Computer Physics Communications 292 (2023) 108888

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Computational Physics

Hybrid eigensolvers for nuclear configuration interaction calculations ☆

Abdullah Alperen [a], Hasan Metin Aktulga [a], Pieter Maris [b], Chao Yang [c,*]

[a] Michigan State University, East Lansing, MI 48824, USA
[b] Dept. of Physics and Astronomy, Iowa State University, Ames, IA 50011, USA
[c] Lawrence Berkeley National Laboratory, CA 94720, USA

ARTICLE INFO

Article history:
Received 11 May 2023
Received in revised form 31 July 2023
Accepted 2 August 2023
Available online 7 August 2023

Keywords:
Nuclear configuration interaction calculation
Large-scale eigenvalue computation
Hybrid eigensolver

ABSTRACT

We examine and compare several iterative methods for solving large-scale eigenvalue problems arising from nuclear structure calculations. In particular, we discuss the possibility of using block Lanczos method, a Chebyshev filtering based subspace iterations and the residual minimization method accelerated by direct inversion of iterative subspace (RMM-DIIS) and describe how these algorithms compare with the standard Lanczos algorithm and the locally optimal block preconditioned conjugate gradient (LOBPCG) algorithm. Although the RMM-DIIS method does not exhibit rapid convergence when the initial approximations to the desired eigenvectors are not sufficiently accurate, it can be effectively combined with either the block Lanczos or the LOBPCG method to yield a hybrid eigensolver that has several desirable properties. We will describe a few practical issues that need to be addressed to make the hybrid solver efficient and robust.

1. Introduction

The computational study of the structure of atomic nuclei involves solving the many-body Schrödinger equation for a nucleus consisting of Z protons and N neutrons, with $A = Z + N$ the total number of nucleons,

$$\hat{H}\,\Psi_i(\vec{r}_1,\ldots,\vec{r}_A) = E_i\,\Psi_i(\vec{r}_1,\ldots,\vec{r}_A)\,, \tag{1}$$

where $\hat{H}$ is the nuclear Hamiltonian, E_i are the discrete energy levels of the low-lying spectrum of the nucleus, and Ψ_i the corresponding A-body wavefunctions. A commonly used approach to address this problem is the no-core Configuration Interaction (CI) method (or No-Core Shell Model) [1], in which the many-body Schrödinger equation, Eq. (1), becomes an eigenvalue problem

$$H\,x_i = \lambda_i\,x_i\,, \tag{2}$$

where H is an $n \times n$ square matrix that approximates the many-body Hamiltonian $\hat{H}$, λ_i is the ith eigenvalue of H, and x_i is the corresponding eigenvector. The size n of the symmetric matrix H grows rapidly with the number of nucleons A and with the desired numerical accuracy, and can easily be several billion or more; however, this matrix is extremely sparse, at least for nuclei with $A \geq 6$. Furthermore, we are typically interested in only a few (5 to 10) eigenvalues at the low end of the spectrum of H. An iterative method that can make use of an efficient Hamiltonian-vector multiplication procedure is therefore often the preferred method to solve Eq. (2) for the lowest eigenpairs.

For a long time, the Lanczos algorithm [2] with full orthogonalization was the default algorithm to use because it is easy to implement and because it is quite robust even though it requires storing hundreds of Lanczos basis vectors. Indeed, there are several software packages [3–8] in which the Lanczos algorithm is implemented for nuclear structure calculations. Here we focus on the software MFDn (Many-Fermion Dynamics for nuclear structure) [9–11], which is a hybrid MPI/OpenMPI code that is being used at several High-Performance Computing centers; it has recently also been ported to GPUs using OpenACC [12,13].

In recent work [14], we have shown that the low-lying eigenvalues can be computed efficiently by using the Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) algorithm [15]. The advantages of the LOBPCG algorithm, which we will describe with some detail in the next section, over the Lanczos algorithm include

- The algorithm is a block method that allows us to multiply H with several vectors simultaneously. That is, instead of an SpMV, one performs an Sparse Matrix-Matrix multiplication (SpMM) of a sparse square $n \times n$ matrix on a tall skinny $n \times n_b$ matrix at every iteration, which introduces an additional level of concurrency in the computation and enables us to exploit

☆ The review of this paper was arranged by Prof. Z. Was.

* Corresponding author.
E-mail address: cyang@lbl.gov (C. Yang).

https://doi.org/10.1016/j.cpc.2023.108888

data locality better. In order to converge 5 to 10 eigenpairs we typically use blocks of $n_b = 8$ to $n_b = 16$ vectors – this can also be tuned to the hardware of the HPC platform.
- The algorithm allows us to make effective use of pre-existing approximations to several eigenvectors.
- The algorithm allows us to take advantage of a preconditioner that can be used to accelerate convergence.
- Other dense linear algebra operations can be implemented as level 3 BLAS.

Even though Lanczos is efficient in terms of the number of Sparse Matrix Vector multiplications (SpMV) it uses, we have shown that the LOBPCG method often takes less wallclock time to run because performing a single SpMM on n_b vectors is more efficient than performing n_b SpMVs sequentially, which is required in the Lanczos algorithm.

However, there are occasionally some issues with LOBPCG:

- The method can become unstable near convergence. Although methods for stabilizing the algorithm has been developed and implemented [16,17], they do not completely eliminate the problem.
- Even though the algorithm in principle only requires storing three blocks of vectors, in practice, many more blocks of vectors are needed to avoid performing additional SpMMs in the Rayleigh-Ritz procedure. This is a problem for machines on which high bandwidth memory is in short supply (such as GPUs).

In this paper, we examine several alternative algorithms for solving large-scale eigenvalue problems in the context of nuclear configuration interaction calculations. In particular, we will examine the block Lanczos algorithm [18] and the Chebyshev Filtered Subspace Iteration (ChebFSI) [19,20]. Both are block algorithms that can benefit from an efficient implementation of the SpMM operation and can take advantage of good initial guesses to several eigenvectors, if they are available. Neither one of these algorithms can incorporate a preconditioner, which is a main drawback. However, as we will show in Sect. 3, in the early iterations of these algorithms, good approximations to the desired eigenpairs emerge quickly, even though the total number of SpMVs required to obtain accurate approximations can be higher compared to the Lanczos and LOBPCG algorithms. This observation suggests that these algorithms can be combined with algorithms that are effective in refining existing eigenvector approximations. One such refinement algorithm is the Residual Minimization Method (RMM) with Direct Inversion of Iterative Subspace (DIIS) correction [21–23]. This algorithm has an additional feature that it can reach convergence to a specific eigenpair without performing orthogonalization against approximations to other eigenpairs as long as a sufficiently accurate initial guess is available. Therefore, this algorithm can also be used to compute (or refine) different eigenpairs independently. This feature introduces an additional level of concurrency in the eigenvalue computation that enhances the parallel scalability.

The paper is organized as follows. In the next section, we give an overview of the Lanczos, block Lanczos, LOBPCG, and ChebFSI algorithms. We also describe the RMM-DIIS algorithm and discuss how it can be combined with (block) Lanczos, LOBPCG and ChebFSI to form a hybrid algorithm to efficiently compute the desired eigenpairs. In Sect. 3, we give several numerical examples to demonstrate the effectiveness of each of these algorithms in terms of the number of iterations. The performance benefits of a hybrid algorithm designed by combining RMM-DIIS with one of the block algorithms are discussed in Sect. 4. We also discuss the practical issue of deciding when switch to RMM-DIIS from block Lanczos or LOBPCG and how to implement RMM-DIIS to maximize its performance benefit.

2. Numerical algorithms

We review several algorithms for computing a few algebraically smallest eigenvalues and the corresponding eigenvectors. We denote the eigenvalues of the $n \times n$ nuclear CI Hamiltonian H arranged in an increasing order by $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. Their corresponding eigenvectors are denoted by x_1, x_2,..., x_n. We are interested in the first $n_{\text{ev}} \ll n$ eigenvalues and eigenvectors. If we define $X = [x_1, x_2, \ldots, x_{n_{\text{ev}}}]$ and $\Lambda = \text{diag}\left\{\lambda_1, \lambda_2, \ldots, \lambda_{n_{\text{ev}}}\right\}$, respectively, we have $HX = X\Lambda$. For no-core CI calculations in nuclear physics we are often interested in only a few eigenvalues, that is $n_{\text{ev}} \sim 5$ to 10. For the block solvers described below, it is generally beneficial to use slightly more vectors than the number of desired eigenvectors, $n_b > n_{\text{ev}}$, and in practice we use $n_b = 8$ or 16 for best performance.

2.1. Lanczos algorithm

The Lanczos algorithm is a classical algorithm for solving large scale eigenvalue problems. The algorithm generates an orthonormal basis of a k-dimensional Krylov subspace

$$\mathcal{K}(H; v_1) = \{v_1, Hv_1, ..., H^{k-1}v_1\}, \tag{3}$$

where v_1 is an appropriately chosen and normalized starting guess. Such a basis is produced by a Gram–Schmidt process in which the key step of obtaining the $(j+1)$st basis vector v_{j+1} is

$$w_j = (I - V_j V_j^T)Hv_j, \qquad v_{j+1} = w_j/\|w_j\|, \tag{4}$$

where V_j is a matrix that contains all previous orthonormal basis vectors, i.e.,

$$V_j = \left(v_1, v_2, ..., v_j\right).$$

The projection of H into the k-dimensional subspace spanned by columns of V_k is a tridiagonal matrix T_k that satisfies

$$HV_k = V_k T_k + w_k e_k^T, \tag{5}$$

where e_k is the last column of a $k \times k$ identity matrix. Approximate eigenpairs of H are obtained by solving the $k \times k$ eigenvalue problem

$$T_k q = \theta q. \tag{6}$$

It follows from (5), (6) and the fact that $V_k^T V_k = I_k$, $V_k^T w_k = 0$ that the relative residual norm associated with an approximate eigenpair $(\theta, V_k q)$ can be estimated by

$$\frac{\|H(V_k q) - \theta(V_k q)\|}{|\theta|} = \frac{\|w_k\| \cdot |e_k^T q|}{|\theta|}, \tag{7}$$

for $\theta \neq 0$.

2.2. Block Lanczos

One of the drawbacks of the standard Lanczos algorithm is that it is not easy for this algorithm to take advantage of good initial guesses to more than one desired eigenvector. Although we can take a simple linear combination (or average) of the initial guesses to several desired eigenvectors as the initial vector v_1, the Lanczos algorithm tends to converge to one of the eigenvectors much faster than others.

An algorithm that can take advantage of multiple starting guesses to different eigenvectors is the Block Lanczos algorithm. The Block Lanczos algorithm generates an orthonormal basis of a block Krylov subspace

$$\mathcal{K}(H; V_1) = \{V_1, HV_1, ..., H^{k-1}V_1\}, \tag{8}$$

where V_1 is a matrix that contains $n_b \geq n_{\text{ev}}$ orthonormal basis vectors, where n_{ev} is the number of desired eigenvectors; in practice, we take n_b slightly larger than n_{ev}. With this method we can make use of good initial guesses to the desired eigenvectors, e.g. obtained in smaller calculations.

The Gram–Schmidt process used to generate an orthonormal basis in Lanczos is replaced by a block Gram–Schmidt step that is characterized by

$$W_j = (I - \mathbf{V}_j\mathbf{V}_j^T)HV_j, \tag{9}$$

where the matrix $\mathbf{V}_j$ contains j block orthonormal bases, i.e.,

$$\mathbf{V}_j = \left(V_1, V_2, ..., V_j\right). \tag{10}$$

The normalization step in (4) is simply replaced by a QR factorization step, i.e.

$$W_j = V_{j+1}R_{j+1},$$

where $V_{j+1}^T V_{j+1} = I_{n_b}$, and R_{j+1} is an $n_b \times n_b$ upper triangular matrix.

The projection of H into the subspace spanned by the columns of $\mathbf{V}_k$ is a block tridiagonal matrix $\mathbf{T}_k$ that satisfies

$$H\mathbf{V}_k = \mathbf{V}_k\mathbf{T}_k + W_k E_k^T, \tag{11}$$

where E_k is the last n_b columns of an $n_b \cdot k \times n_b \cdot k$ identity matrix.

Approximate eigenpairs of H are obtained by solving the $n_b \cdot k \times n_b \cdot k$ eigenvalue problem

$$\mathbf{T}_k q = \theta q. \tag{12}$$

It follows from (11), (12) and the fact that $\mathbf{V}_k^T\mathbf{V}_k = I_{n_b k}$, $\mathbf{V}_k^T W_k = \mathbf{0}$ that the relative residual norm associated with an approximate eigenpair $(\theta, V_k q)$ can be estimated by

$$\frac{\|H(V_k q) - \theta(V_k q)\|}{|\theta|} = \frac{\|W_k\|_F \cdot \|E_k^T q\|}{|\theta|}, \tag{13}$$

Algorithm 1 outlines the main steps of the Block Lanczos algorithm.

Both the Lanczos and Block Lanczos algorithms produce approximation to the desired eigenvector in the form of

$$z = p_d(H)v_0,$$

where v_0 is some starting vector and d is the degree of the polynomial. The convergence rate of these algorithms is often analyzed in terms of a minmax polynomial approximation of a Dirac-δ function centered at the eigenvalue of interest on the interval that contains the spectrum of H [24,25]. Intuitively, the higher the degree of the polynomial, the more accurate the approximate eigenvector and the corresponding eigenvalue are. However, for the same number of multiplications of the sparse matrix H with a vector (SpMVs), denoted by m, the degree of the polynomial generated in a Block Lanczos algorithm is $d = m/n_b$, whereas, in the standard Lanczos algorithm, the degree of the polynomial is $d = m$. Because the accuracy of the approximate eigenpairs obtained from the Lanczos and Block Lanczos methods is directly related to d, we expect more SpMVs to be used in a Block Lanczos algorithm to reach convergence. On the other hand, in a Block Lanczos method, one can perform n_b SpMVs as a single SpMM on a tall skinny matrix consisting of a block of n_b vectors, which is generally more efficient than performing n_b separate SpMVs in succession. As a result, the Block Lanczos method can take less time even if it performs more SpMVs. The computational efficiency of the Block Lanczos algorithm is carefully discussed in [26] in the context of shift-invert Block Lanczos algorithm.

Algorithm 1: The Block Lanczos algorithm.

Input: The sparse matrix H, the number of desired eigenvalues n_{ev}, an initial guess to the eigenvectors associated with the lowest $n_b \geq n_{\text{ev}}$ eigenvalues $X^{(0)} \in \mathbb{R}^{n\times n_b}$, convergence tolerance (tol) and maximum number of iteration allowed ($maxiter$);

Output: (Λ, X), where Λ is a $n_{\text{ev}} \times n_{\text{ev}}$ diagonal matrix containing the desired eigenvalues, and $X \in \mathbb{R}^{n\times n_{\text{ev}}}$ contains the corresponding eigenvector approximations;

Generate $V_1 \in \mathbb{R}^{n\times n_b}$ that contains an orthonormal basis of $X^{(0)}$;
$\mathbf{V}_1 = (V_1)$;
$\mathbf{T}_1 = \mathbf{V}_1^T H\mathbf{V}_1$;
Solve the projected eigenvalue problem $\mathbf{T}_1 U = U\Theta$, where $U^T U = I$, Θ is a diagonal matrix containing eigenvalues of $\mathbf{T}_1$ in an ascending order;
Determine number of converged eigenpairs n_c by checking the Ritz residual estimate (13);
goto 16 if $n_c \geq n_{\text{ev}}$;
do $i = 1, 2, \ldots, maxiter$
  $W_i = (I - \mathbf{V}_i\mathbf{V}_i^T)HV_i$;
  Generate V_{i+1} that contains an orthonormal basis of W_i;
  $\mathbf{V}_{i+1} \leftarrow (\mathbf{V}_i\ V_i)$;
  Update $\mathbf{T}_{i+1} = \mathbf{V}_{i+1}^T H\mathbf{V}_{i+1}$;
  Solve the projected eigenvalue problem $\mathbf{T}_{i+1}U = U\Theta$, where $U^T U = I$, Θ is a diagonal matrix containing eigenvalues of $\mathbf{T}_{i+1}$ in an ascending order;
  $X_{i+1} = \mathbf{V}_{i+1}U(:, 1 : n_{\text{ev}})$;
  Determine number of converged eigenpairs n_c by checking the Ritz residual estimate (13);
  $i \leftarrow i + 1$ and exit the loop if $n_c \geq n_{\text{ev}}$;
$\Lambda \leftarrow \Theta$; $X \leftarrow X_i$;

Furthermore, both with the standard Lanczos algorithm and with the Block Lanczos algorithm the memory to store the previous Lanczos vectors and the computational cost of the Gram–Schmidt process, Eq. (4), increase with the number of iterations. In both the Lanczos and Block Lanczos algorithm, one can restart the algorithm after d iteration with an improved set of approximate eigenvectors if the Gram–Schmidt process or the storage of m Lanczos vectors becomes a bottleneck [27–29,8]. This approach can also be used for check-point and restart purposes.

2.3. LOBPCG

It is well known that the invariant subspace associated with the smallest n_{ev} eigenvalues and spanned by columns of $X \in \mathbb{R}^{n\times n_{\text{ev}}}$ is the solution to the trace minimization problem

$$\min_{X^T X = I} \text{trace}(X^T HX). \tag{14}$$

The LOBPCG algorithm developed by Knyazev [15] seeks to solve (14) by using the updating formula

$$X^{(i+1)} = X^{(i)}C_1^{(i+1)} + W^{(i)}C_2^{(i+1)} + P^{(i-1)}C_3^{(i+1)}, \tag{15}$$

to approximate the eigenvector corresponding to the n_{ev} leftmost eigenvalues of H, where $W^{(i)} \in \mathbb{R}^{n\times n_{\text{ev}}}$ is the preconditioned gradient of the Lagrangian

$$\mathcal{L}(X, \Lambda) = \frac{1}{2}\text{trace}(X^T HX) - \frac{1}{2}\text{trace}\left[(X^T X - I)\Lambda\right] \tag{16}$$

associated with (14) at $X^{(i)}$, and $P^{(i-1)}$ is the search direction obtained in the $(i-1)$st iterate of the optimization procedure, and $C_1^{(i+1)}$, $C_2^{(i+1)}$, $C_3^{(i+1)}$ are a set of coefficient matrices of matching

dimensions that are obtained by minimizing (16) within the subspace $S^{(i)}$ spanned by

$$S^{(i)} \equiv \left(X^{(i)} \;\; W^{(i)} \;\; P^{(i-1)} \right). \tag{17}$$

To improve the convergence of the LOBPCG algorithm, one can include a few more vectors in $X^{(i)}$ so that the number of columns in $X^{(i)}$, $W^{(i)}$ and $P^{(i)}$ is $n_b \geq n_{\text{ev}}$.

The preconditioned gradient $W^{(i)}$ can be computed as

$$W^{(i)} = K^{-1}(HX^{(i)} - X^{(i)}\Theta^{(i)}), \tag{18}$$

where $\Theta^{(i)} = X^{(i)T} H X^{(i)}$, and K is a preconditioner that approximates H in some way. Properties of a good preconditioner are discussed in [30–32]. However, for many applications, it is not easy to find K's that have these properties. When H is diagonally dominant, choosing K to be the diagonal or block diagonal part of H often works well. Other possibilities include incomplete factorizations [33] of $H - \sigma I$ for some appropriately chosen shift σ. Here, we use a preconditioner based on a specific block diagonal part of H that preserves an important symmetry of H, namely the total spin (vector sum of the orbital motion and the intrinsic nucleon spin). We subtract a constant that approximates the smallest eigenvalue of H from the diagonal of this block diagonal matrix. This block diagonal structure arises naturally in the implementation of MFDn; furthermore, this choice leads to a local preconditioner, that is, there is no communication overhead when applying the preconditioner.[1]

The subspace minimization problem that yields the coefficient matrix $C_1^{(i+1)}$, $C_2^{(i+1)}$, $C_3^{(i+1)}$, which are three block rows of a $3n_b \times n_b$ matrix $C^{(i+1)}$, can be solved as a generalized eigenvalue problem

$$\left(S^{(i)T} H S^{(i)} \right) C^{(i+1)} = \left(S^{(i)T} S^{(i)} \right) C^{(i+1)} D^{(i+1)}, \tag{19}$$

where $D^{(i+1)}$ is a $n_b \times n_b$ diagonal matrix containing n_b leftmost eigenvalues of the projected matrix pencil $\left(S^{(i)T} H S^{(i)}, S^{(i)T} S^{(i)} \right)$. The procedure that forms the projected matrices $S^{(i)T} H S^{(i)}$ and $S^{(i)T} S^{(i)}$ and solves the projected eigenvalue problem (19) is often referred to as the *Rayleigh–Ritz* procedure [34]. Note that the summation of the last two terms in (15) represents the search direction followed in the ith iteration, i.e.,

$$P^{(i)} = W^{(i)} C_2^{(i+1)} + P^{(i-1)} C_3^{(i+1)}. \tag{20}$$

Algorithm 2 outlines the main steps of the basic LOBPCG algorithm. The most computationally costly step of Algorithm 2 is the multiplication of H with a set of vectors. Although it may appear that we need to perform such calculations in steps 8 (where the projected matrix $S^{(i)T} H S^{(i)}$ is formed) and 10, the multiplication of H with $X^{(i)}$, $X^{(i+1)}$ and $P^{(i)}$ can be avoided because $HX^{(i+1)}$ and $HP^{(i)}$ satisfy the following recurrence relationships

$$HX^{(i+1)} = HX^{(i)} C_1^{(i+1)} + HW^{(i)} C_2^{(i+1)} + HP^{(i-1)} C_3^{(i+1)}, \tag{21}$$

$$HP^{(i)} = HW^{(i)} C_2^{(i+1)} + HP^{(i-1)} C_3^{(i+1)}. \tag{22}$$

Therefore, the only SpMM we need to perform is $HW^{(i)}$. Again, for the nuclear CI calculations of interest, the dimension n of the sparse symmetric matrix H can be several billions, whereas $W^{(i)}$ is a tall skinny $n \times n_b$ matrix with n_b typically of the order of 8 to 16.

[1] Note that other CI codes for nuclear structure calculations generally have a different ordering of the basis states, which may obfuscate this block diagonal structure.

Algorithm 2: The basic LOBPCG algorithm.

Input: The sparse matrix H, a preconditioner K, an initial guess to the eigenvectors associated with the lowest $n_b \geq n_{\text{ev}}$ eigenvalues $X^{(0)} \in \mathbb{R}^{n \times n_b}$, number of desired eigenvalues (n_{ev}), convergence tolerance (tol) and maximum number of iteration allowed ($maxiter$);

Output: (Λ, X), where Λ is a $n_{\text{ev}} \times n_{\text{ev}}$ diagonal matrix containing the desired eigenvalues, and $X \in \mathbb{R}^{n \times n_{\text{ev}}}$ contains the corresponding eigenvector approximations;

1 $[C^{(1)}, \Theta^{(1)}] = \texttt{RayleighRitz}(H, X^{(0)})$;
2 $X^{(1)} = X^{(0)} C^{(1)}$;
3 $R^{(1)} = HX^{(1)} - X^{(1)}\Theta^{(1)}$;
4 $P^{(0)} = \emptyset$;
5 **do** $i = 1, 2, \ldots,$*maxiter*
6 $\quad W^{(i)} = K^{-1} R^{(i)}$;
7 $\quad S^{(i)} = \left[X^{(i)}, W^{(i)}, P^{(i-1)} \right]$;
8 $\quad [C^{(i+1)}, \Theta^{(i+1)}] = \texttt{RayleighRitz}(H, S^{(i)})$;
9 $\quad X^{(i+1)} = S^{(i)} C^{(i+1)}$;
10 $\quad R^{(i+1)} = HX^{(i+1)} - X^{(i+1)}\Theta^{(i+1)}$;
11 $\quad P^{(i)} = W^{(i)} C_2^{(i+1)} + P^{(i-1)} C_3^{(i+1)}$;
12 $\quad$ Determine number of converged eigenpairs n_c by comparing the relative norms of the leading n_{ev} columns of $R^{(i+1)}$ against the convergence tolerance tol;
13 $\quad$ exit if $n_c \geq n_{\text{ev}}$;
14 $\Lambda \leftarrow \Theta^{(i)}(1 : n_{\text{ev}}, 1 : n_{\text{ev}})$; $X \leftarrow X^{(i)}(:, 1 : n_{\text{ev}})$;

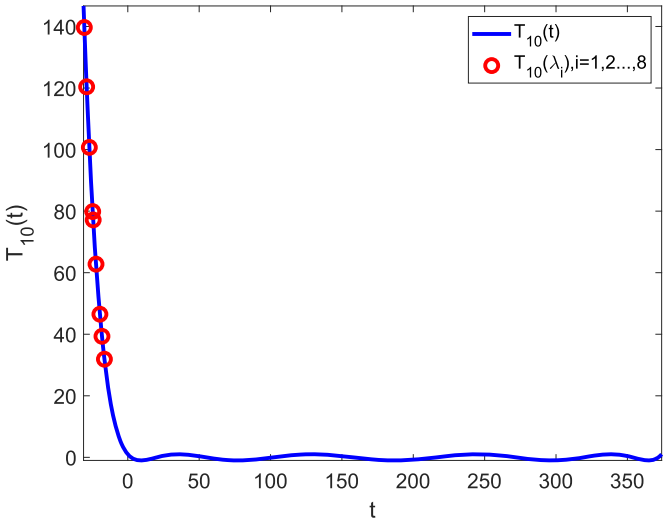

Fig. 1. Chebyshev polynomials of the first kind.

2.4. Chebyshev filtering

An mth-degree Chebyshev polynomial of the first kind can be defined recursively as

$$T_m(t) = 2tT_{m-1}(t) - T_{m-2}(t), \tag{23}$$

with $T_0(t) = 1$ and $T_1(t) = t$. The magnitude of $T_m(t)$ is bounded by 1 within $[-1, 1]$ and grows rapidly outside of this interval. By mapping the unwanted eigenvalues of the nuclear many-body Hamiltonian H enclosed by $[\lambda_{\text{F}}, \lambda_{\text{ub}}]$ to $[-1, 1]$ through the linear transformation $(t - c)/e$, where $c = (\lambda_{\text{F}} + \lambda_{\text{ub}})/2$ and $e = (\lambda_{\text{ub}} - \lambda_{\text{F}})/2$, we can use $\hat{T}_m(H) = T_m((H - cI)/e)v$ to amplify the eigenvector components in v that correspond to eigenvalues outside of $[\lambda_{\text{F}}, \lambda_{\text{ub}}]$. Fig. 1 shows a 10th degree Chebyshev polynomial defined on the spectrum of a Hamiltonian matrix and how the leftmost eigenvalues λ_i, $i = 1, 2, \ldots, 8$ are mapped to $T_{10}(\lambda_i)$.

Applying $T_m((H - cI)/e)$ repeatedly to a block of vectors V filters out the eigenvectors associated with eigenvalues in $[\lambda_{\text{F}}, \lambda_{\text{ub}}]$. The desired eigenpairs can be obtained through the standard Rayleigh–Ritz procedure [34].

Algorithm 3: The Chebyshev filtering based subspace iteration (ChebFSI).

Input: The sparse matrix H, an initial guess to the eigenvectors associated with the lowest $n_b \geq n_{ev}$ eigenvalues $X^{(0)} \in \mathbb{R}^{n \times n_b}$, number of desired eigenvalues (n_{ev}), the degree of the Chebyshev polynomial d; the spectrum cutoff λ_F and upper bound λ_{ub}; convergence tolerance (tol) and maximum number of subspace iteration allowed ($maxiter$);

Output: (Λ, X), where Λ is a $n_{ev} \times n_{ev}$ diagonal matrix containing the desired eigenvalues, and $X \in \mathbb{R}^{n \times n_{ev}}$ contains the corresponding eigenvector approximations;

1 $e = (\lambda_{ub} - \lambda_F)/2$;
2 $c = (\lambda_{ub} + \lambda_F)/2$;
3 $[Q, R] = \texttt{CholeskyQR}(H, X^{(0)})$;
4 **do** $i = 1, 2, \ldots$, *maxiter*
5 $\quad W = 2(HQ - cQ)/e$;
6 $\quad$ **do** $j = 2, \ldots, d$
7 $\quad\quad Y = 2(HW - cW)/e - Q$;
8 $\quad\quad Q \leftarrow W$;
9 $\quad\quad W \leftarrow Y$;
10 $\quad [Q, R] = \texttt{CholeskyQR}(Y)$;
11 $\quad T = Q^T H Q$;
12 $\quad$ Solve the eigenvalue problem $TS = S\Theta$, where Θ is diagonal, and update Q by $Q \leftarrow QS$;
13 $\quad$ Determine number of converged eigenpairs n_c by comparing the relative norms of the leading n_{ev} columns of $R = HQ - QD$ against the convergence tolerance tol;
14 $\quad$ exit if $n_c \geq n_{ev}$;
15 $\Lambda \leftarrow \Theta(1 : n_{ev}, 1 : n_{ev})$; $X \leftarrow Q(:, 1 : n_{ev})$;

To obtain an accurate approximation to the desired eigenpairs, a high degree Chebyshev polynomial may be needed. Instead of applying a high degree polynomial once to a block of vectors, which can be numerically unstable, we apply Chebyshev polynomial filtering within a subspace iteration to iteratively improve approximations to the desired eigenpairs. We will refer to this algorithm as a Chebyshev Filtering based Subspace Iteration (ChebFSI). The basic steps of this algorithm are listed in Algorithm 3.

Owing to the three-term recurrence in (23), $W = \hat{T}_m(H)V$ can be computed recursively without forming $\hat{T}_m(H)$ explicitly in advance. Lines 5 to 9 of Algorithm 3 illustrates how this step is carried out in detail. To maintain numerical stability, we orthonormalize vectors in W. The orthonormalization can be performed by a (modified) Gram–Schmidt process or by a Householder transformation based QR factorization [35].

In Algorithm 3, the required inputs are a filter degree, d, an estimated upper bound of the spectrum of H, λ_{ub}, and an estimated spectrum cutoff level, λ_F. The estimation of the upper bound λ_{ub} can be calculated by running a few Lanczos iterations [36–38], and λ_F can often be set at 0 or an estimation of the $n_b + 1$st leftmost eigenvalue of H obtained from the Lanczos algorithm. In the subsequent subspace iterations, λ_F can be modified based on more accurate approximations to the desired eigenvalues. See the work of Saad [19] and Zhou *et al.* [20] for more details on Chebyshev filtering.

2.5. RMM-DIIS

The Residual Minimization Method (RMM) [21,22] accelerated by Direct Inversion of Iterative Subspace (DIIS) [23] was developed in the electronic structure calculation community to solve a linearized Kohn-Sham eigenvalue problem in each self-consistency field (SCF) iteration. Given a set of initial guesses to the desired eigenvectors, $\{x_j^0\}$, $j = 1, 2, \ldots, n_{ev}$, the method produces successively more accurate approximations by seeking an optimal linear combination of previous approximations to the jth eigenvector by minimizing the norm of the corresponding sum of residuals. To be specific, let $x_j^{(i)}$, $i = 0, 1, \ldots, \ell - 1$ be approximations to the jth eigenvector of H obtained in the previous $\ell - 1$ steps of the RMM-DIIS algorithm, and $\theta_j^{(i)}$ be the corresponding eigenvalue approximations. In the ℓth iteration (for $\ell > 1$), we first seek an approximation in the form of

$$\tilde{x}_j = \sum_{i=\min\{0,\ell-s\}}^{\ell-1} \alpha_i x_j^{(i)}, \tag{24}$$

where

$$\sum_{i=\ell-s}^{\ell-1} \alpha_i = 1, \tag{25}$$

for some fixed $1 \leq s \leq s_{max}$. The coefficients α_i's are obtained by solving the following constrained least squares problem

$$\min \| \sum_{i=\min\{0,\ell-s\}}^{\ell-1} \alpha_i r_j^{(i)} \|^2, \tag{26}$$

where $r_j^{(i)} = Hx_j^{(i)} - \theta_j^{(i)} x_j^{(i)}$ is the residual associated with the approximate eigenpair $(\theta_j^{(i)}, x_j^{(i)})$, subject to the same constraint defined by (25). The constrained minimization problem can be turned into an unconstrained minimization problem by substituting $\alpha_{\ell-1} = 1 - \sum_{i=\min\{\ell-s\}}^{\ell-2} \alpha_i$ into (26).

Once we solve (26), we compute the corresponding residual

$$\tilde{r}_j = H\tilde{x}_j - \tilde{\theta}_j \tilde{x}_j,$$

where $\tilde{\theta}_j = \langle \tilde{x}_j, H\tilde{x}_j \rangle / \langle \tilde{x}_j, \tilde{x}_j \rangle$. A new approximation to the desired eigenvector is obtained by projecting H into the two-dimensional subspace W_j spanned by $\tilde{x}_j$ and $\tilde{r}_j$, and solving the 2×2 generalized eigenvalue problem

$$(W_j^T H W_j) g = \theta (W_j^T W_j) g. \tag{27}$$

Such an approximation can be written as

$$x_j^{(\ell)} = W_j g, \tag{28}$$

where g is the eigenvector associated with the smaller eigenvalue of the matrix pencil $(W_j^T H W_j, W_j^T W_j)$.

Although Rayleigh–Ritz procedures defined by (27) and (28) are often used to compute the lowest eigenvalue of H, the additional constraint specified by (24) and (25) keeps $x_j^{(\ell)}$ close to the initial guess of the jth eigenvector. Therefore, if the initial guess is sufficiently close to the jth eigenvector, $x_j^{(\ell)}$ can converge to this eigenvector instead of the eigenvector associated with the smallest eigenvalue of H. Algorithm 4 outlines the main steps of the RMM-DIIS algorithm.

2.6. Comparison summary

The computational cost of all iterative methods discussed above is dominated by the number of Hamiltonian matrix vector multiplications, that is, the number of SpMVs. In the Lanczos algorithm, the number of SpMVs is the same as the number of iterations. In block algorithms such as the Block Lanczos algorithm and the LOBPCG algorithm, the number of SpMVs is the product of the block size, n_b, and the number of iterations. The number of SpMVs used in the ChebFSI method is the product of the number of subspace iterations, the block size n_b, and the degree d of the Chebyshev polynomial used. The number of SpMVs used in RMM-DIIS is the sum of the RMM-DIIS iterations for each of the n_{ev} desired eigenpairs; depending on the architecture, one could combine n_{ev} SpMVs on single vectors in one SpMM on a tall skinny $n \times n_{ev}$ matrix to improve performance.

Algorithm 4: The RMM-DIIS algorithm.

Input: The sparse matrix H, an initial guess to the n_{ev} desired eigenvectors $\{x_j^{(0)}\}$, $j = 1, 2, \ldots n_{\text{ev}}$; maximum dimension of DIIS subspace s; convergence tolerance (tol) and maximum number of iteration allowed ($maxiter$);

Output: $\{(\theta_j, x_j)\}$, $j = 1, 2, \ldots, n_{\text{ev}}$, where θ_j is the approximation to the jth lowest eigenvalue, and x_j is the corresponding approximate eigenvector ;

for *j=1,2,…,*n_{ev} **do**
- $x_j^{(0)} \leftarrow x_j^{(0)} / \|x_j^{(0)}\|$;
- $\theta^{(0)} = \langle x_j^{(0)}, Hx_j^{(0)} \rangle$;
- $r_j^{(0)} = Hx_j^{(0)} - \theta_j^{(0)} x_j^{(0)}$;
- $\tilde{x}_j^{(1)} = x_j^{(0)}$;
- $\tilde{r}_j^{(1)} = r_j^{(0)}$;
- **do** $i = 1, 2, \ldots,$*maxiter*
  - **if** $i > 1$ **then**
    - Solve the residual minimization least squares problem (26);
    - Set $\tilde{x}_j^{(i)}$ according to (24);
    - $\tilde{x}_j^{(i)} \leftarrow \tilde{x}_j^{(i)} / \|\tilde{x}_j^{(i)}\|$;
    - Set the residual $\tilde{r}_j^{(i)} = \sum_{\ell=\min\{0,i-s\}}^{i-1} \alpha_i x_j^{(\ell)}$;
  - Set $W_j = (\tilde{x}_j^{(i)}, \tilde{r}_j^{(i)})$;
  - Solve the Rayleigh–Ritz problem (27) and obtain $(\theta_j^{(i)}, x_j^{(i)})$;
  - Compute the residual $r_j^{(i)} = Hx_j^{(i)} - \theta_j^{(i)} x_j^{(i)}$;
  - exit the do loop if $\|r_j^{(i)}\| / |\theta_j^{(i)}| < tol$;

Table 1
A comparison of memory footprint associated with the Lanczos, Block Lanczos, LOBPCG, ChebFSI and RMM-DIIS methods.

Method	Memory cost
Lanczos	$n \cdot (k_{\text{Lan}} + n_{\text{ev}}) + \mathcal{O}(k_{\text{Lan}}^2)$
Block Lanczos	$n \cdot n_b \cdot (k_{\text{blockLan}} + n_{\text{ev}}) + \mathcal{O}\left((n_b \cdot k_{\text{blockLan}})^2\right)$
LOBPCG	$7n \cdot n_b + \mathcal{O}(9n_b^2)$
ChebFSI	$4n \cdot n_b + \mathcal{O}(n_b^2)$
RMM-DIIS	$n \cdot (3n_{\text{ev}} + s_{\max})$

In addition to SpMVs, some dense linear algebra operations are performed in these algorithms to orthonormalize basis vectors and to perform the Rayleigh–Ritz calculations. The cost of orthonormalization can become large if too many Lanczos iterations or Block Lanczos iterations are performed; for the Lanczos and Block Lanczos algorithm once can perform a restart once the orthonormalization cost becomes too large. The orthonormalization cost is relatively small in LOBPCG, ChebFSI, and RMM-DIIS.

In Table 1, we compare the memory usage of each method discussed above. Note that the first term for Lanczos, Block Lanczos, LOBPCG and ChebFSI in this table is generally the dominant term. We use $\mathcal{O}(c)$ to denote a small multiple (i.e., typically 2 or 3) of c. The number of iterations taken by a Block Lanczos iteration k_{blockLan} is typically smaller than the number of Lanczos iterations k_{Lan} when the same number of eigenpairs are computed by these methods. However, $k_{\text{blockLan}} \cdot n_b$ is often larger than k_{Lan}. It is possible to use a smaller amount of memory in LOBPCG and ChebFSI at the cost of performing more SpMMs. For example, if we were to explicitly compute $HX^{(i+1)}$ and $HP^{(i)}$ in the LOBPCG algorithm to perform the Rayleigh–Ritz calculation instead of updating these blocks according to (21) and (22) respective, we can reduce the LOBPCG memory usage to $4n \cdot n_b + \mathcal{O}(9n_b^2)$. For the RMM-DIIS algorithm, we assume that we compute one eigenpair at a time. The parameter $s_{\max}$ is the maximum dimension of the DIIS subspace constructed to correct an approximate eigenvector. This parameter is often chosen to be between 10 and 20. If we batch the refinement of several eigenvectors together to make use of SpMMs, the memory cost of RMM-DIIS will increase by a factor of n_{ev}.

2.7. Hybrid algorithms

Among the methods discussed above, the Lanczos, Block Lanczos, LOBPCG, and ChebFSI methods can all proceed with an arbitrary starting guess of the desired eigenvectors although all, except the Lanczos algorithm, can (and generally do) benefit from the availability of good starting guesses to several eigenvectors. On the other hand, as an eigenvector refinement method, the RMM-DIIS method requires a reasonably accurate approximation of the desired eigenvectors as a starting point. Therefore, a more effective way to use the RMM-DIIS method is to combine it with one of the other methods, i.e., we can start with Lanczos, Block Lanczos, LOBPCG or ChebFSI method and switch to RMM-DIIS when the approximate eigenvectors become sufficiently accurate.

In particular, in the Lanczos and Block Lanczos methods the basis orthogonalization cost as well as the memory requirement become progressively higher with increasing number of iterations. Therefore, a notable benefit to switch from the Lanczos or Block Lanczos methods to RMM-DIIS is to lower the orthogonalization cost and memory requirement.

Although the orthogonalization cost and memory requirement for the LOBPCG method is fixed throughout all LOBPCG iterations, the subspace (17) from which eigenvalue and eigenvector approximations are drawn becomes progressively more ill-conditioned as the norms of the vectors in (18) become smaller. The ill-conditioned subspace can make the LOBPCG algorithm numerically unstable even after techniques proposed in [16,17] are applied. Therefore, it may be desirable to switch from LOBPCG to RMM-DIIS, when the condition number of the subspace is not too large.

The orthogonalization cost and memory requirement for ChebFSI are also fixed. The method is generally more efficient in the early subspace iterations when $T_n(H)$ is applied to a block of vectors in each iteration. However, as the approximate eigenvectors converge, applying $T_n(H)$ to a block of vectors in a single iteration results in a higher cost compared to RMM-DIIS that can refine each approximate eigenvector separately. Again, it may become advantageous to switch to RMM-DIIS, when approximate eigenvectors become sufficiently accurate in ChebFSI.

3. Numerical examples

In this section, we compare and analyze the performance of each of the five algorithms presented in Section 2 using numerical examples, in terms of the number of SpMVs that are needed to achieve the requested tolerance for the lowest n_{ev} eigenpairs. The initial tests were done using MATLAB, and we tested the effectiveness of algorithms in terms of the number of SpMVs required to reach the requested tolerance for n_{ev} eigenpairs. Subsequently, we have also implemented these algorithms in Fortran90 and experimented with these algorithms in MFDn.

3.1. Test problems

The test problems we use are the many-body Hamiltonian matrices associated with four different nuclei, ^{6}Li, ^{7}Li, ^{11}B, and ^{12}C, where the superscripts indicate the total number of nucleons (protons plus neutrons) in the nuclei. These Hamiltonian matrices are constructed in different CI model spaces labeled by the $N_{\max}$ parameter, using the two-body potential Daejeon16 [39]. In Table 2, we list the matrix size n as well as the number of nonzero matrix elements in half the symmetric matrices. Note that the matrix size n depends on A (the number of nucleons) and the basis truncation parameter $N_{\max}$, but is independent of the interaction. The number of nonzero matrix elements for a given nucleus and interaction is the same for any two-body interaction; but with three-body interactions, the number of nonzero matrix elements is more than

Table 2
Test problems used in the numerical experiments.

Nucleus	$N_{\max}$	Matrix size n	# Non-zeros
^{6}Li	6	197,822	106,738,802
^{7}Li	6	663,527	421,938,629
^{11}B	4	814,092	389,033,682
^{12}C	4	1,118,926	555,151,572

Table 3
The dimensions and number of nonzero matrix elements in half the matrices for ^{6}Li, ^{7}Li, ^{11}B and ^{12}C that are constructed in a lower dimensional configuration model space labeled by a smaller $N_{\max}$ value.

System	$N_{\max}$	Matrix size n	#Non-zeros
^{6}Li	4	17,040	4,122,448
^{7}Li	4	48,917	14,664,723
^{11}B	2	16,097	2,977,735
^{12}C	2	17,725	3,365,099

an order of magnitude larger for the same matrix size. Also, the number of iterations needed in any of the iterative solvers will generally depend on the interaction (as well as the nucleus and the truncation).

Before solving eigenvalue problems for the nuclei and model spaces listed in Table 2, we first construct a good initial guess for each of the n_{ev} desired eigenvectors by computing the lowest few eigenvalues and the corresponding eigenvectors of smaller Hamiltonian matrices constructed from a lower dimensional CI model space labeled by $N_{\max} - 2$ values. Table 3 shows the matrix dimensions n and the number of nonzero matrix elements in these smaller Hamiltonians. The initial guesses to the desired eigenvectors of the Hamiltonian matrices listed in Table 2 are obtained by padding the eigenvectors of the smaller Hamiltonian matrices by zeros to match the dimension of the original problems to be solved. As we can see, since the dimension of the problems listed in Table 3 are an order of magnitude or two smaller than the corresponding problems listed in Table 2, they can be solved relatively easily and quickly by almost any method.

All algorithms presented in section 2 have been implemented in MATLAB which is ideal for prototyping new algorithms. The preconditioner we use in each LOBPCG run is choosen to be $K = H_D - \sigma I$, where H_D is a block diagonal matrix obtained from the diagonal blocks of H, and σ is chosen to be an approximation to the smallest eigenvalue of H obtained from a few iterations of a LOBPCG run without preconditioning. We do not explicitly form the inverse of K. The application of K^{-1} in (18) is achieved by solving linear systems of equations

$$KW^{(i)} = HX^{(i)} - X^{(i)}\Theta^{(i)}$$

using an iterative solver such as the MINRES [40] or the formal orthogonal method [41].

For each test problem, we typically perform two sets of experiments for each algorithm. In the first set of experiments, we compute $n_{\text{ev}} = 5$ lowest eigenvalues and their corresponding eigenvectors. In the second set, we increase the number of eigenpairs to be computed to $n_{\text{ev}} = 10$. All calculations are performed in double precision arithmetic.

3.2. The performance of single method solvers

In this section, we report and compare the performance of the Lanczos, block Lanczos, LOBPCG, ChebFSI and RMM-DIIS methods when they are applied to the test problems listed in Table 2. For block methods such as the block Lanczos, LOBPCG and ChebFSI methods, we set the block size, i.e., the number of vectors in the matrix $\mathbf{V}_j$ in (9), the matrix $X^{(i)}$ in (15), to $n_b = 8$ when computing the $n_{\text{ev}} = 5$ lowest eigenpairs of H, or to $n_b = 16$ when computing the $n_{\text{ev}} = 10$ lowest eigenpairs of H, and an SpMM is performed to multiply H with n_b vectors all at once, rather than performing n_{ev} separate SpMVs. Even though RMM-DIIS is not a block method, the n_{ev} SpMVs performed in this algorithm can also be fused together as a single SpMM as we explain below.

For block methods, we choose the starting guess for each method as the eigenvectors of the Hamiltonian constructed in a smaller CI space (with a smaller $N_{\max}$ value), as listed in Table 3, padded with zeros to match the size of the Hamiltonian in the larger CI space (with a larger $N_{\max}$ value) as mentioned earlier. This is also used in the RMM-DIIS method which only requires a starting guess for each of the desired eigenpairs. We should note that when such a starting guess is not sufficiently close to the desired eigenvector associated with the larger $N_{\max}$ value, the convergence of the RMM-DIIS method can be slow as we will see from the numerical examples presented below. Note that it is also possible that the RMM-DIIS algorithm converges to a set of eigenpairs that do not correspond to the lowest n_{ev} eigenvalues of H.

For the Lanczos algorithm, we take the initial guess v_0 to be the linear combination of augmented eigenvectors associated with the lowest n_{ev} eigenvalues of the Hamiltonian constructed from the smaller CI space, i.e.,

$$v_0 = \frac{1}{n_{\text{ev}}} \sum_{i=1}^{n_{\text{ev}}} \hat{z}_i \,,$$

where $\hat{z}_i$ is the zero padded eigenvector associated with the ith eigenvalue of the Hamiltonian constructed from the smaller CI space.

All methods are terminated when the relative residual norms or estimated residual norm associated with all desired eigenpairs are below the threshold of $\tau = 10^{-6}$. A relative residual norm for an approximate eigenpair (θ, z) is defined to be

$$\frac{|Hz - z\theta|}{|\theta|} \,.$$

For the Lanczos and block Lanczos methods, we use (7) and (13) to estimate the relative residual norm without performing additional Hamiltonian matrix and vector multiplications.

The convergence of the ChebFSI algorithm depends on the choice of several parameters. Here, we use a 10th degree Chebyshev polynomial, i.e. $d = 10$ in the ChebFSI method. The upper bound of the spectrum λ_{ub} is determined by first running 10 Lanczos iterations and using Rayleigh–Ritz approximation to the largest eigenpairs (θ_{10}, u_{10}) to set λ_{ub} to $\theta_{10} + \|r_{10}\|$, where $r_{10} = Hu_{10} - \theta_{10}u_{10}$. We set the parameter λ_F simply to 0 because the desired eigenvalues are bound states of the nucleus of interest and are expected to be negative. We apply the technique of deflation for converged eigenvectors, i.e., once the relative residual norm of an approximate eigenpair falls below the convergence tolerance of 10^{-6}, we "lock" the approximate eigenvector in place and do not apply H to this vector in subsequent computations. These vectors will still participate in the Rayleigh–Ritz calculation performed in steps 11 and 12 of Algorithm 3 and be updated as part of the Rayleigh–Ritz procedure.

In Tables 4 and 5, we compare the performance of Lanczos, block Lanczos, LOBPCG, ChebFSI and RMM-DIIS in terms of the total number SpMVs performed in each of these methods. It is clear from these tables that the Lanczos method uses the least number of SpMVs. However, the number of SpMVs used by both the LOBPCG, block Lanczos and RMM-DIIS is within a factor of 3 when $n_{\text{ev}} = 5$ eigenpairs are computed. Because 8 SpMVs can be fused as a single SpMM, which is more efficient, in the block Lanczos and LOBPCG method, the total wall clock time used by these methods

Table 4
SpMV count for different algorithms on MATLAB to compute five lowest eigenvalues.

System	Lanczos	Block Lanczos	LOBPCG	ChebFSI	RMM-DIIS
^{6}Li	95	208	184	480	174
^{7}Li	109	280	240	960	291
^{11}B	82	240	192	950	152
^{12}C	106	248	192	890	181

Table 5
SpMV count for different algorithms on MATLAB to compute ten lowest eigenvalues.

System	Lanczos	Block Lanczos	LOBPCG	ChebFSI	RMM-DIIS
^{6}Li	114	464	464	690	686
^{7}Li	192	512	464	1,350	884
^{11}B	180	480	432	2,470	$> 3,000$
^{12}C	164	480	400	2,300	499

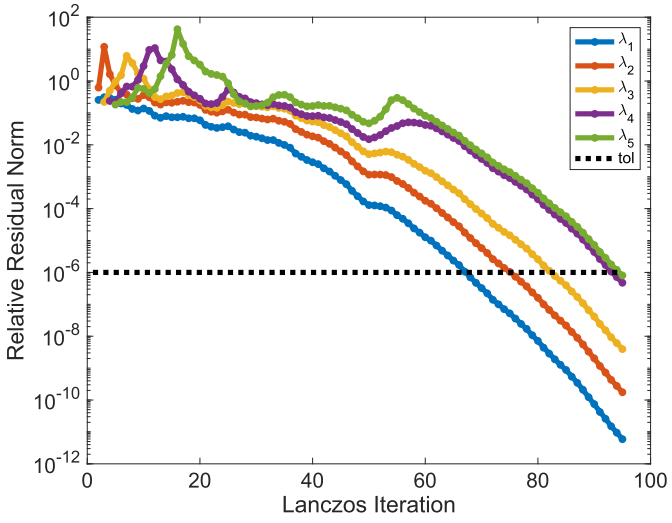

Fig. 2. The convergence of the lowest five eigenvalues of the ^{6}Li Hamiltonian in the Lanczos algorithm as function of the number of iterations, which equals the number of SpMVs.

can be less than that used by the Lanczos method. Five SpMVs can also be fused in the RMM-DIIS method, even though the algorithm targets each eigenvalue separately. Because different eigenvalues may converge at a different rate, we may switch to using SpMVs when some of the eigenpairs converge. Whether it is beneficial to make such a switch depends on the performance difference between SpMM and SpMV, which may be architecture dependent. We will discuss this issue more in the next section.

Table 5 shows that the number of SpMVs used in the Lanczos algorithm increases only slightly when we compute $n_{ev} = 10$ eigenpairs. However, the number of SpMVs required in the block Lanczos, LOBPCG, ChebFSI and RMM-DIIS increase at a higher rate. This is mainly due to the fact that once a sufficiently large Krylov subspace has been constructed, we can easily obtain approximations to more eigenpairs without enlarging the subspace much further. Furthermore, RMM-DIIS fails to converge for the 7th, 8th, and 9th eigenpairs of ^{11}B. Inspection of the obtained spectra with the other methods reveals that these three eigenvalues for this particular case are near-degenerate. Specifically, the eigenvalues of the 7th and 8th state differ by 0.6% and these two states have the same conserved quantum numbers so they can easily mix; while the 8th and 9th states do have different quantum numbers, but their eigenvalues differ by only 0.03%. Indeed, it should not be surprising that refining individual eigenpairs may fail to converge for near-degenerate states.

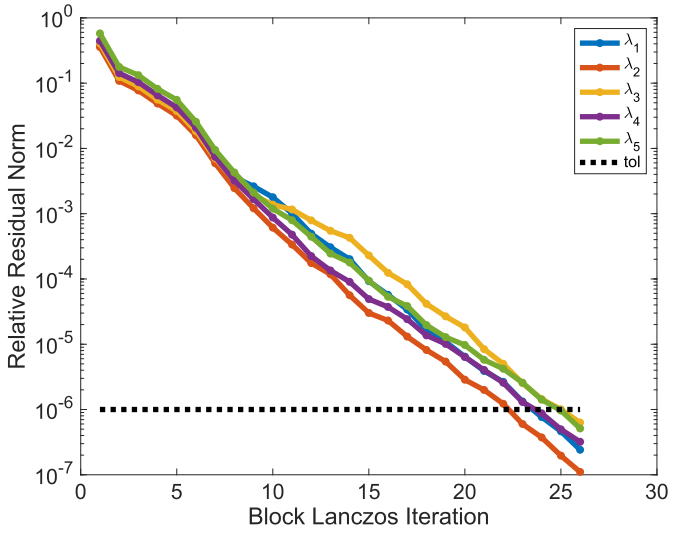

Fig. 3. The convergence of the lowest five eigenvalues of the ^{6}Li Hamiltonian in the block Lanczos algorithm; the number of SpMVs is $n_b = 8$ times the number of iterations.

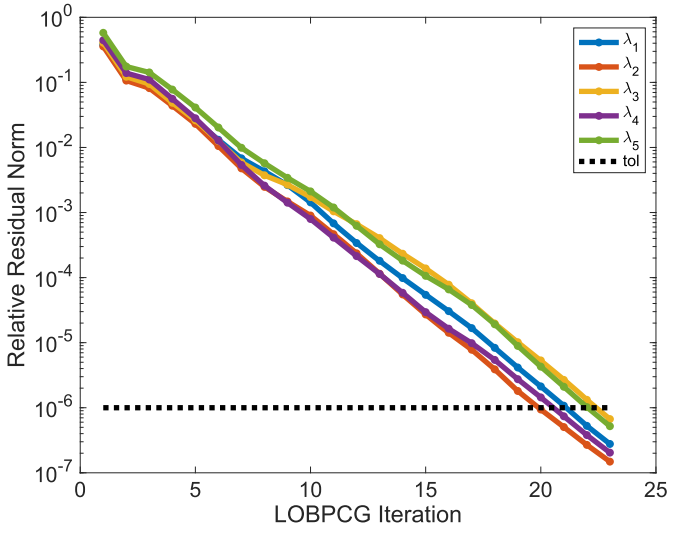

Fig. 4. The convergence of the lowest five eigenvalues of the ^{6}Li Hamiltonian in the LOBPCG algorithm; the number of SpMVs is $n_b = 8$ times the number of iterations.

Figs. 2, 3, 4, 5, 6 show the convergence history of the Lanczos, block Lanczos, LOBCPG, ChebFSI and RMM-DIIS methods for ^{6}Li. In these figures, we plot the relative residual norm of each approximate eigenpair with respect to the iteration number. We can clearly see that accurate approximations to some of the eigenpairs start to emerge in the Lanczos method when the dimension of the Krylov subspace (i.e., iteration number) is sufficiently large. Note that the eigenpairs do not converge at the same rate. Generally, the smallest eigenvalue converges first, followed by the second, third, fourth and the fifth eigenvalues. However, these eigenvalues do not necessarily have to converge in order. Although the relative residual for each eigenpair eventually goes below the convergence threshold of 10^{-6}, the reduction of the relative residual norm is not monotonic with respect to the Lanczos iteration number. The relative residual can sometimes increase after the Krylov subspace becomes sufficiently large and new spectral information becomes available.

All approximate eigenpairs appear to converge at a similar rate in the block Lanczos and LOBPCG methods. This is one of the advantages of a block method. The LOBPCG method performs slightly

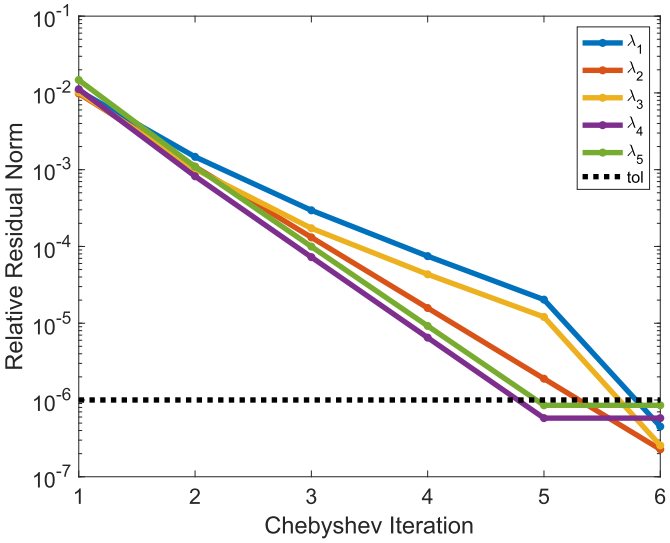

Fig. 5. The convergence of the lowest five eigenvalues of the ^{6}Li Hamiltonian in the Chebyshev algorithm. The number of SpMVs is $n_b = 8$ times the degree of the Chebyshev polynomial, $d = 10$, times the number of iterations.

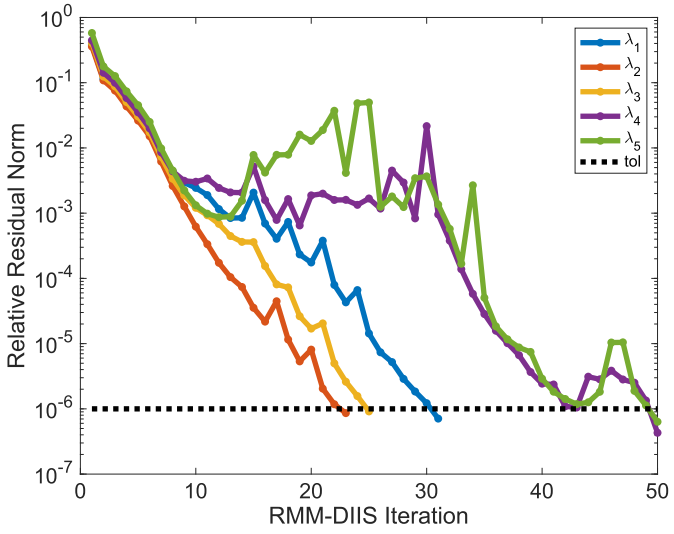

Fig. 6. The convergence of the lowest five eigenvalues of the ^{6}Li Hamiltonian in the RMM-DIIS algorithm. The number of SpMVs is the sum of the number of iterations for each eigenvalue.

better in terms of the total number of SpMMs (or the number of iterations) used. This is due to the fact that LOBPCG makes use of an effective preconditioner.

The ChebFSI is also a block method. Fig. 5 shows that only five subspace iterations are required to obtain converged λ_2, λ_4 and λ_5, and two more subspace iterations are required to obtain converged λ_1 and λ_3. The difference in the convergence rates for different eigenvalues is likely due to the variation in the contributions from different eigenvectors in the initial subspace constructed from the eigenvectors of the Hamiltonian associated with a smaller configuration space. This could also be related to differences in the internal structure of the different states. Although the number of subspace iterations used in ChebFSI is relatively small, each subspace iteration needs to perform $n_b \cdot d$ SpMVs, where n_b is the number of vectors in the initial subspace and d is the degree of the Chebyshev polynomial. When $n_b = 8$ and $d = 10$, a total of 480 SpMVs are used to compute the lowest five eigenpairs as reported in Table 4. Note that this count is less than $8 \times 10 \times 7 = 560$ because a deflation scheme that locks the converged eigenvector is used in ChebFSI. When $n_b = 16$, $d = 10$, a total of 690 SpMVs are used to compute 10 lowest eigenpairs, as reported in Table 5. Because these SpMV counts are significantly higher than those used in other methods, ChebFSI appears to be not competitive for solving this type of eigenvalue problem. Therefore, from this point on, we will not discuss this method any further.

The convergence of the RMM-DIIS method is interesting. We observe from Fig. 6 that the first three eigenvalues of the ^{6}Li Hamiltonian converge relatively quickly. The number of RMM-DIIS iterations required to reach convergence is 31 for the first eigenvalue, 25 for the third eigenvalue and 23 for the second eigenvalue. Altogether, 79 SpMVs are used to obtain accurate approximations to the three smallest eigenvalues and their corresponding eigenvectors, which is almost same as the 78 SpMVs used in the Lanczos algorithm for obtaining these three eigenpairs. However, the fourth and fifth eigenvalues take much longer to converge. Yet, it is important to note that RMM-DIIS iterations that start with different initial guesses all converge to different eigenpairs even though no orthogonalization is performed between approximate eigenvectors produced in different RMM-DIIS iterations. Table 5 shows that more RMM-DIIS iterations are needed to obtain accurate approximations to larger eigenvalues deeper inside the spectrum of ^{6}Li Hamiltonian, partly due to the near-degeneracy of these eigenvalues in this case.

4. Performance of hybrid algorithms

As we already discussed in Section 2, the RMM-DIIS algorithm can be very effective when a good initial guess to the target eigenvector is available, with the caveat that it may perform poorly for near-degenerate states, even if they have different quantum numbers. In principle, we can obtain reasonably accurate initial guesses for the target eigenvectors from any of the Lanczos, block Lanczos, LOBPCG or ChebFSI algorithms. However, combining the RMM-DIIS algorithm with the Lanczos algorithm or the ChebFSI algorithm may be less attractive, partly because convergence rates for different eigenpairs in these algorithms are different, at least in the example discussed in Section 3. With the Lanczos algorithm we see from Fig. 2 that the left most eigenvalues typically converge much faster than larger eigenvalues. As a result, the RMM-DIIS method can only be effectively used to refine the eigenvectors associated with larger eigenvalues when they become sufficiently accurate. At that point, the left most eigenvalues are likely to have converged already. Similarly, in Fig. 5 we see that the first and third eigenpair converge slower than the other three eigenpairs. Furthermore, the ChebFSI algorithm tends to be less efficient than either the block Lanczos or the LOBPCG algorithms; and also the Lanczos algorithm tends to be less efficient than block Lanczos or LOBPCG, at least when the multiplication of the sparse Hamiltonian H with several vectors can be implemented efficiently. We therefore do not consider the hybrid ChebFSI and RMM-DIIS nor the hybrid Lanczos and RMM-DIIS method here.

In this section we therefore combine RMM-DIIS with either the block Lanczos or the LOBPCG algorithm to devise a hybrid algorithm that first runs a few block Lanczos or LOBPCG iterations and then use the RMM-DIIS method to refine approximated eigenvectors returned from the block Lanczos or the LOBPCG algorithm simultaneously. Fig. 7 shows that RMM-DIIS indeed converges rapidly when the initial guesses to the eigenvectors associated with the lowest 5 eigenvalues are chosen to be the approximated eigenvectors produced from 10 LOBPCG iterations.

Table 6
SpMV count in hybrid block Lanczos/RMM-DIIS for computing five lowest eigenvalues.

Nucleus	5 block Lanczos	10 block Lanczos	15 block Lanczos	20 block Lanczos	block Lanczos only
^{6}Li	183	163	173	185	208
^{7}Li	205	218	234	221	280
^{11}B	170	178	195	199	240
^{12}C	179	180	179	201	248

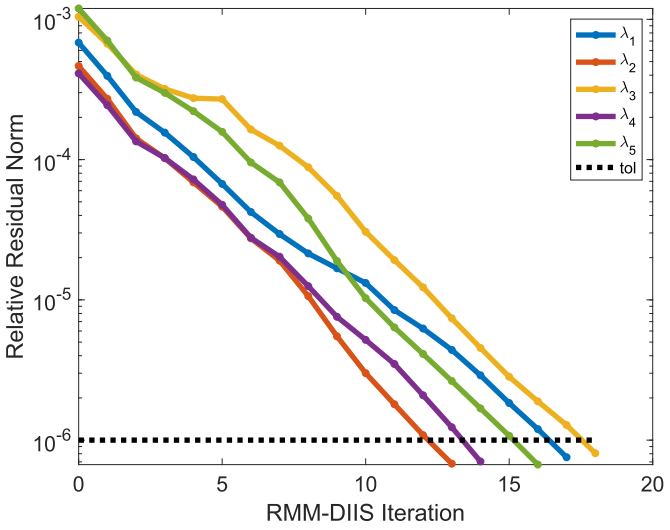

Fig. 7. The convergence of the lowest five eigenvalues of the ^{6}Li Hamiltonian in the RMM-DIIS algorithm after 10 iterations of LOBPCG.

4.1. Switching to RMM-DIIS

A practical question that arises when implementing a hybrid block Lanczos/RMM-DIIS or LOBPCG/RMM-DIIS eigensolver is how to decide when to switch from one algorithm to another. Running more block Lanczos or LOBPCG iterations will produce more accurate approximations to the desired eigenvectors that can then be quickly refined by the RMM-DIIS algorithm. But the cost of running block Lanczos or LOBPCG may dominate the overall cost of the computation. On the other hand, running fewer block Lanczos or LOBPCG iterations may produce a set of approximate eigenvectors that require more RMM-DIIS iterations. There is clearly a trade off, and the optimal point to switch from one to the other depends not only on the specific nucleus, model space, and interaction, but also on details of the numerical implementation and the computational hardware.

Table 6 shows the total number of SpMVs required in a hybrid block Lanczos/RMM-DIIS algorithm in which 5, 10, 15 or 20 block Lanczos iterations are performed first, subsequently followed by the RMM-DIIS procedure. We observe that this hybrid scheme uses fewer total SpMVs for all test problems than that used in a simple block Lanczos or RMM-DIIS run. A similar observation can be made from Table 7 when 10 lowest eigenvalues are computed except for one case, out of 16 cases: for ^{7}Li, the hybrid algorithm that starts with 5 block Lanczos iterations uses more SpMVs than the non-hybrid block Lanczos only algorithm. The optimal number of block Lanczos iterations we should perform (in terms of the total SpMV count) before switching to the RMM-DIIS procedure is problem dependent. Because good approximations to interior eigenvalues only emerge when the Krylov subspace produced by the block Lanczos iteration is sufficiently large, more block Lanczos iterations are required in the hybrid algorithm to yield an optimal SpMV count when 10 eigenvalues and the corresponding eigenvectors are needed, as we can see in Table 7.

Also a hybrid LOBPCG and RMM-DIIS algorithm performs better than the LOBPCG algorithm by itself, provided a sufficiently large number of LOBPCG iterations are performed first to obtain good starting vectors for the subsequent RMM-DIIS procedure, as can be seen in Tables 8 and 9. From Tables 6, 7, 8 and 9, we observe that the total SpMV count is optimal when the number of block Lanczos or LOBPCG iterations is sufficiently large, but not too large. However, the optimal number of block Lanczos/LOBPCG iterations is problem dependent.

One practical way to determine when to switch from block Lanczos or LOBPCG to RMM-DIIS is to examine the change in the approximate eigenvalue. Because RMM-DIIS can be viewed as an eigenvector refinement method, it works well when an approximate eigenvalue becomes sufficiently accurate while the corresponding approximate eigenvector still needs to be corrected. Since we do not know the exact eigenvalues in advance, we use the average changes in the relative difference between approximate eigenvalues obtained in two consecutive iterations (e.g., the $(k-1)$st and the kth iterations) defined as

$$\tau = \frac{1}{n_{\text{ev}}} \sqrt{\sum_{j=1}^{n_{\text{ev}}} \left(\frac{\theta_j^{(k)} - \theta_j^{(k-1)}}{\theta_j^{(k)}} \right)^2}, \tag{29}$$

as a metric to decide when to switch from block Lancos or LOBPCG to RMM-DIIS.

Tables 10 and 11 show the optimal SpMV count of the hybrid block Lanczos/RMM-DIIS and LOBPCG/RMM-DIIS algorithms when computing five or 10 lowest eigenvalues, respectively. Unlike Tables 6, 7, 8 and 9 where we show the results only for $5, 10, 15$ or 20 iterations, these optimal numbers are found by exhaustively trying every possible iteration count of block Lanczos and LOBPCG before switching to RMM-DIIS. Along with the optimal numbers, Tables 10 and 11 also show the SpMV count of the hybrid algorithms when block Lanczos and LOBPCG procedures are stopped as τ goes below 10^{-4} and 10^{-7}.

Table 11 shows that the number of SpMV counts in both the hybrid block Lanczos/RMM-DIIS and LOBPCG/RMM-DIIS algorithms appear to be nearly optimal when $\tau \leq 10^{-7}$. However, when fewer eigenvalues are needed, we can possibly stop the block Lanczos procedure in the hybrid block Lanczos/RMM-DIIS algorithm when τ is below 10^{-4} as shown in Table 10. The $\tau \leq 10^{-7}$ criterion still seems to be a good one for the hybrid LOBPCG/RMM-DIIS algorithm even when fewer eigenpairs are needed.

4.2. Optimal implementation

As we indicated earlier, the SpMV count may not be the best metric to evaluate the performance of a block eigensolver that performs SpMVs for a block of vectors as a SpMM operation. It has also been shown in [42,14] that on many-core processors, the LOBPCG algorithm can outperform the Lanczos algorithm even though its SpMV count is much higher. The reason that a block algorithm can outperform a single vector algorithm such as the Lanczos method on a many-core processor is that SpMM can take advantage of high concurrency and memory locality.

Table 7
SpMV count in hybrid block Lanczos/RMM-DIIS for computing 10 lowest eigenvalues.

Nucleus	5 block Lanczos	10 block Lanczos	15 block Lanczos	20 block Lanczos	block Lanczos only
^{6}Li	422	361	347	367	464
^{7}Li	544	422	418	402	512
^{11}B	419	422	392	382	480
^{12}C	430	415	369	369	480

Table 8
SpMV count in hybrid LOBPCG/RMM-DIIS for computing five lowest eigenvalues.

System	5 LOBPCG	10 LOBPCG	15 LOBPCG	20 LOBPCG	LOBPCG only
^{6}Li	191	158	163	165	184
^{7}Li	211	200	190	194	240
^{11}B	260	162	159	165	192
^{12}C	186	157	153	166	192

Table 9
SpMV count in hybrid LOBPCG/RMM-DIIS for computing 10 lowest eigenvalues.

System	5 LOBPCG	10 LOBPCG	15 LOBPCG	20 LOBPCG	LOBPCG only
^{6}Li	432	344	337	356	464
^{7}Li	529	398	364	365	464
^{11}B	758	397	334	355	423
^{12}C	523	340	317	334	400

Table 10
SpMV count for hybrid algorithms wrt switching threshold on MATLAB to compute five lowest eigenvalues.

Nucleus	Opt Hybrid Block Lan	$\tau \leq 10^{-4}$	$\tau \leq 10^{-7}$	Opt Hybrid LOBPCG	$\tau \leq 10^{-4}$	$\tau \leq 10^{-7}$
^{6}Li	157	194	169	156	187	159
^{7}Li	205	211	224	190	210	190
^{11}B	166	176	195	158	188	159
^{12}C	172	180	179	152	169	153

Table 11
SpMV count for hybrid algorithms wrt switching threshold on MATLAB to compute 10 lowest eigenvalues.

Nucleus	Opt Hybrid Block Lan	$\tau \leq 10^{-4}$	$\tau \leq 10^{-7}$	Opt Hybrid LOBPCG	$\tau \leq 10^{-4}$	$\tau \leq 10^{-7}$
^{6}Li	340	384	346	330	393	335
^{7}Li	401	515	418	363	478	363
^{11}B	379	525	384	332	625	333
^{12}C	362	408	368	312	437	317

The performance of SpMV and SpMM can be measured in terms of number of floating point operations performed per second (GFLOPS). In the following experiments, we measure the performance these computational kernels by running the LOBPCG solver implemented in the MFDn software on a single AMD EPYC 7763 socket on a Perlmutter CPU node maintained at the National Energy Research Scientific Computing (NERSC) Center. The EPYC socket contains 64 cores [43]. We disabled hyperthreading so that 64 OpenMP threads were used within a single MPI rank. Tables 12 and 13 show that the SpMM GFLOPS measured in the LOBPCG algorithm is much higher than the SpMV GFLOPS measured in the Lanczos algorithm. As a result, we can evaluate the performance of a block eigensolver by dividing the actual SpMV count by the ratio between SpMM and SpMV GFLOPS to obtain an "effective" SpMV count. For example, because the SpMM and SpMV GFLOPS ratio is $24.2/4.1 \approx 5.9$ for ^{6}Li, the effective SpMV count for performing 23 iterations of the LOBPCG algorithm with a block size 8 is $23 \times 8/5.9 \approx 31$ which is much lower than the 95 SpMVs performed in the Lanczos method, even though the actual number of SpMVs performed in the LOBPCG iteration is $23 \times 8 = 184 > 95$.

Strictly speaking, RMM-DIIS is a single vector method, i.e., in each RMM-DIIS run, we refine one specific eigenvector associated with a target eigenvalue which has become sufficiently accurate as discussed earlier. However, because the refinement of different eigenvectors can be performed independently from each other, we can perform several RMM-DIIS refinements simultaneously. The simultaneous RMM-DIIS runs can be implemented by batching the SpMVs in each RMM-DIIS iteration together as a single SpMM. This

Table 12
GFLOPs achieved by the SpMV/SpMM kernels within MFDn on one AMD EPYC node with 64 threads in a single MPI rank. The sparse Hamiltonian is applied to 8 vectors in SpMM.

Nucleus	SpMV GFLOPS	SpMM GFLOPS	Ratio
^{6}Li	4.1	24.2	5.9
^{7}Li	6.7	36.0	5.4
^{11}B	6.4	33.5	5.2
^{12}C	6.0	28.0	4.7

Table 13
GFLOPs achieved by the SpMV/SpMM kernels within MFDn on one AMD EPYC node with 64 threads. The sparse Hamiltonian is applied to 16 vectors in SpMM.

Nucleus	SpMV GFLOPS	SpMM GFLOPS	Ratio
^{6}Li	4.1	38.9	9.5
^{7}Li	6.7	56.0	8.4
^{11}B	6.4	50.3	7.9
^{12}C	6.0	47.5	7.9

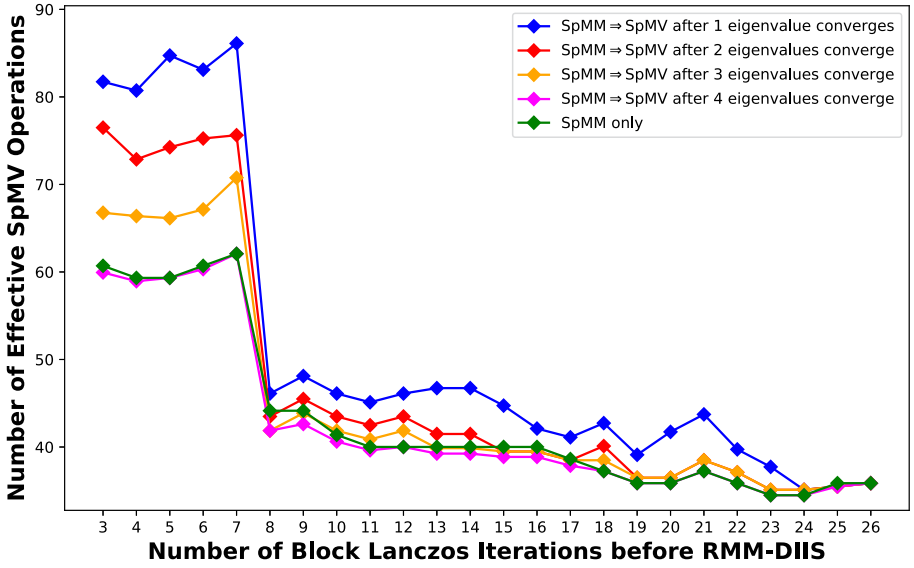

Fig. 8. Total effective SpMV cost of the block Lanczos/RMM-DIIS algorithm to compute the lowest five eigenvalues for ^{6}Li w.r.t. the switching strategy from SpMM to SpMV within RMM-DIIS. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

step constitutes the major cost of the RMM-DIIS method. The least squares problems given in Eqs. (26)–(28) for different eigenvectors can be solved in sequence in each step, since they do not cost much computation.

Because different eigenvectors may converge at different rates as we have already seen in Fig. 6, we need to decide what to do when one or a few eigenvectors have converged. One possibility is to decouple the batched RMM-DIIS method after a certain number of eigenvectors have converged, and switch from using a single SpMM in a coupled RMM-DIIS implementation to using several SpMVs in a decoupled RMM-DIIS implementation. Another possibility is to just keep using the coupled RMM-DIIS with a single SpMM in each step without updating the eigenvectors that have already converged. In this case, the SpMM calculation performs more floating point operations than necessary. However, because an SpMM can be carried out at a much higher GFLOPs than an SpMV, the overall performance of the computation may not be degraded even with the extra computation.

In Fig. 8, we show the effective number of SpMVs performed in several hybrid block Lanczos and RMM-DIIS runs for the ^{6}Li test problem. The horizontal axis represents the number of block Lanczos iterations performed before we switch to RMM-DIIS. The blue dots show the number of effective SpMVs used in the hybrid method when we switch from SpMM to SpMV after one eigenvector has converged. Similarly, the red, orange and magenta dots show the effective SpMV counts when we switch from SpMM to SpMV after two, three and four eigenvectors have converged respectively. The green dots show the effective number of SpMVs when we always use SpMM regardless of how many eigenvectors have converged. As we can see from this figure, the number of effective SpMVs is relatively high when we switch to RMM-DIIS after a few block Lanczos iterations. This is because it will take RMM-DIIS longer to converge if the initial eigenvector approximations produced from the block Lanczos iterations are not sufficiently accurate. Regardless of when we switch from block Lanczos to RMM-DIIS, using SpMM throughout the RMM-DIIS algorithm appears to almost always yield the lowest effective SpMV count. We can also see that the difference in the effective SpMV count is relatively large when we switch from block Lanczos to RMM-DIIS too early. This is understandable because some of the approximate eigenvectors are more accurate than others when we terminate the block Lanczos iteration too early. As a result, the number of RMM-DIIS steps required to reach convergence may vary significantly from one eigenvector to another. The difference in the rate of convergence prevents us from batching several SpMVs into a single SPMM. If we switch after more block Lanczos iterations have been performed, this difference becomes quite small as all approximate eigenvectors are sufficiently accurate and converge more or less at the same rate. The relative difference (τ) between eigenvalue approximations from two consecutive RMM-DIIS iterations falls below 10^{-7} at the 13th block Lanczos iteration. If we switch to RMM-DIIS at that point and use SpMM throughout the RMM-DIIS iteration, the number of effective SpMVs used in the hybrid scheme is about 40, which is slightly higher than the optimal 32 effective SpMVs required if we were to switch to RMM-DIIS after 23 block Lanczos iterations.

An early switch allows us to keep the basis orthogonalization cost of the block Lanczos algorithm as low as possible. As we can see from Fig. 9, the cost of orthogonalization as a percentage of the SpMM cost can become significantly higher as we perform more block Lanczos iterations. In particular, for all test problems, the orthogonalization cost exceeds 50% of the SpMM cost after 20 block Lanczos iterations. We note that the performance shown in Fig. 9 is measured from the wallclock time of an implementation of the block Lanczos algorithm in the MFDn software executed on a single node of the Perlmutter using one MPI rank and 64 threads.

Fig. 10 shows that it is beneficial to use SpMM throughout the hybrid LOBPCG/RMM-DIIS algorithm even after some of the eigenvectors have converged. When a sufficient number of LOBPCG iterations have been performed, the total SPMV count does not vary much. In this case, we should terminate LOBPCG as early as possible to avoid potential numerical instabilities that can be introduced by the numerical rank deficiency of the preconditioned residual vectors [17]. Fig. 11 shows that the estimated condition number of the subspace from which approximated eigenvalues and eigenvectors are extracted in the LOBPCG algorithm increases rapidly as we perform more LOBPCG iterations. Although the optimal SpMV count is attained when we switch to RMM-DIIS after 20 LOBPCG iterations, it is not unreasonable to terminate LOBPCG sooner, for example, after 12 iterations, when the estimated condition number of the LOBPCG subspace is around 10^9. This is also the point at which the average relative change in the approximations to the desired eigenvalues τ just moves below 10^{-7}. Therefore, the previously discussed strategy of using $\tau < 10^{-7}$ to decide when to

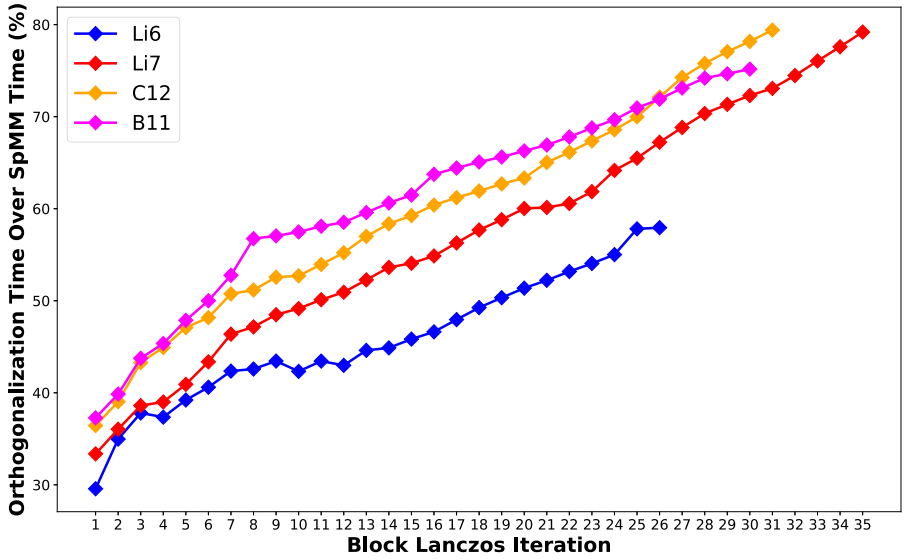

Fig. 9. The percentage of cumulative time spent on orthogonalization compared to SpMM in the block Lanczos algorithm for all test problems to compute the lowest five eigenvalues.

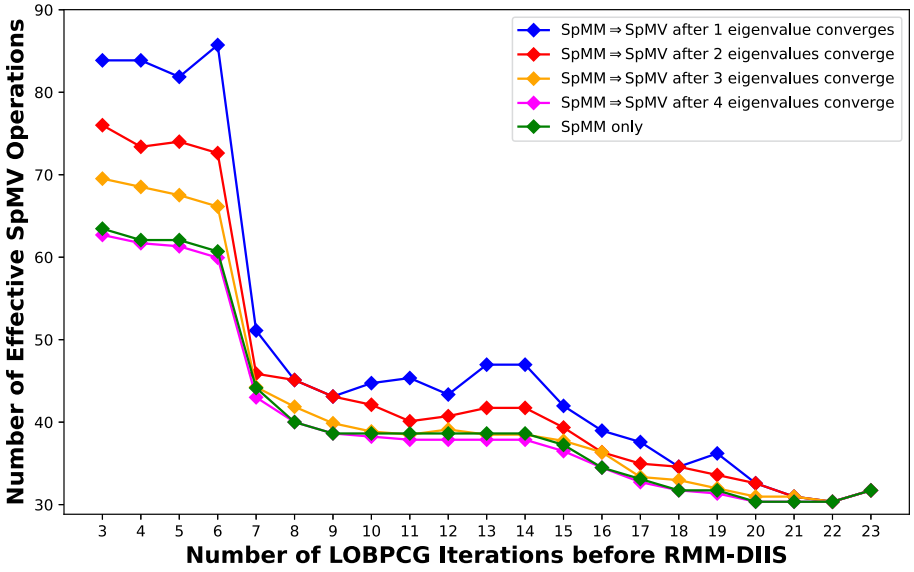

Fig. 10. Total effective SpMV cost of the LOBPCG/RMM-DIIS algorithm to compute the lowest five eigenvalues for ^{6}Li w.r.t. switching strategy from SpMM to SpMV within RMM-DIIS.

switch to RMM-DIIS works well. Even though this strategy would lead to a slight increase in the number of effective SpMV operations compared with the optimal effective SpMV count achieved when we switch to RMM-DIIS after 20 iterations, it makes the hybrid algorithm more robust and stable. We should also note that in a practical calculation the optimal effective SpMV count and when the optimality is achieved is unknown a priori. This optimality is problem dependent, and is also architecture dependent.

5. Conclusion

In this paper, we examine and compare a few iterative methods for solving large-scale eigenvalue problems arising from nuclear structure calculations. We observe that the block Lanczos and LOBPCG methods are generally more efficient than the standard Lanczos method and the stand-alone RMM-DIIS method in terms of the effective number of SpMVs. The Chebyshev filtering based subspace iteration method is not competitive with other methods even though it requires the least amount of memory and has been found to be very efficient for other applications. We show that by combining the block Lanczos or LOBPCG algorithm with the RMM-DIIS algorithm, we obtain a hybrid solver that can outperform existing solvers. The hybrid LOBPCG/RMM-DIIS method is generally more efficient than block Lanczos/RMM-DIIS when a good preconditioner is available. The use of RMM-DIIS in the block Lanczos/RMM-DIIS hybrid algorithm allows us to limit the orthogonalization cost in the block Lanczos iterations. In the LOBPCG/RMM-DIIS hybrid algorithm, the use of RMM-DIIS allows us to avoid the numerical instability that may arise in LOBPCG when the residuals of the approximate eigenpairs become small. We discuss the practical issue of how to decide when to switch from block Lanczos or LOBPCG to RMM-DIIS. A strategy based on monitoring the averaged relative changes in the desired approximate eigenvalues has been found to work well. Although the RMM-DIIS method is a single vector refinement scheme, we show the SpMVs in multiple independent RMM-DIIS iterations targeting different eigenpairs can be batched together and implemented as a single SpMM. Such

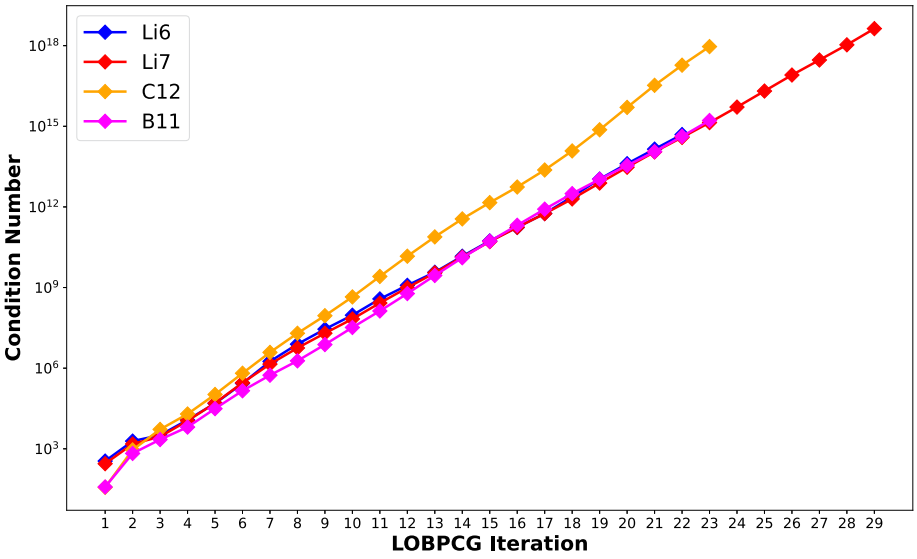

Fig. 11. The condition number of the LOBPCG subspace from which approximate eigenpairs are extracted to compute the lowest five eigenvalues of the ^{6}Li Hamiltonian.

a batching scheme significantly improves the performance of the hybrid solver and is found to be useful even after some of the approximate eigenpairs have converged.

The actual performance of the solvers discussed in this work, in terms of wall clock time, depends not only on the algorithm, but also on the detailed implementation and underlying hardware. For extremely large (matrix sizes n of well over 10 billion), but extremely sparse matrices, the standard Lanczos algorithm may actually be the most efficient in terms of wall-clock time for obtaining approximate eigenpairs, due to lower communication overhead when utilizing thousands of compute nodes. Also for applications that need thousands of eigenvalues and corresponding eigenvectors, the memory requirement for Lanczos, block Lanczos, LOBPCG and Chebyshev filtering may be too high for problems of very large dimensions. In such a case, alternative algorithms such as the spectral slicing algorithm presented in [44,45] may be used to compute a subset of desired eigenvalues and eigenvectors at a time as we sweep through the low end of the spectrum. The approximated eigenvectors can be refined by the RMM-DIIS algorithm discussed earlier.

In typical large-scale nuclear structure calculations, we are primarily interested in the lowest eigenvalues and corresponding quantum numbers. In such cases, one can use approximate convergence of the eigenvalues as an initial stopping criterion, and subsequently refine a subset of those eigenvectors (not necessarily those corresponding to the lowest eigenvalues) with the RMM-DIIS algorithm, before using those refined eigenvectors to evaluate additional observables of interest such as charge radii, magnetic and quadrupole moments, and electroweak transitions.

Declaration of competing interest

We declare that we have no financial and personal relationships with other people or organizations that can inappropriately influence our work, there is no professional or other personal interest of any nature or kind in any product, service or company that could be construed as influencing the position presented in, or the review of, the manuscript titled “Hybrid Eigensolvers for Nuclear Configuration Interaction Calculations”.

Data availability

Data will be made available on request.

Acknowledgements

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Nuclear Physics under Grants DE-SC0023495 and DE-SC0023175, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program via the FASTMath Institute under Contract No. DE-AC02-05CH11231, and National Science Foundation’s Office of Advanced Cyberinfrastructure under Grant 1845208. This work used resources of the National Energy Research Scientific Computing Center (NERSC) using NERSC Award ASCR-ERCAP m1027 for 2023, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] B.R. Barrett, P. Navratil, J.P. Vary, Prog. Part. Nucl. Phys. 69 (2013) 131–181, https://doi.org/10.1016/j.ppnp.2012.10.003.
[2] C. Lanczos, J. Res. Natl. Bur. Stand. 45 (4) (1950) 255–282.
[3] E. Caurier, F. Nowacki, Acta Phys. Pol. B 30 (1999) 705.
[4] B. Brown, W. Rae, Nucl. Data Sheets 120 (2014) 115–118, https://doi.org/10.1016/j.nds.2014.07.022.
[5] C.W. Johnson, W.E. Ormand, P.G. Krastev, Comput. Phys. Commun. 184 (2013) 2761–2774, https://doi.org/10.1016/j.cpc.2013.07.022, arXiv:1303.0905.
[6] C.W. Johnson, W.E. Ormand, K.S. McElvain, H. Shan, BIGSTICK: a flexible configuration-interaction shell-model code, arXiv:1801.08432, 2018, https://github.com/cwjsdsu/BigstickPublick/.
[7] N. Shimizu, Nuclear shell-model code for massive parallel computation, “KSHELL”, arXiv:1310.5431, 2013.
[8] N. Shimizu, T. Mizusaki, Y. Utsuno, Y. Tsunoda, Comput. Phys. Commun. 244 (2019) 372–384, https://doi.org/10.1016/j.cpc.2019.06.011, arXiv:1902.02064.
[9] P. Sternberg, E.G. Ng, C. Yang, P. Maris, J.P. Vary, M. Sosonkina, H.V. Le, in: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, SC08, IEEE, 2008, pp. 1–12.
[10] P. Maris, M. Sosonkina, J.P. Vary, E. Ng, C. Yang, Proc. Comput. Sci. 1 (1) (2010) 97–106, https://doi.org/10.1016/j.procs.2010.04.012, iCCS 2010.
[11] H.M. Aktulga, C. Yang, E.G. Ng, P. Maris, J.P. Vary, Concurr. Comput. 26 (16) (2014) 2631–2651, https://doi.org/10.1002/cpe.3129.
[12] B. Cook, P.J. Fasano, P. Maris, C. Yang, D. Oryspayev, in: S. Bhalachandra, C. Daley, V. Melesse Vergara (Eds.), Accelerator Programming Using Directives, WACCPD 2021, Springer International Publishing, 2022, pp. 112–132, arXiv:2110.10765.
[13] P. Maris, C. Yang, D. Oryspayev, B. Cook, J. Comput. Sci. 59 (2022) 101554, https://doi.org/10.1016/j.jocs.2021.101554, arXiv:2109.00485.
[14] M. Shao, H.M. Aktulga, C. Yang, E.G. Ng, P. Maris, J.P. Vary, Comput. Phys. Commun. 222 (2018) 1–13, https://doi.org/10.1016/j.cpc.2017.09.004.
[15] A.V. Knyazev, SIAM J. Sci. Comput. 23 (2) (2001) 517–541, https://doi.org/10.1137/S1064827500366124.

[16] U. Hetmaniuk, R. Lehoucq, J. Comput. Phys. 218 (2006) 324–332.
[17] J.A. Duersch, M. Shao, C. Yang, M. Gu, SIAM J. Sci. Comput. 40 (5) (2018) C655–C676.
[18] G.H. Golub, R. Underwood, in: J.R. Rice (Ed.), Mathematical Software III, 1977, pp. 361–377.
[19] Y. Saad, Math. Comput. 42 (1984) 567–588, https://doi.org/10.1090/S0025-5718-1984-0736453-8, https://www.ams.org/journals/mcom/1984-42-166/S0025-5718-1984-0736453-8/.
[20] Y. Zhou, J.R. Chelikowsky, Y. Saad, J. Comput. Phys. 274 (2014) 770–782, https://doi.org/10.1016/j.jcp.2014.06.056, http://www.sciencedirect.com/science/article/pii/S0021999114004744.
[21] D.M. Wood, A. Zunger, J. Phys. A, Math. Gen. 18 (9) (1985) 1343, https://doi.org/10.1088/0305-4470/18/9/018.
[22] Z. Jia, G.W. Stewart, Math. Comput. 70 (2001) 637–647.
[23] P. Pulay, Chem. Phys. Lett. 73 (1980) 393–398.
[24] Y. Saad, SIAM J. Numer. Anal. 17 (5) (1980) 687–706, http://www.jstor.org/stable/2156670.
[25] R.C. Li, L. Zhang, Numer. Math. 131 (2015) 83–113, https://doi.org/10.1007/s00211-014-0681-6.
[26] R.G. Grimes, J.G. Lewis, H.D. Simon, SIAM J. Matrix Anal. Appl. 15 (1) (1994) 228–272, https://doi.org/10.1137/S0895479888151111.
[27] R.B. Lehoucq, D.C. Sorensen, C. Yang, ARPACK Users' Guide, Society for Industrial and Applied Mathematics, 1998, https://epubs.siam.org/doi/abs/10.1137/1.9780898719628.
[28] G.W. Stewart, SIAM J. Matrix Anal. Appl. 23 (3) (2002) 601–614, https://doi.org/10.1137/S0895479800371529.
[29] Y. Zhou, Y. Saad, Numer. Algorithms 47 (2008) 341–359, https://doi.org/10.1007/s11075-008-9192-9.
[30] A.V. Knyazev, K. Neymeyr, SIAM J. Matrix Anal. Appl. 31 (2) (2009) 621–628, https://doi.org/10.1137/080727567.
[31] K. Neymeyr, SIAM J. Numer. Anal. 50 (6) (2012) 3188–3207, https://doi.org/10.1137/11084488X.
[32] M. Argentati, A. Knyazev, K. Neymeyr, E. Ovtchinnikov, M. Zhou, Found. Comput. Math. 17 (2017) 713–727.
[33] T. Manteuffel, Math. Comput. 34 (1980) 473–497.
[34] B.N. Parlett, The Symmetric Eigenvalue Problem, Prentice Hall, Englewood Cliffs, 1980.
[35] G.H. Golub, C.F.V. Loan, Matrix Computations, 3rd ed., Johns Hopkins University Press, 1996.
[36] J.L.M.V. Dorsselaer, M.E. Hoschstenbach, H.A.V. der Vorst, SIAM J. Matrix Anal. Appl. 22 (3) (2000) 837–852.
[37] Y. Zhou, Y. Saad, M.L. Tiago, J.R. Chelikowsky, Phys. Rev. E 74 (2006) 066704.
[38] R. Li, Y. Zhou, Linear Algebra Appl. 435 (2011) 480–493.
[39] A.M. Shirokov, I.J. Shin, Y. Kim, M. Sosonkina, P. Maris, J.P. Vary, Phys. Lett. B 761 (2016) 87–91, https://doi.org/10.1016/j.physletb.2016.08.006, arXiv:1605.00413.
[40] C.C. Paige, M.A. Saunders, SIAM J. Numer. Anal. 12 (4) (1975) 617–629, https://doi.org/10.1137/0712047.
[41] Y. Saad, Numerical Methods for Large Eigenvalue Problems, revised edition, Classics in Applied Mathematics, vol. 66, SIAM, Philadelphia, 2011.
[42] H.M. Aktulga, A. Buluç, S. Williams, C. Yang, in: 2014 IEEE 28th International Parallel and Distributed Processing Symposium, IPDPS 2014, IEEE, 2014, pp. 1213–1222.
[43] NERSC, NERSC Cori Systems, https://docs.nersc.gov/systems/cori. (Accessed June 2021).
[44] R. Li, Y. Xi, E. Vecharynski, C. Yang, Y. Saad, SIAM J. Sci. Comput. 38 (4) (2016) A2512–A2534, https://doi.org/10.1137/15M1054493.
[45] R. Li, Y. Xi, L. Erlandson, Y. Saad, SIAM J. Sci. Comput. 41 (4) (2019) C393–C415, https://doi.org/10.1137/18M1170935.