7): it contains all MPI functions;
- *USR group* (marked as green in Fig. 7): it contains all user functions that do not appear on a call-path to any MPI function;
- *COM group* (marked as blue in Fig. 7): it contains functions implemented by users that have a call-path to MPI functions.

Among the three groups, *MPI group* and *COM group* are both MPI-related. When talking about the implementation of SMG2S, the *MPI group* relates to MPI communication and synchronization functions, thus it is clear to be a group of *communication functions*. *COM group* relates to the user-defined functions which call the basic MPI functions and objects without any communications, thus it can also be identified as a group of *computation functions*.

For the profiling of computation and communication with weak scaling tests, the number of processors ranges from 48 to 768, meanwhile, the matrix size ranges from 2×10^6 to 6.4×10^7. The related results are given in Fig. 7a, where the percentages of time of different function groups in the total execution time are shown. We can conclude that, with the increase of processor number, communication becomes more and more important compared with the whole execution time. However, it is still fairly low even with 768 processors.

For the profiling of computation and communication with strong scaling tests, the number of processors ranges also from 48 to 768, meanwhile, the matrix size is fixed as 1.6×10^7. The related results are given in Fig. 7b, where the exact consumed time



(a) Profiling of computation and communication with weak scaling tests.

(b) Profiling of computation and communication with strong scaling tests.

FIGURE 7 Measurement of the trade-off between communication and computation of SMG2S.

of different function groups is shown. We can conclude that, with the increase of processor number, the total execution time decreases. However, the exact communication time keeps almost still for all the tests. This signifies that the communication scheme of SMG2S is efficient even with a large number of processors.

4.5.2 | Maximum Memory Requirement

During the procedures of profiling of the communication and computation by Score-P, the related maximum memory requirement for different tests has also been measured. The results are also shown in Fig. 7, marked by the dashed lines. For the measurement of memory related to the weak scaling test, we can find that when the matrix size n doubles, the memory requirement also doubles. The formula of memory cost has been given in Section 4.3 as $\mathcal{O}(n)$, which can be perfectly proved by these tests. The memory requirement related to the strong scaling test keeps almost still, since it is measured with fixed matrix size but different numbers of processors.

4.5.3 | Machine Instruction and CPU Usage

After the measurement of the trade-off between communication and computation, we develop the experimentation to profile the CPU performance related to the execution of SMG2S, which shows us more related performance details. The CPU related profiling is effectuated by PAPI[¶]. PAPI provides access to a collection of components that expose performance measurement opportunities across the hardware and software stack by intervening in the codes. In the experiments, *Cycles per processor per second*, *instructions per processor per second*, *instructions per cycle* and *CPU usage* are measured with different matrix sizes and numbers of processors.

TABLE 3 Profiling of machine instruction and CPU utilization of SMG2S.

	T1	T2	T3	T4	T5	T6	T7	T8
Processor number	24	24	24	24	192	192	192	192
Matrix size	7.5×10^5	1.5×10^6	6×10^6	1.2×10^7	6×10^6	1.2×10^7	4.8×10^7	9.6×10^7
Cycles per second per processor	2.87×10^9	2.85×10^9	2.87×10^9	2.87×10^9	2.64×10^9	2.75×10^9	2.84×10^9	2.86×10^9
Instructions per second per processor	3.92×10^9	3.89×10^9	3.94×10^9	3.95×10^9	4.14×10^9	4.31×10^9	4.43×10^9	4.51×10^9
Instructions per cycle	1.37	1.36	1.37	1.37	1.56	1.56	1.55	1.57
CPU usage	99.2%	96.0%	99.4%	99.8%	72.8%	86.6%	97.3%	98.6%

The results are shown in Table 3. In the experiments, we profiled SMG2S using different matrix sizes, and either 24 processors (1 node on *JURECA*) or 192 processors (8 nodes on *JURECA*). Firstly, we can conclude that for all the tests using 1 node, they

[¶]Performance Application Programming Interface

have similar machine cycles and instructions performance even they have different matrix sizes. They have also a very high CPU usage. For the tests with 8 nodes, they have a relative higher *ic* compared with the tests with 1 node, and the *CPU usage* augments from 72.8% to 98.6% with the increase of matrix sizes.

5 | ACCURACY VERIFICATION

In the previous section, we showed the good parallel performance of SMG2S, and then it was necessary to verify that the generated matrices are able to maintain given spectra with sufficient accuracy.

5.1 | Verification based on Shift Inverse Power Method

The scenario of accuracy verification for each given value can be summarized as finding its nearest eigenvalue, and verifying if this value is near enough to the given one. In this section, we present a method for accuracy verification using Shifted Inverse Power method, which can be easily implemented in parallel. Shifted Inverse Power method is able to compute the eigenpair whose eigenvalue is the nearest to a given value in a few iterations.

In details, for checking if the given value λ is the eigenvalue of a matrix, we select a shifted value σ which is close enough to λ. An eigenpair (λ', v') with the relation $Av' = \lambda' v'$ can be approximated in very few steps by Shifted Inverse Power method, with λ' the closest eigenvalue to σ. Since σ is very close to λ, it should be that λ and λ' are the same eigenvalue of a system, and v' should be the eigenvector related to λ. In reality, even if the computed eigenvalue is very close to the true one, the related eigenvector may be quite inaccurate. For the right eigenpairs, the formula $Av' \approx \lambda v'$ should be satisfied. Thus, we define the relative error as Formula (8) to quantify the accuracy.

$$error = \frac{||Av' - \lambda v'||_2}{||Av'||_2} \tag{8}$$

If $\lambda' = \lambda$, $error = 0$, if not, this generated matrix will not have an exact eigenvalue as λ. In reality, the exact solution cannot always be guaranteed due to the arithmetic rounding errors of floating operations. A threshold could be set for accepting it or not.

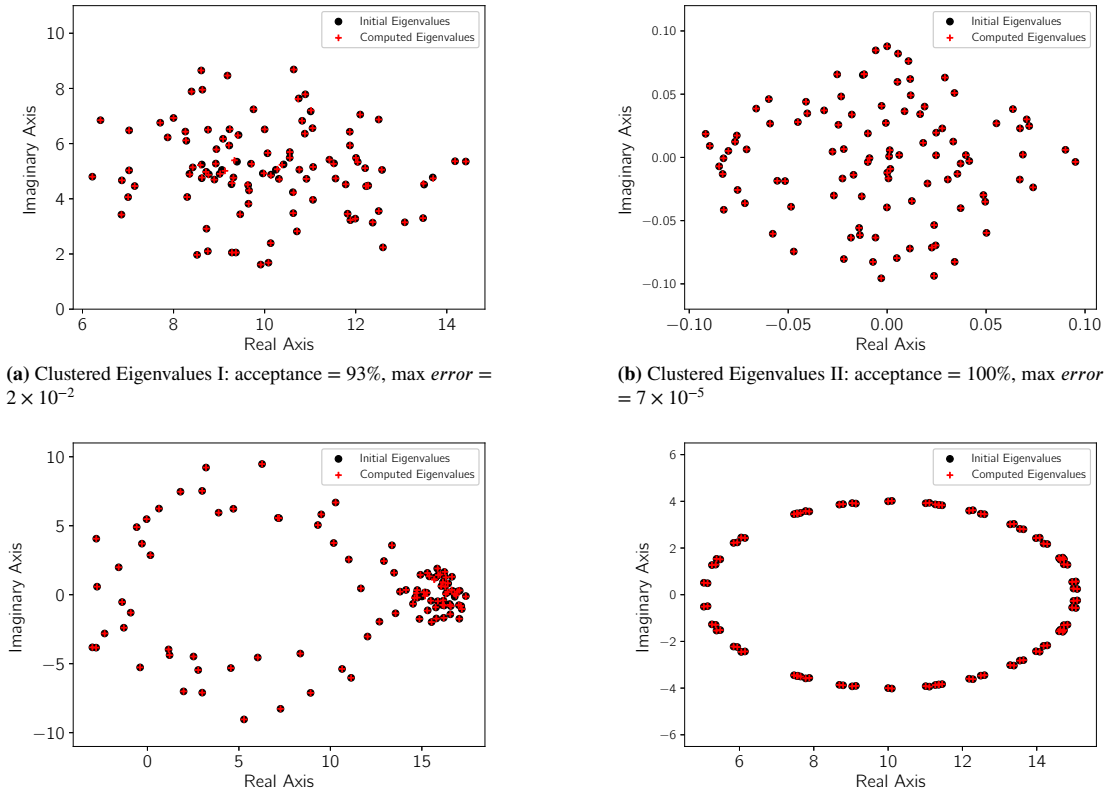
5.2 | Experimental Results

We tested the accuracy verification with four different selected spectral distributions. Fig. 8a and Fig. 8b are cases of clustered eigenvalues with different scales. Fig. 8c is a special case with the dominant part of eigenvalues clustered in a small region. Fig. 8d is a case that composes the conjugate and closest pair eigenvalues. These figures compare the difference between the given spectra (noted as initial eigenvalues in the figures) and the approximated ones (noted as computed eigenvalues) by Shifted Inverse Power Method. The acceptance threshold is 1.0×10^{-3}.

This acceptance rate for Fig. 8a, Fig. 8b, Fig. 8c and Fig. 8d are respectively 93%, 100%, 94% and 100%. The maximal *error* for them are respectively 3×10^{-2}, 7×10^{-5}, 3×10^{-2} and 3×10^{-7}. In conclusion, SMG2S is able to keep accurately the given spectra even for the clustered and closest eigenvalues. In some cases, a very little number of too clustered eigenvalues may result in the inaccuracy, but in general, the generated matrix can fulfill the need to evaluate the linear system and eigenvalue solvers.

6 | EXAMPLE: EVALUATING ITERATIVE SOLVERS BY SMG2S

SMG2S is designed to evaluate different linear systems and eigenvalue solvers. In this section, we give an example to demonstrate its workflow and ability to evaluate these solvers. A class of Krylov iterative methods is one of the most powerful tools to solve large sparse linear systems. Convergence analysis of these methods is not only of great theoretical importance, but it can also guide to answer practically relevant questions about improving their performance using the preconditioners. As introduced in Section 2.1, the convergence of Krylov iterative solvers and some related preconditioners are correlated with the spectral distribution of coefficient matrices. In this section, two iterative solvers are selected to illustrate the importance of SMG2S.



(a) Clustered Eigenvalues I: acceptance = 93%, max *error* = 2×10^{-2}

(b) Clustered Eigenvalues II: acceptance = 100%, max *error* = 7×10^{-5}

(c) Dominant Clustered Eigenvalues: acceptance = 94%, max *error* = 3×10^{-2}

(d) Conjugate and Closest Eigenvalues: acceptance = 100%, max *error* = 3×10^{-7}

FIGURE 8 Verification using Different Types of Spectra.

6.1 | SMG2S workflow to evaluate iterative Solvers

As presented in the previous section, SMG2S is implemented in parallel with optimized communication based on MPI and C++. The released open source software of SMG2S[#] provides the interfaces to different programming languages such as C and Python, and to the scientific computational libraries PETSc, SLEPc[‖] and Trilinos[**], which facilitates the users to evaluate their implementations of iterative solvers. More details of these interfaces can be found in the manual of SMG2S [25].

The workflow of SMG2S for evaluating the iterative solvers is shown in Fig. 9. It generates the test matrices in parallel, which means that each slice of these matrices is already allocated on the corresponding computing unit. The interfaces provided to PETSc, Trilinos, and other public or personal parallel solvers, can directly restore the distributed data into the necessary data structures of these different libraries. The restored data can be directly used to evaluate the numerical methods, without concerning about the load of matrix file from local file systems. In practice, for the large-scale matrix applications on extreme-scale machines, their I/O can be the bottleneck. This feature of SMG2S can significantly reduce the I/O of applications and improve their efficiency to evaluate the numerical methods.

6.2 | Selected Iterative Solvers

We select Krylov iterative method GMRES (Generalized Minimal Residual) method and UCGLE (Unite and Conquer GMRES/LS-ERAM method) to understand their behaviors of convergence with different spectral distributions, and parallel performance on supercomputers. At first, we give a glance at the two selected solvers and illustrate the importance of evaluating their numerical performance with different spectra.

[#] https://smg2s.github.io
[‖] Scalable Library for Eigenvalue Problem Computations
[**] https://trilinos.github.io

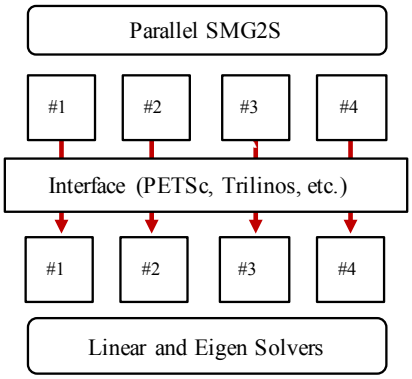


FIGURE 9 SMG2S Workflow and Interface.

6.2.1 | GMRES Method

GMRES is a Krylov iterative method to solve non-Hermitian linear systems $Ax = b$. It approximates the solution x_m from an initial guess x_0, with the minimal residual in a selected Krylov subspace. It was introduced by Saad and Schultz in 1986[26].

With the Arnoldi reduction in GMRES, the formulas can be constructed as follows

$$\begin{aligned} AV_m &= V_m H_m + \omega_m e_m^T = V_{m+1}\overline{H} \\ V_m^T A V_m &= H_m. \end{aligned} \tag{9}$$

In fact, any vector x in subspace $x_0 + K_m$ can be written as

$$x = x_0 + V_m y. \tag{10}$$

with y a m-dimensional vector, V_m an orthonormal basis of the Krylov subspace K_m. The norm of residual $R(y)$ of $Ax = b$ can be given as:

$$\begin{aligned} R(y) &= ||b - Ax||_2 = ||b - A(x_0 + V_m y)||_2 \\ &= ||V_{m+1}(\beta e_i - \overline{H}_m y)||_2 = ||\beta e_i - \overline{H}_m y||_2. \end{aligned} \tag{11}$$

x_m can be obtained as $x_m = x_0 + V_m y_m$ where $y_m = argmin_y ||\beta e_i - \overline{H}_m y||_2$. The minimizer y_m is inexpensive to compute from this least-squares problem if m is typically small. If m is large, GMRES can be restarted after a number of iterations, to avoid enormous memory and computational requirements. For the general normal matrix, GMRES converge more quickly with larger Krylov subspace size. The difficulty of restarted GMRES is that it can stagnate when the matrix is not positive definite. Usually, the preconditioning techniques are used to speed up the convergence. Krylov iterative methods, including GMRES, are often implemented for parallel machines, since the 1990s, e.g., the implementation of iterative solvers for the non-Hermitian matrix on the Connection Machine (CM-2) presented by Petiton[27].

6.2.2 | UCGLE

Another iterative solver to be evaluated is UCGLE which was introduced by X. Wu et al.[28] to solve large non-Hermitian linear systems with the reduction of global communications. UCGLE consists of three computational components with asynchronous communications: ERAM Component, GMRES Component, and LSP (Least Squares Polynomial) Component. UCGLE is a variant of hybrid iterative method preconditioned by a selected polynomial, targeting for the coming exascale computing.

Inside UCGLE, GMRES Component is implemented as a classic restarted GMRES solver, which is used to solve the linear systems. LSP Component and ERAM Component serve as the preconditioning part, which accelerates the convergence of GMRES Component by a Least Squares polynomial. This Least Squares polynomial preconditioning technique was firstly introduced by Y. Saad[14] in 1987. The information used to speed up convergence are the dominant eigenvalues. Inside UCGLE, ERAM Component is implemented as an eigenvalue solver, which is able to approximate a selected number of eigenvalues, and asynchronously send them to LSP Component. The latter uses these received eigenvalues to construct a refined polygon, and then a Least Squares polynomial. A temporary solution can be generated by this polynomial, which will be used as a new restarted vector for GMRES Component. Theoretically, Least Squares polynomial is able to accelerate the convergence since it

can amplify the Euclidean norm of dominant eigenvalues. However, their effects on different spectral distribution are not clear. The degree of Least Squares polynomial d is an important parameter to speed up the convergence. The value of this parameter should be relatively large to allow enough acceleration by the Least Squares polynomial. However, it cannot be too large. Otherwise, the norm of residual vectors will be too large to converge in time for each cycle of restart.

The introduction of SMG2S allows evaluating both the parallel and numerical performance of UCGLE on large-scale supercomputers. The numerical performance of UCGLE with different linear systems is not presented yet by X. Wu et al.[28]. Thus, the first part of this section will evaluate the convergence performance of UCGLE, and compared it with conventional GMRES methods. The preconditioning part of UCGLE uses the dominant eigenvalues to accelerate the convergence, and it is urgently necessary to evaluate the influence of the spectral distributions on this acceleration.

6.3 | Setup of Test Matrices by SMG2S

We use SMG2S to generate test matrices with different spectral properties for both GMRES method and UCGLE. In the experiments, all Right-hand Sides of linear systems are generated in random using given seed states, and then normalized into the same value. In this section, eight test matrices are generated by SMG2S($spec$), where $spec$ implies the spectral generation functions for different spectra. The definition of these functions is given in TABLE 4. The dimension of all test matrices is fixed as 2000×2000. Thus the number of generated eigenvalues for each test matrix is also $N = 2000$. We give the $spec$ of spectrum I in TABLE 4 as an example to explain the formula of generation functions: its first part is defined as $0.6+(rand(0,0.01)+0.55)\cos(2\pi i/N-\pi)$, which signifies that the real parts of eigenvalues of spectrum I are the floating numbers generated randomly by this function, with $i \in 0, 1, \cdots, N-1$ the indices for these generated eigenvalues. Similarly the imaginary parts of eigenvalues of spectrum I are randomly generated by the function $(rand(0,0.01)+0.1)\sin(2\pi i/N-\pi)$. The spectrum VII and VIII in TABLE 4 are generated by the spectrum generation functions defined in a different manner. In these two cases, their first 50 and the rest eigenvalues are respectively generated with different given functions. These special spectrum generation functions facilitate the definition of spectral distribution into two separately clustered groups. The spectral distributions of all the eight tests are plotted in Fig. 10 and Fig. 11 (the black dot points with the legends noted as *Given Eigenvalues* in the figures).

TABLE 4 Spectrum generation functions: the size of all spectra is fixed as $N = 2000$, $i \in 0, 1, \cdots, N-1$ is the indices for the eigenvalues.

N^o	real part	imaginary part
I	$0.6 + (rand(0,0.01) + 0.55)\cos(2\pi i/N - \pi)$	$(rand(0,0.01) + 0.1)\sin(2\pi i/N - \pi)$
II	$0.3 + (rand(0,0.01) + 0.55)\cos(2\pi i/N - \pi)$	$(rand(0,0.01) + 0.1)\sin(2\pi i/N - \pi)$
III	$-0.6 + (rand(0,0.01) + 0.55)\cos(2\pi i/N - \pi)$	$(rand(0,0.01) + 0.1)\sin(2\pi i/N - \pi)$
IV	$0.6 + (rand(0,0.01) + 0.55)\cos(4\pi i/N - \pi)$	$0.2 + (rand(0,0.01) + 0.1)\sin(4\pi i/N - \pi)\ \forall i < 1000$ $-0.2 - (rand(0,0.01) + 0.1)\sin(4\pi i/N - \pi)\ \forall i \geq 1000$
V	$0.006 + rand(0, 0.5)$	0.0
VI	$-0.006 + rand(0, 0.5)$	0.0
VII	$60.0 + (rand(0,0.0001) + 0.00012)\cos(2\pi i/N - \pi)\ \forall i < 50$ $0.6 + (rand(0,0.01) + 0.55)\cos(2\pi i/N - \pi)\ \forall i \geq 50$	$(rand(0,0.0001) + 0.00012)\sin(2\pi i/N - \pi)\ \forall i < 50$ $(rand(0,0.01) + 0.1)\sin(2\pi i/N - \pi)\ \forall i \geq 50$
VIII	$-60.0 - (rand(0,0.0001) + 0.00012)\cos(2\pi i/N - \pi)\ \forall i < 50$ $-0.6 - (rand(0,0.01) + 0.55)\cos(2\pi i/N - \pi)\ \forall i \geq 50$	$(rand(0,0.0001) + 0.00012)\sin(2\pi i/N - \pi)\ \forall i < 50$ $(rand(0,0.01) + 0.1)\sin(2\pi i/N - \pi)\ \forall i \geq 50$

6.4 | Numerical Performance Evaluation of Iterative Solvers by SMG2S

Inside UCGLE, the Least Squares Polynomial uses the dominant eigenvalues to accelerate the convergence of iterative methods. The behaviors of Least Squares polynomial preconditioning are studied using the test matrices generated by SMG2S with eight types of spectra in TABLE 4. The dimension of all eight test matrices is 2000. In this section, the impacts of spectral distributions are evaluated for both classic restarted GMRES and UCGLE. Classic GMRES and GMRES Component in UCGLE are both implemented with the one provided by PETSc, ERAM Component is implemented with the eigensolver provided by SLEPc,

and the LSP Component is implemented based on the routines provided by LAPACK. Their asynchronous communications are supported by the specified implementation through MPI standard.

For all the tests, the Krylov subspace size m_a for ERAM Component is limited. Thus the accuracy of approximated eigenvalues is also low. These approximated eigenvalues by ERAM Component of all the eight tests are also plotted in Fig. 10 and Fig. 11 (the red crosses with the legends noted as *Approximated Eigenvalues* in the figures). As shown in the figures, the approximated eigenvalues are not really accurate compared with the given ones, due to the limitation of Krylov subspace of ERAM Component. However, the speedup of Least Squares polynomial preconditioning can still be splendid. The purpose to limit m_a is to make sure generate enough eigenvalues for the first restart cycle of GMRES Component inside UCGLE. If this subspace size is too large, or the conditions for the eigenvalues to be accepted are too strict, there will be no acceleration by Least Squares polynomial preconditioning. The convergence performance of GMRES and UCGLE are also given in Fig. 10 and Fig. 11.

6.4.1 | Spectral Distribution as Ellipses

The first three spectra are the ones with eigenvalues distributed as ellipses in the real-imaginary plane, which are given in Fig. 10a, 10b and 10c.

As shown in Fig. 10a, the generated spectrum I is quasi-symmetric to the real axis, and all the real parts of these eigenvalues are positive, which are marked as the black dot points in the figure. The eigenvalues approximated by ERAM Component are labeled as the red crosses in the figure. In the experiments, denote m_g as the Krylov subspace size of both GMRES Component and classic GMRES method. For spectrum I, the two solvers are both tested with $m_g = 20$ and 80. Firstly, we can conclude that for the classic GMRES, its convergence with $m_g = 20$ and 80 performs similarly, the enlargement of Krylov subspace size has no effects on the acceleration of convergence. For UCGLE with $m_g = 20$, it has more than 3× speedup on the convergence compared the classic GMRES. However, for the case with larger Krylov subspace size $m_g = 80$, UCGLE only has about 1.4× speedup over the classic GMRES. In fact, for the matrices with a spectral distribution similar to spectrum I, the Least Squares polynomial preconditioning is always efficient enough, and enlargement of Krylov subspace size is less useful to speed up the convergence. Thus, it is better to benefit from this preconditioning as soon as possible. Hence, smaller m_g might be preferred for GMRES Component, since it can profit the Least Squares polynomial preconditioning in time.

Spectrum II is also quasi-symmetric to the real axis. The real parts of some eigenvalues in this spectrum are positive, and for the rest, their real parts are negative, shown as Fig. 10b. In this case, neither GMRES or UCGLE can achieve the convergence even with much larger Krylov subspace size $m_g = 200$. Moreover, UCGLE will quickly achieve the divergence. In this case, where the original point is inside the spectrum of the coefficient matrix, Least Squares polynomial method cannot give a good approximation solution for the restart cycle of GMRES Component, which leads to the difficulty for convergence. In the current implementation of UCGLE, for most cases with the origin point inside spectrum, the preconditioning of Least Squares polynomial is not applicable.

The spectrum III in Fig. 10c is also quasi-symmetric to the real axis, and the real parts of all the eigenvalues are negative. This case is similar to the one in Fig. 10a, that UCGLE with $m_g = 20$ has about 3× speedup over the classic GMRES, and UCGLE with $m_g = 80$ has almost 1.3× speedup. The acceleration of Least Squares polynomial preconditioning inside UCGLE is obvious, and this kind of spectral distribution is suitable for the Least Squares polynomial preconditioning.

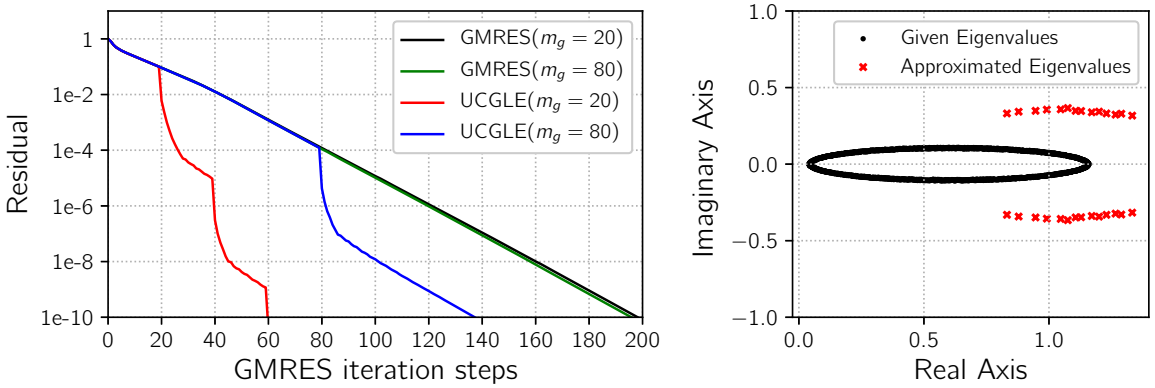
6.4.2 | Spectral Distribution as Two Ellipses Symmetric to Real Axis

The spectrum IV in Fig. 10d consists of two separate ellipses which are symmetric to the real axis, and the real parts of all eigenvalues are positive. The Krylov subspace size m_g for GMRES method and UCGLE is both set to be 20. We could conclude that UCGLE is suitable for this case with almost 4× speedup compared the conventional GMRES.

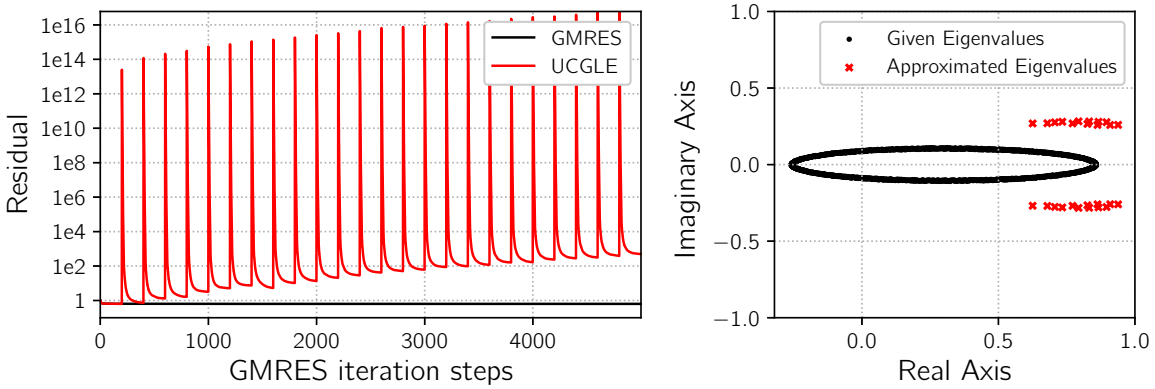
6.4.3 | Spectral Distribution with Real Eigenvalues

The spectrum V in Fig. 11a and spectrum VI in Fig. 11b are generated in random with all the eigenvalues located on the real axis; the imaginary parts of all eigenvalues are zero. For spectrum V, all eigenvalues are positive. The eigenvalues approximated by ERAM Component are complex, which are also marked by the red crosses in this figure. UCGLE has more than 6× speedup for this spectrum, compared with the classic GMRES.

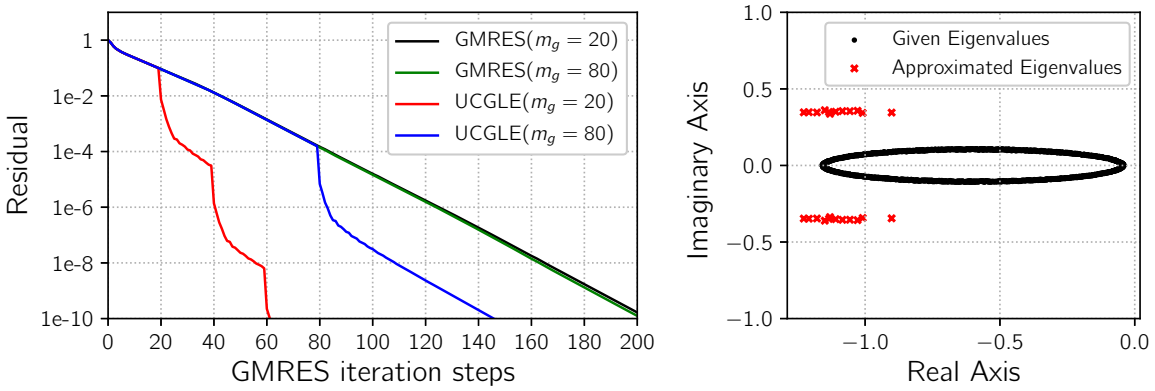
The spectrum VI in Fig. 11b is generated with a small number of eigenvalues being negative, and the others keep still positive. In this case, the origin point is inside of the spectrum, the convergence for both GMRES method and UCGLE is hard to be



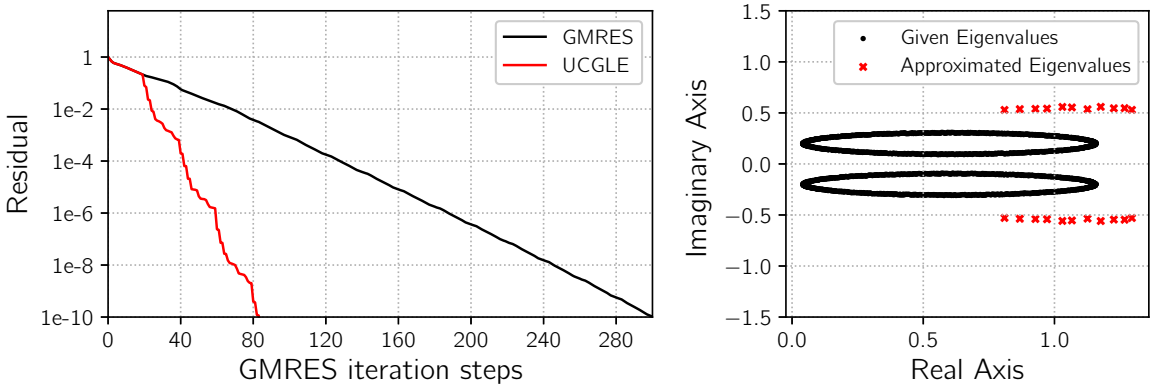
(a) Spectral Distribution I: matrix size = 2000, $d = 10$.



(b) Spectral Distribution II: matrix size = 2000, $m_g = 200$, $d = 10$.

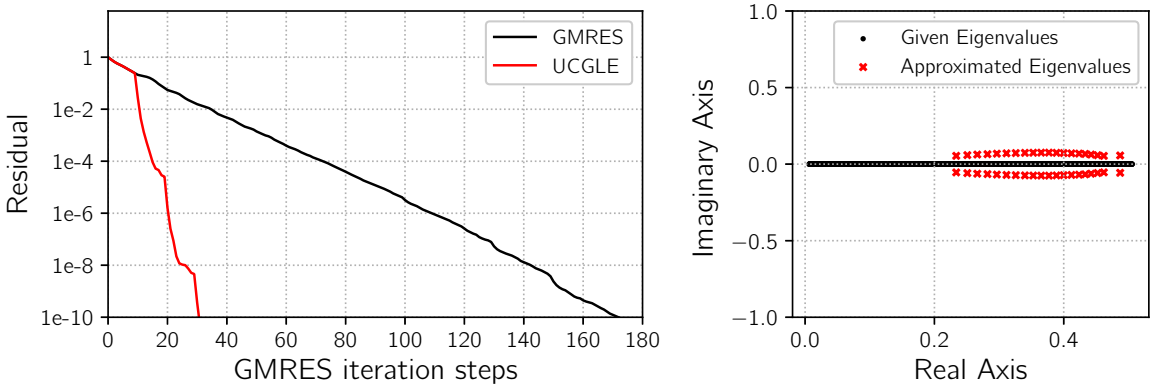


(c) Spectral Distribution III: matrix size = 2000, $d = 10$.

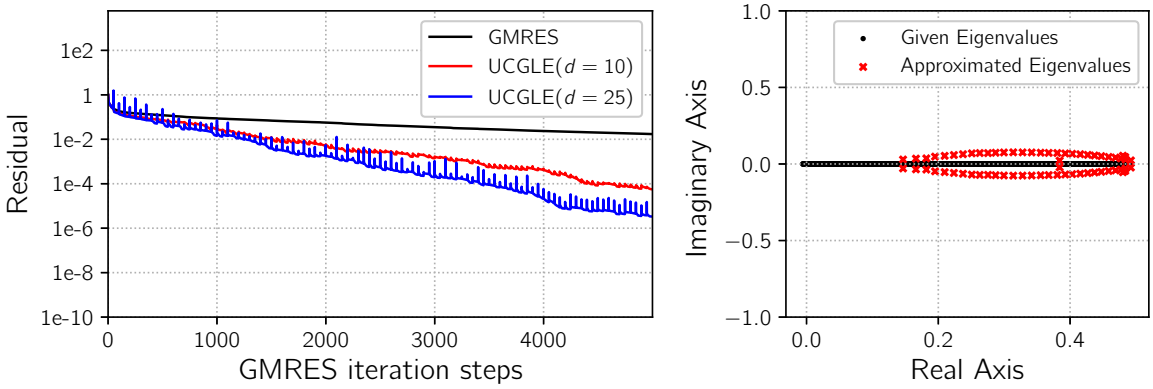


(d) Spectral Distribution IV: matrix size = 2000, $m_g = 20$, $d = 10$.

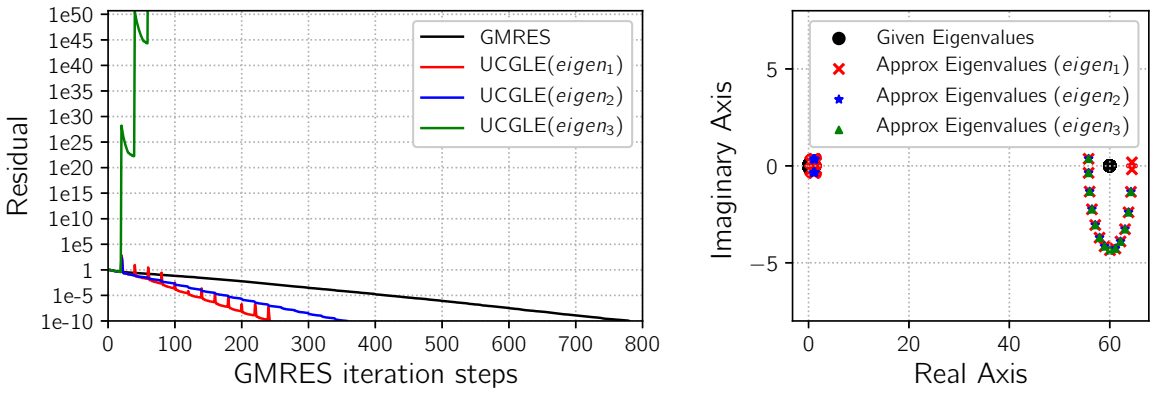
FIGURE 10 Impacts of Spectrum on Convergence.



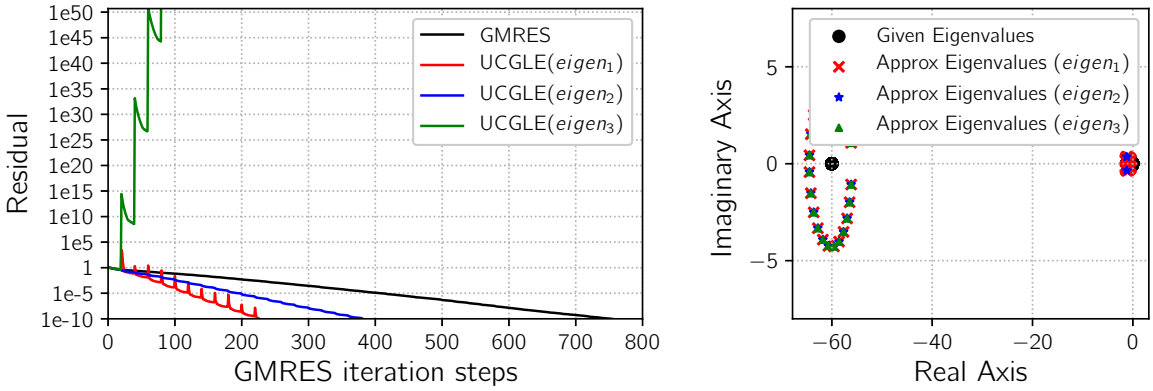
(a) Spectral Distribution V: matrix size = 2000, $m_g = 10$, $d = 10$.



(b) Spectral Distribution VI: matrix size = 2000, $m_g = 50$.



(c) Spectral Distribution VII: matrix size = 2000, $m_g = 20$, $d = 10$.



(d) Spectral Distribution VIII: matrix size = 2000, $m_g = 20$, $d = 10$.

FIGURE 11 Impacts of Spectrum on Convergence.

obtained. However, UCGLE can still have a little speedup over the classic GMRES method. Moreover, If we continue to augment the degree of Least Squares polynomial d from 10 to 25, further acceleration can still be observed.

6.4.4 | Spectral Distribution with Two Separately Clustered Sets

The spectrum VII and VIII in Fig. 11c and Fig. 11d are also quasi-symmetric to the real axis, and the real parts of eigenvalues for the two spectra are respectively all positive and negative. The two spectra are generated with a special manner which makes the eigenvalues be grouped into two separate clustered sets with a relatively long distance in the real-imaginary plane. With the change of Krylov subspace size m_a of ERAM Component, different numbers of eigenvalues can be approximated. For both spectra, three cases are tested in the experiments with different numbers of eigenvalues approximated and used to construct the Least Squares polynomial. The three cases are respectively denoted as $eigen_1$, $eigen_2$ and $eigen_3$, which are marked with different colors and markers in Fig. 11c and Fig. 11d. In the experiments, $eigen_1$ and $eigen_2$ are cases which approximate a number of eigenvalues in both two clustered sets. However, $eigen_3$ are a little number of eigenvalues approximated in only one clustered set with larger Euclidean norms (the right clustered group in Fig. 11c, and the left clustered group in Fig. 11d). Additionally, $eigen_1$ approximates more eigenvalues in the both two clustered groups than $eigen_2$. We could conclude from Fig. 11c and Fig. 11d that UCGLE with the case $eigen_1$ converge the most rapidly, which achieves about 3× speedup than the conventional GMRES. UCGLE with the case $eigen_2$ has almost 2× speedup. However, UCGLE with $eigen_3$ diverges quickly in very few iteration steps. The reason is that the case $eigen_3$ approximates only one clustered group, and the polygon constructed by these eigenvalues cannot represent the real spectral distribution of linear systems, which makes the Euclidean norm of residual vector generated by Least Squares polynomial explode in short time, and it is impossible to achieve the convergence.

6.5 | Scaling Performance Evaluation of Iterative Solvers by SMG2S

The iterative methods implemented in parallel on supercomputers introduce communications between different computing units. UCGLE is specifically designed following a distributed and parallel programming scheme. Its purpose is to improve the parallel performance of iterative methods by minimizing the number of communications, promoting the asynchronicity and reducing the synchronize points. The parallel performance of UCGLE was already evaluated on the supercomputer *ROMEO*[28]. The large-scale test matrices were built by performing several copies of a same small industrial matrix onto the block diagonal.

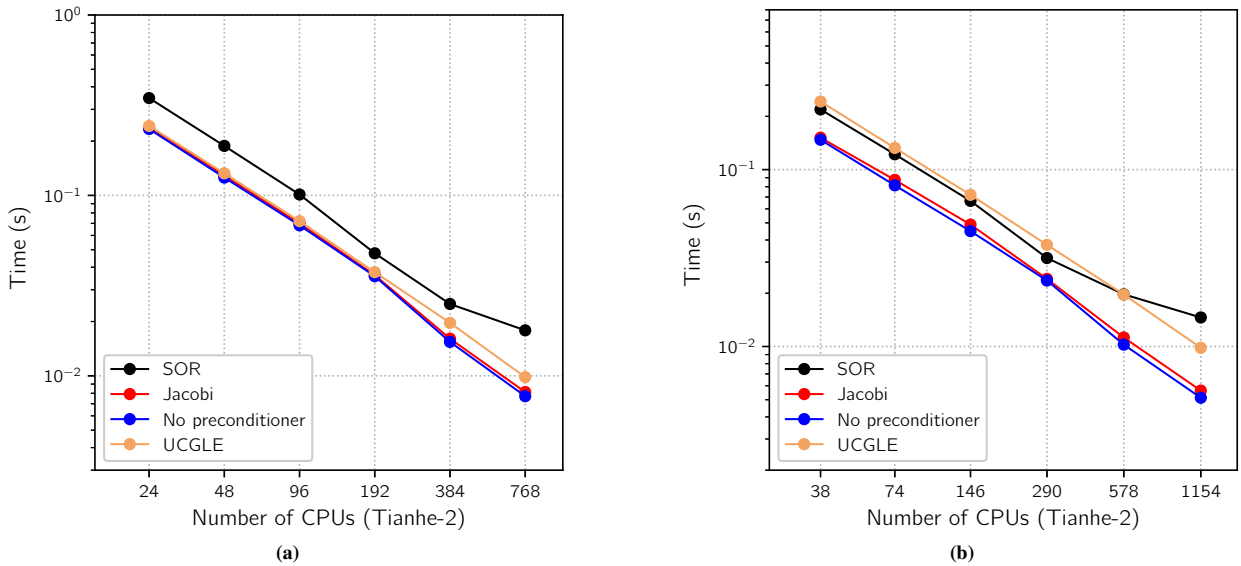


FIGURE 12 Scalability per iteration and performance comparison of UCGLE with GMRES with or without preconditioners on *Tianhe-2*. A base 10 logarithmic scale is used for Y-axis of (a) and (b); a base 2 logarithmic scale is used for X-axis of (a).

SMG2S can generate large-scale test matrices in parallel. New scaling results are obtained on supercomputer *Tianhe-2* with SMG2S. The strong scaling performance of classic GMRES without/with preconditioners (Jacobi and SOR preconditioners) and UCGLE are evaluated on *Tianhe-2* with more CPUs. The size of the generated test matrix is 1.8×10^7. The experimental

results are shown in Fig. 12. Same as the scaling tests on *ROMEO* presented by X. Wu et al.[28], Y-axis in Fig. 12 represents the average time per iteration, which is calculated after a fixed number of iteration steps. Fig. 12a shows the comparison of strong scalability of GMRES Component in UCGLE with other conventional methods. In this test, conventional methods have the same number of computing unit number with GMRES Component in UCGLE. Fig. 12b gives the performance comparison of different methods. In this test, conventional methods have the same number of computing unit number with the total number of UCGLE. Better scaling performance can still be obtained for UCGLE compared with SOR preconditioned GMRES, which is similar to the results from *ROMEO* presented by X. Wu et al.[28]. In Fig. 12b, the time per iteration of UCGLE seems not comparable with other methods since it allocates more computing units for ERAM and LSP Components. However, it can accelerate the convergence by the Least Squares polynomial with good scalability, better than the preconditioners Jacobi and SOR, which is also already shown[28].

6.6 | Remarks

Several remarks can be gotten from the previous experiments which evaluate the performance of GMRES method and UCGLE by SMG2S:

(1) First of all, the efficiency and ability of SMG2S for the benchmark of the numerical and parallel performance of iterative methods are proved. The convergence analysis of these linear algebra solvers with different spectral distributions and the parallel efficiency evaluation can guide the users and researchers to design and implement their methods and preconditioners.

(2) For the Least Squares polynomial preconditioning and UCGLE, if the real parts of the eigenvalues of the coefficient matrix are all positive or negative, and the spectral distribution is (quasi-)symmetric to the real axis, UCGLE can always efficiently accelerate the convergence compared with the classic GMRES method.

(3) If the real parts of some eigenvalues of the coefficient matrix are positive and of some eigenvalues are negative, the divergence of UCGLE can be easily obtained. Generally, the Least Squares polynomial preconditioning and UCGLE are not suitable for this case. However, it might exist some particular matrices with which UCGLE can still achieve limited acceleration compared with the classic GMRES method.

(4) For the matrices with a good spectral distribution as we talked in (2), the approximated eigenvalues do not need to be too accurate to achieve splendid speedup by the Least Squares polynomial preconditioning of UCGLE.

(5) The more eigenvalues approximated to construct the Least Squares polynomial, the more acceleration of UCGLE by the Least Squares polynomial preconditioning will achieve.

(6) For the eigenvalues distributed into the different clustered groups with their real parts to be all positive or negative, in order to obtain the acceleration on the convergence, much more eigenvalues should be approximated to construct the Least Squares polynomial preconditioning. At least, these approximated eigenvalues should be able to represent the different clustered groups of spectra. If the different groups of clustered eigenvalues are very discrete, it is difficult for ERAM Component to approximate all of them with small Krylov subspace size. Hence, more efficient eigensolver should be implemented to replace the existing ERAM Component.

7 | CONCLUSION AND PERSPECTIVES

In this paper, we presented a scalable matrix generator and its parallel implementation on homogeneous and heterogeneous clusters. It allows generating large-scale non-Hermitian matrices with customized eigenvalues to evaluate the impact of spectra on the linear/eigenvalue solvers on large-scale platforms. The experiments proved its good scalability and the ability to keep the given spectra with acceptable accuracy. For large matrices, the I/O operation on supercomputers is always a bottleneck even with the high bandwidth. The matrices generated in parallel by SMG2S, with data already allocated on different processes, can be used directly to evaluate the numerical methods without concerning the I/O operation. The interfaces of SMG2S to C, Python and scientific libraries PETSc and Trilinos are provided. Finally, an example which evaluates GMRES and a hybrid method using tests matrices with different spectral properties demonstrated the efficiency and ability of SMG2S for evaluating the numerical and parallel performance of linear algebra methods.

ACKNOWLEDGMENT

This work was partially supported by the HPC Center of Champagne-Ardenne *Romeo*. It is funded by the project *MYX* of *French National Research Agency (ANR)* (Grant No. ANR-15-SPPE-003) under the SPPEXA framework. Special thanks to the Jülich Supercomputing Centre for providing JURECA platform which helped us to supplement the measurement and profiling results in this paper.

References

1. Shinozuka M, Astill CJ. Random eigenvalue problems in structural analysis. *AIAA journal* 1972; 10(4): 456–462.
2. Shim CH, Maruoka F, Hattori R. Structural analysis on organic thin-film transistor with device simulation. *IEEE Transactions on Electron Devices* 2010; 57(1): 195–200.
3. Liu Y, Talbi A, Pernod P, Bou Matar O. Highly confined Love waves modes by defect states in a holey SiO 2/quartz phononic crystal. *Journal of Applied Physics* 2018; 124(14): 145102.
4. Liu Y, Talbi A, Djafari-Rouhani B, et al. Interaction of Love waves with coupled cavity modes in a 2D holey phononic crystal. *Physics Letters A* 2019.
5. Buffa A, Ciarlet P, Jamelot E. Solving electromagnetic eigenvalue problems in polyhedral domains with nodal finite elements. *Numerische Mathematik* 2009; 113(4): 497–518.
6. Ciftci H, Hall RL, Saad N. Asymptotic iteration method for eigenvalue problems. *Journal of Physics A: Mathematical and General* 2003; 36(47): 11807.
7. Saad Y. Krylov subspace methods for solving large unsymmetric linear systems. *Mathematics of computation* 1981; 37(155): 105–126.
8. Liesen J, Strakos Z. *Krylov subspace methods: principles and analysis*. Oxford University Press . 2013.
9. Trefethen LN. Approximation theory and numerical linear algebra. In: Springer. 1990 (pp. 336–360).
10. Starke G. Field-of-values analysis of preconditioned iterative methods for nonsymmetric elliptic problems. *Numerische Mathematik* 1997; 78(1): 103–117.
11. Nevanlinna O. *Convergence of iterations for linear equations*. Birkhäuser . 2012.
12. Saad Y. *Iterative methods for sparse linear systems*. 82. siam . 2003.
13. Morgan RB. GMRES with deflated restarting. *SIAM Journal on Scientific Computing* 2002; 24(1): 20–37.
14. Saad Y. Least squares polynomials in the complex plane and their use for solving nonsymmetric linear systems. *SIAM Journal on Numerical Analysis* 1987; 24(1): 155–169.
15. Manteuffel TA. The Tchebychev iteration for nonsymmetric linear systems. *Numerische Mathematik* 1977; 28(3): 307–327.
16. Saad Y. SPARSEKIT: a basic tool kit for sparse matrix computation (version2), University of Illinois. 1994.
17. Bai Z, Day D, Demmel J, Dongarra J. A test matrix collection for non-Hermitian eigenvalue problems. *Dept. of Mathematics* 1996; 751: 40506–0027.
18. Duff IS, Grimes RG, Lewis JG. Users' guide for the Harwell-Boeing sparse matrix collection (Release I). 1992.
19. Davis TA, Hu Y. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)* 2011; 38(1): 1.
20. Boisvert RF, Pozo R, Remington K, Barrett RF, Dongarra JJ. Matrix market: a web resource for test matrix collections. In: Springer. 1997 (pp. 125–137).

21. Chu MT, Golub GH. Structured inverse eigenvalue problems. *Acta Numerica* 2002; 11: 1–71.

22. Demmel J, McKenney A. A test matrix generation suite. In: Citeseer. ; 1989.

23. Galicher H, Boillod-Cerneux F, Petiton S, Calvin C. Generate Very Large Sparse Matrices Starting from a Given Spectrum. In lecture notes in Computer Science, 8969, Springer (2014).

24. Wu X, Petiton SG, Lu Y. A Parallel Generator of Non-Hermitian Matrices Computed from Given Spectra. In: Senger H, Marques O, Garcia R, et al., eds. *High Performance Computing for Computational Science – VECPAR 2018* Springer International Publishing; 2019; Cham: 215–229.

25. Wu X. *SMG2S Manual v1.0*. Maison de la Simuation, France; 2018.

26. Saad Y, Schultz MH. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing* 1986; 7(3): 856–869.

27. Petiton SG. Parallel subspace method for non-Hermitian eigenproblems on the Connection Machine (CM2). *Applied Numerical Mathematics* 1992; 10(1): 19–35.

28. Wu X, Petiton SG. A Distributed and Parallel Asynchronous Unite and Conquer Method to Solve Large Scale Non-Hermitian Linear Systems. In: HPC Asia 2018. ACM; 2018; New York, NY, USA: 36–46.