Computer Physics Communications 224 (2018) 81–89

Contents lists available at ScienceDirect

Computer Physics Communications

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

Paradeisos: A perfect hashing algorithm for many-body eigenvalue problems

C.J. Jia [a,*,1], Y. Wang [a,b,*,1], C.B. Mendl [a], B. Moritz [a,c], T.P. Devereaux [a,d]

[a] Stanford Institute for Materials and Energy Sciences, SLAC National Accelerator Laboratory, 2575 Sand Hill Road, Menlo Park, CA 94025, USA
[b] Departments of Applied Physics, Stanford University, CA 94305, USA
[c] Department of Physics and Astrophysics, University of North Dakota, Grand Forks, ND 58202, USA
[d] Geballe Laboratory for Advanced Materials, Stanford University, CA 94305, USA

ARTICLE INFO

Article history:
Received 13 July 2017
Received in revised form 25 September 2017
Accepted 21 November 2017
Available online 2 December 2017

Keywords:
Many-body physics
Strongly correlated system
Perfect hashing
Exact diagonalization
Hubbard model
Checkerboard decomposition

ABSTRACT

We describe an essentially perfect hashing algorithm for calculating the position of an element in an ordered list, appropriate for the construction and manipulation of many-body Hamiltonian, sparse matrices. Each element of the list corresponds to an integer value whose binary representation reflects the occupation of single-particle basis states for each element in the many-body Hilbert space. The algorithm replaces conventional methods, such as binary search, for locating the elements of the ordered list, eliminating the need to store the integer representation for each element, without increasing the computational complexity. Combined with the "checkerboard" decomposition of the Hamiltonian matrix for distribution over parallel computing environments, this leads to a substantial savings in aggregate memory. While the algorithm can be applied broadly to many-body, correlated problems, we demonstrate its utility in reducing total memory consumption for a series of fermionic single-band Hubbard model calculations on small clusters with progressively larger Hilbert space dimension.

1. Introduction

A number of wavefunction- and Green's function-based numerical techniques have been developed to address the problem of strongly interacting electrons in lattice models for condensed matter systems. [1,2] Among the most widely and straightforwardly applied methods is small cluster exact diagonalization, which is particularly well suited to problems with strong interactions where the important physics remains local or relatively short-ranged. One explicitly constructs a many-body Hamiltonian from the full Hilbert space that consists of all allowed multi-particle configurations of single-particle states for the full lattice problem [3–12]. The usefulness of this technique is limited by the exponential growth of the Hilbert space dimension D with increasing size of the clusters.

Full diagonalization of the Hamiltonian – evaluating all eigenvalues and eigenvectors – remains impractical for all but the smallest problems with a typical computational complexity of $O(D^3)$. However, accurate information about the ground (lowest energy) state and several low lying excited states can be sufficient for zero, or low temperature properties of the model. To that end, iterative Krylov subspace methods, such as Lanczos or Arnoldi [13], can be employed to reduce the computational burden. Dynamical properties can be evaluated from the ground (and excited) state eigenfunctions using secondary numerical methods [14,15], such as the continued fraction expansion [3] or bi-conjugate gradient stabilized techniques [16].

A reduction of the computational complexity for these methods also comes from the fact that the Hamiltonian matrix is typically sparse, with a polynomial number of non-zero elements (usually only several hundred non-zero elements) per row or column, typically orders of magnitude smaller than the Hilbert space dimension D. However, exponential growth of the Hilbert space means that the Hamiltonian matrix or even the wavefunctions might be too large to store in the memory available on a single compute node. This is certainly the case for the largest and most challenging problems for exact diagonalization. The nature of modern parallel computing environments with a large number of lightweight cores per node and limited memory per core (or per node) makes the issue of efficient memory utilization in exact diagonalization one of the most significant bottlenecks to performance and scalability.

One solution to this problem has been a "checkerboard" decomposition of the Hamiltonian matrix as illustrated in Fig. 1. This decomposition distributes a different partition of the matrix and wavefunctions (blocks with the same color) to different processors, or compute nodes, to balance the memory distribution. When combined with graph partitioning and matrix reordering

* Corresponding authors.
E-mail addresses: chunjing@stanford.edu (C.J. Jia), yaowang@g.harvard.edu (Y. Wang).
1 Contributed equally to this work.

https://doi.org/10.1016/j.cpc.2017.11.011
0010-4655/

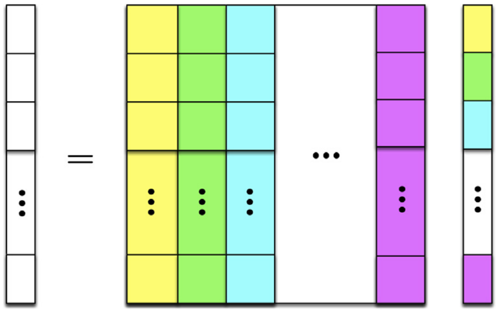

Fig. 1. Checkerboard decomposition of the Hamiltonian matrix. The colors represent a range of indices in the many-body basis, for both the sparse matrix and wavefunctions, distributed between different compute nodes.

algorithms, such as those in the PARMETIS library [17], this scheme maximizes data locality, reducing parallel communications. The remaining performance and memory bottleneck comes from the construction of the matrix from the many-body basis (Hilbert space) and the one-to-one mapping between each state and the appropriate index. A traditional binary search algorithm may be employed when the states are properly ordered; however, this method usually requires a hash table, which encodes the index-state map, resident in memory for each processor, or compute node, to minimize communications overhead.

An ideal hashing scheme would allow for the evaluation of the index-state map, in both the forward and reverse directions, without any memory overhead. Previous studies have found partial solutions to this issue. A study by Lin [9] demonstrated reduced memory usage with a higher-dimensional search algorithm, but requiring a longer computational time. Another study by Liang [18] reported a perfect hashing algorithm for many-body systems, with reduced memory cost and no increase in time complexity; however, the algorithm was only demonstrated for a Hilbert space with fixed particle number, without considering changes to the algorithm due to symmetry reduction of the effective Hilbert space. Such a perfect hashing algorithm which saves memory without sacrificing computing time, and can be employed in symmetry reduced subspaces, would represent a considerable advance for problems of this type.

In this paper, we devise such a perfect hashing algorithm, *Paradeisos*, to replace conventional search with a direct mapping. This eliminates the need to store a table for the mapping between the vector–matrix index and the many-body basis. The method becomes particularly advantageous for large problem sizes where symmetry reduction is a must and in parallel computing environments where the "checkerboard" decomposition must be used for distributed matrix–vector storage. We describe *Paradeisos* in the framework of a single-band, or single-orbital, model for two species of interacting fermions ("spin-up" $\uparrow$ and "spin-down" $\downarrow$) on a lattice. Extension of this algorithm to multi-orbital models, or models for spin and bosonic systems, can be accomplished with straightforward modifications. The paper is organized as follows: first, we introduce the algorithm and the forward and backward mapping in the general case; second, we discuss the implementation of the algorithm in symmetry projected subspaces of the full Hilbert space; then, we explore the performance of the algorithm and its scaling with the number of parallel processes; and finally, we summarize and discuss the generalization to spin and bosonic systems.

2. Hilbert space and model Hamiltonian

2.1. Many-body basis

Start with the single-site fermionic problem. There are four configurations due to the Pauli exclusion principle: $|0\rangle$, $|\uparrow\rangle$, $|\downarrow\rangle$, $|\uparrow\downarrow\rangle$ (or in occupation and spin direct product basis $|0\rangle_\uparrow \otimes |0\rangle_\downarrow$, $|1\rangle_\uparrow \otimes |0\rangle_\downarrow$, $|0\rangle_\uparrow \otimes |1\rangle_\downarrow$, $|1\rangle_\uparrow \otimes |1\rangle_\downarrow$). For a multi-site problem, one typically works in the canonical ensemble with fixed total electron number. In general, for an N-site cluster with $n_\uparrow$ spin-up and $n_\downarrow$ spin-down electrons, the Hilbert space dimension $D = C_N^{n_\uparrow} \cdot C_N^{n_\downarrow}$, where C_m^n is the binomial coefficient.

Any basis element of the Hilbert space, represented in terms of the occupation of single-particle states, can be constructed from the vacuum $|0\rangle$ by repeated application of creation operators $c_{i\sigma}^\dagger$, where $\sigma \in \{\uparrow, \downarrow\}$ and $i \in \{0, 1, \dots, N-1\}$ indexes lattice positions, such that

$$|\varphi\rangle = \underbrace{\dots c_{i\uparrow}^\dagger \dots}_{n_\uparrow} \underbrace{\dots c_{j\downarrow}^\dagger \dots}_{n_\downarrow} |0\rangle.$$

Note the "normal ordering" convention with spin-down operators on the right, and we take the increasing site index right-to-left. The convention ensures proper antisymmetrization for fermion exchange. As an illustration to which we will return throughout the discussion of the algorithm, consider a simple 4-site lattice model with two spin-up ($n_\uparrow = 2$) and three spin-down ($n_\downarrow = 3$) electrons. In this case, $D = 24$ and one particular element of the Hilbert space basis

$$c_{2\uparrow}^\dagger c_{0\uparrow}^\dagger c_{3\downarrow}^\dagger c_{1\downarrow}^\dagger c_{0\downarrow}^\dagger |0\rangle = (0\uparrow 0 \uparrow) \otimes (\downarrow 0 \downarrow\downarrow).$$

For practical computations, all basis elements can be enumerated by noting the correspondence of the occupation and spin direct product representation with bit sequences for unique integer values. Turn again to the example of one particular basis element for the 4-site lattice model.

$$(0\uparrow 0 \uparrow) \otimes (\downarrow 0 \downarrow\downarrow) \rightarrow 0101_2 \otimes 1011_2,$$

where each occurrence of $\uparrow$ or $\downarrow$ has been replace by a 1 (occupied) with 0 otherwise. The subscript 2 on each sequence denotes binary notation for clarity. A lexicographical binary representation can be constructed separately for every spin-up and spin-down element comprising the Hilbert space basis. For the 4-site example, results are presented in the following table.

index	$\uparrow$ element	index	$\downarrow$ element
0	$0011_2 = 3$	0	$0111_2 = 7$
1	$0101_2 = 5$	1	$1011_2 = 11$
2	$0110_2 = 6$	2	$1101_2 = 13$
3	$1001_2 = 9$	3	$1110_2 = 14$
4	$1010_2 = 10$		
5	$1100_2 = 12$		

In general, the lexicographical next state, advancing the index $p \rightarrow p+1$, can be determined using just a few bit operations [19–21]. A tensor product of the spin-up and spin-down contributions defines the complete basis. This tensor product structure typically breaks down within restricted symmetry subspaces, which we will address in a subsequent section.

2.2. Many-body Hamiltonian

Although the algorithm does not depend on the details of the many-body Hamiltonian, for simplicity, we consider the single-band Hubbard model. Note that the symmetry of the Hamiltonian affects the selection of the subspace. The single-band Hubbard Hamiltonian H includes both kinetic and on-site Coulomb interaction terms, written conventionally as

$$\begin{aligned} H &= H_{kin} + H_{int} \\ &= -\sum_{ij,\sigma} \left(t_{ij} c_{i\sigma}^\dagger c_{j\sigma} + h.c.\right) + U \sum_i n_{i\uparrow} n_{i\downarrow}, \end{aligned} \tag{1}$$

where $n_{i\sigma} = c_{i\sigma}^\dagger c_{i\sigma}$ is the standard number operator. The kinetic terms include "hopping" of electrons on the lattice with an energy t_{ij}, typically restricted to nearest or next-nearest neighboring sites, and U parameterizes the on-site interaction strength.

H can be viewed as a one-to-many map of the Hilbert space into itself. Consider first the interaction terms. In a representation based on real-space occupation on the lattice, the interaction depends on the "double occupancy" — how many sites have occupation by both ↑ and ↓ fermions. Comprised solely of number operators, this term represents a map of each many-body Hilbert space basis element to itself with a prefactor. In bit operations, the "double occupancy" (and by extension the Hamiltonian matrix elements) can be computed from the bitwise `AND` between the ↑ and ↓ parts of the direct product.

The kinetic terms represent a more complicated map. The movement of fermions on the lattice, or "hopping", mixes basis elements through a change in the fermion occupation. One must determine not only the weights for the mapping (Hamiltonian matrix elements), but also identify the index (or indices) within the basis to which H maps each element. Return to our example of the 4-site lattice; and consider a kinetic term with only nearest-neighbor hopping t and sites linked together cyclically, as in a one-dimensional loop ($0 \leftrightarrow 1 \leftrightarrow 2 \leftrightarrow 3 \leftrightarrow 0$). For simplicity, we address only the spin-up kinetic term acting on our test element $0101_2 \otimes 1011_2$, where the first bit sequence of the product corresponds to spin-up. In this case,

$$H^{\uparrow}_{kin}(0101_2 \otimes 1011_2) = \\ -t\big(1001_2 + 0011_2 + 0110_2 - 1100_2\big) \otimes 1011_2,$$

where the minus sign in front of 1100_2 results from the normal ordering convention and the antisymmetry of fermion exchange. One can immediately read-off the index for each state from the table created for this example; however, ideally one would like a more automated and less brute force method for determining the new indices under this mapping for a state-of-art size of quantum many-body problem.

2.3. Index determination

One may employ conventional binary search algorithm [22], to determine the indices to which a function such as H will map each basis element. The binary search algorithm has time complexity $\mathcal{O}(\ln D)$. Since $D = C_N^{n_\uparrow} \cdot C_N^{n_\downarrow} < 4^N$, an upper bound for the time complexity of binary search is $\mathcal{O}(\ln 4^N)$ or $\mathcal{O}(N)$. However, one must typically store the enumerated bit sequences for all basis elements in memory to reduce the communications overhead.

In the following sections we describe *Paradeisos*, a direct "forward mapping" from a bit sequence to its basis index, mitigating the need for search, and by extension storage. Crucially, this algorithm has the same time complexity as traditionally binary search methods. When combined with advanced matrix decomposition techniques for parallel computing environments, *Paradeisos* has the potential to significantly reduce aggregate memory consumption and improve performance (see the section on "Numerical Benchmarks").

3. Mapping functions

The key feature of *Paradeisos* is a mapping from a bit sequence to an index. As opposed to binary search, which identifies an index from an ordered list through an iterative series of bisections, *Paradeisos* determines the (absolute) index of an arbitrary bit sequence of the list by mathematically evaluating the total number of different configurations between it and a known pattern.

First, we define the distance between two elements φ_1 and φ_2 by the difference of their indices $\mathrm{dist}\{\varphi_1, \varphi_2\} = \mathrm{idx}(\varphi_2) - \mathrm{idx}(\varphi_1)$. However, directly evaluating this distance is non-trivial. It can be straightforward in the special case of *simple neighbors*, where two bit sequences differ by exchange of a single 1-bit across consecutive 0-bits. As an example consider two elements, each with 4 non-zero *least significant bits* (LSBs) up to the position of this exchange. The first

$$|\varphi_1\rangle := \ldots \overset{j_2}{0}\,0\,\overset{j_1}{0}\,\underbrace{\mathbf{1}\,0\,0\,1\,0\,1\,\overset{i=0}{1}}_{n=4}{}_2$$

has an index $\mathrm{idx}(\varphi_1) = p$, and the second

$$|\varphi_2\rangle := \ldots \underbrace{\overset{j_2}{\mathbf{1}}\,0\,\overset{j_1}{0}\,0\,0\,0\,1\,0\,1\,\overset{i=0}{1}}_{n=4}{}_2$$

has an index $\mathrm{idx}(\varphi_2) = q$. The distance between these two bit sequences depends on the number of configurations of the n bits between them, which is given simply by

$$\mathrm{dist}\{\varphi_1, \varphi_2\} = q - p = C^n_{j_2} - C^n_{j_1}, \tag{2}$$

with a convention that the bit position starts from $i = 0$ and that $C^n_m = 0$ for $m < n$. Note that Eq. (2) is independent of the particular bit configuration to the right of j_1, nor does it depend upon the configuration to the left of j_2 if the total number of 1-bits were greater than n.

One can construct the mapping between a bit sequence and its corresponding index, via successive exchanges from a given sequence with a known index, by taking advantage of the distance between simple neighbors. By construction, index 0 for the first element in a list of N-bit numbers with n non-zero bits ($n < N$) belongs to the bit sequence with the n LSBs set to 1:

$$|\varphi_0\rangle := \underbrace{\overset{N-1}{0}\,\ldots\,\overset{n}{0}\,\underbrace{\overset{n-1}{1}\,\ldots\,\overset{i=0}{1}}_{n}}_{N}{}_2,$$

$$\mathrm{idx}(\varphi_0) = 0$$

For any other N-bit sequence $|\varphi\rangle$ in the ordered list, its n 1-bits will occupy positions $j_{n-1} > j_{n-2} > \cdots > j_1 > j_0$. Starting from $|\varphi_0\rangle$, one can construct a sequence of n simple neighbors $\{|\varphi_0^{(n)}\rangle, \ldots, |\varphi_0^{(1)}\rangle = |\varphi\rangle\}$ by sequential exchange of the mth LSB, $\{m = n, \ldots, 1\}$. Thus,

$$\begin{aligned}
|\varphi_0\rangle &:= \overset{N-1}{0}\;\ldots\;\overset{n}{0}\;\overset{n-1}{1}\;\ldots\;\overset{i=0}{1}{}_2 \\
|\varphi_0^{(n)}\rangle &:= \ldots\;\overset{j_{n-1}}{1}\;\ldots\;\overset{n-1}{0}\;\overset{n-2}{1}\;\ldots\;\overset{i=0}{1}{}_2 \\
|\varphi_0^{(n-1)}\rangle &:= \ldots\;\overset{j_{n-1}}{1}\;\ldots\;\overset{j_{n-2}}{1}\;\ldots\;\overset{n-2}{0}\;\overset{n-3}{1}\;\ldots\;\overset{i=0}{1}{}_2 \\
&\;\;\vdots \\
|\varphi_0^{(2)}\rangle &:= \ldots\;\overset{j_{n-1}}{1}\;\ldots\;\overset{j_{n-2}}{1}\;\ldots\overset{j_1}{1}\ldots\overset{1}{0}\;\overset{i=0}{1}{}_2 \\
|\varphi\rangle = |\varphi_0^{(1)}\rangle &:= \ldots\;\overset{j_{n-1}}{1}\;\ldots\;\overset{j_{n-2}}{1}\;\ldots\overset{j_1}{1}\ldots\overset{j_0}{1}\ldots{}_2,
\end{aligned}$$

and one can now read-off the index simply as the accumulated distance between simple neighbors

$$\mathrm{idx}(\varphi) = C^1_{j_0} + C^2_{j_1} + \cdots + C^{n-1}_{j_{n-2}} + C^n_{j_{n-1}}. \tag{3}$$

***Paradeisos* forward map**
Goal: For n electrons and N sites, map $\varphi \mapsto \mathrm{idx}(\varphi)$

function *forward_map* (φ)
  declare idx = 0, m = 0.
  for $i = 0$ to $N - 1$:
    if ith bit in φ is TRUE:
      $m \leftarrow m + 1$
      $\mathrm{idx} \leftarrow \mathrm{idx} + C^m_i$
    end
  end
  return idx

For efficiency, the binomial coefficients C_i^m for $i, m \in \{0, 1, \ldots, N-2, N-1\}$ can be precomputed recursively $C_i^m = C_{i-1}^m + C_{i-1}^{m-1}$, and stored in a lookup-table. The time complexity for *Paradeisos* is O(N), comparable to traditional binary search.

As an example, return to the problem of 2 spin-up and 3 spin-down electrons on a 4 site lattice and, for simplicity, consider only element 1100_2 in the spin-up part of the Hilbert space basis. After exchange of the highlighted (boldface/underline) bits,

$$|\varphi_0\rangle := 00\underline{\mathbf{1}}1_2, \qquad \mathrm{idx}(\varphi_0) = 0$$
$$\downarrow$$
$$|\varphi_0^{(2)}\rangle := \underline{\mathbf{1}}001_2, \qquad \mathrm{idx}(\varphi_0^{(2)}) = C_3^2 = 3,$$

and

$$|\varphi_0^{(2)}\rangle := 100\underline{\mathbf{1}}_2, \qquad \mathrm{idx}(\varphi_0^{(2)}) = C_3^2 = 3$$
$$\downarrow$$
$$|\varphi\rangle = |\varphi_0^{(1)}\rangle := 1\underline{\mathbf{1}}00_2, \qquad \mathrm{idx}(\varphi) = C_3^2 + C_2^1 = 5.$$

One can refer back to the table to verify the result. The index in the full Hilbert space, composed from tensor products between basis elements in the two spin sectors, can be computed from the known dimension of the spin-resolved Hilbert spaces. Given our convention with spin-down configurations occupying the first N LSBs, $\mathrm{idx}(\varphi) = C_N^{n_\downarrow} \cdot \mathrm{idx}_\uparrow(\varphi_\uparrow) + \mathrm{idx}_\downarrow(\varphi_\downarrow)$, where $\uparrow$ /$\downarrow$ subscripts denote the appropriate spin-restricted subspace.

To be complete, *Paradeisos* requires a "backward mapping" for determining a bit sequence from a known index. This map follows from the accumulated distance between simple neighbors to determine the position of non-zero bits. Pseudocode summarizes the procedure.

***Paradeisos* backward map**
Goal: For n electrons and N sites, map $\mathrm{idx}(\varphi) \mapsto \varphi$

```
function backward_map(idx; n, N)
    declare φ = 00...00₂, m = n, p = idx
    for i = N − 1 to 0:
        if p ≥ C_i^m:
            set φ's ith bit to TRUE
            p ← p − C_i^m
            m ← m − 1
        end
    end
    return φ
```

One can verify this procedure in the previous example by inspection.

4. Mapping within symmetry subspaces

One typically uses symmetries to reduce the dimension of the effective Hilbert space. These may include SU(2) spin symmetry [1], point group symmetries such as translational, rotational [23] and inversion symmetry, or time reversal symmetry that partition the original Hilbert space and block diagonalize the Hamiltonian. In the following sections, we illustrate the implementation of *Paradeisos* within both translation and inversion symmetry restricted subspaces.

4.1. Translational symmetry

For regular lattice models defined on small clusters with periodic boundary conditions, translation symmetry can lead to a reduction in the Hilbert space dimension by a factor $\sim N$, the total number of lattice sites or unit cells. Translational invariance typically prompts one to choose a momentum space representation via a discrete Fourier transform of the operators $c_{\mathbf{k}}^\dagger = \frac{1}{\sqrt{N}}\sum_j e^{i\mathbf{k}\cdot\mathbf{r}_j c_j^\dagger}$ (similarly for the hermitian conjugate, annihilation operator $c_{\mathbf{k}}$). Transforming the single-band Hubbard model of Eq. (1) to this momentum space basis leads to

$$H = \sum_{\mathbf{k},\sigma} \varepsilon_{\mathbf{k}} c_{\mathbf{k}\sigma}^\dagger c_{\mathbf{k}\sigma} + \frac{U}{N}\sum_{\mathbf{k},\mathbf{k}',\mathbf{q}} c_{\mathbf{k}+\mathbf{q}\uparrow}^\dagger c_{\mathbf{k}'-\mathbf{q}\downarrow}^\dagger c_{\mathbf{k}'\downarrow} c_{\mathbf{k}\uparrow} \tag{4}$$

where $\varepsilon_{\mathbf{k}}$ is the kinetic energy in momentum space, typically referred to as the bare band structure. For the single-band Hubbard model on a square lattice with only nearest, t, and next-nearest neighbor, t', hopping terms, $\varepsilon_{\mathbf{k}} = -2t(\cos k_x + \cos k_y) - 4t' \cos k_x \cos k_y$. The Pauli exclusion principle also applies in momentum space; so given a normal ordering of the discrete single-particle momenta $\{\mathbf{k}_0, \ldots, \mathbf{k}_{N-1}\}$, bit sequences relate to fermion occupation of single-particle momentum states. Note that the full Hilbert space dimension D in the momentum space representation remains the same as that in the real space representation; however, inspection of the two terms in the Hamiltonian reveals an effective dimensional reduction. The kinetic term is now diagonal in momentum space, whereas the interaction term scatters fermions of momenta $\mathbf{k}$ and $\mathbf{k}'$ to new momenta $\mathbf{k}+\mathbf{q}$ and $\mathbf{k}'-\mathbf{q}$, while leaving the total momentum of the many-body state $\mathbf{K}$ unchanged. Thus, $\mathbf{K}$ can be used to partition the basis elements of the Hilbert space with $D_{\mathbf{K}} < D$.

Restriction to a translational subspace is equivalent to fixing the total momentum $\mathbf{K}$, modulo the first BZ. To apply the *Paradeisos* mapping functions in the restricted subspace, one must replace the binomial coefficients by an extended combinatorial $C_i^m(\mathbf{k})$ following the recursion

$$C_i^m(\mathbf{k}) = C_{i-1}^{m-1}(\mathbf{k}-\mathbf{k}_i) + C_{i-1}^m(\mathbf{k}), \tag{5}$$

where $C_i^m(\mathbf{k})$ counts the number of ways to arrange m electrons in a state with i-bits and a fixed total momentum $\mathbf{k}$. The initial conditions for the recursion are

$$C_i^0(\mathbf{k}) = \begin{cases} 1, \text{ if } \mathbf{k} = \mathbf{0}\ (\Gamma - \text{point}) \\ 0, \text{ otherwise} \end{cases}$$

and

$$C_i^1(\mathbf{k}) = \begin{cases} 1, \text{ if } \mathbf{k} = \mathbf{k}_i \\ 0, \text{ otherwise.} \end{cases}$$

Following these rules and Eq. (3), an n fermion state with occupation $j_0 < j_1 < \cdots < j_{n-2} < j_{n-1}$ and corresponding total momentum $\mathbf{K} = \mathbf{k}_{j_0} + \cdots + \mathbf{k}_{j_{n-1}}$ has an index (within the momentum subspace $D_{\mathbf{K}}$)

$$\begin{aligned}\mathrm{idx}(\varphi) &= C_{j_0}^1(\mathbf{k}_{j_0}) + C_{j_1}^2(\mathbf{k}_{j_0} + \mathbf{k}_{j_1}) + \cdots \\ &\quad \cdots + C_{j_{n-2}}^{n-1}(\mathbf{K} - \mathbf{k}_{j_{n-1}}) + C_{j_{n-1}}^n(\mathbf{K}).\end{aligned} \tag{6}$$

One can precompute the coefficients $C_i^m(\mathbf{k})$ using dynamical programming and store them in a lookup-table. A pseudocode description of this forward mapping in the restricted subspace would be

***Paradeisos* k-space forward map**
Goal: For n electrons and N momenta, map $\varphi \mapsto \mathrm{idx}(\varphi)$

```
function forward_map_k(φ)
    declare idx = 0, m = 0, k=0.
    for i = 0 to N − 1:
        if ith bit in φ is TRUE:
            m ← m + 1
            k ← mod(k + k_i,BZ)
            idx ← idx + C_i^m(k)
        end
    end
    return idx, k
```

Following a similar procedure as in the full Hilbert space, the momentum restricted subspace backward mapping has a pseudocode

***Paradeisos* k-space backward map**
Goal: For n electrons, N momenta,
and total momentum $\mathbf{K}$, map idx(φ) $\mapsto \varphi$

function *backward_map_k* (idx; n, N, $\mathbf{K}$)
  declare $\varphi = 00\dots00_2$, $\mathbf{k} = \mathbf{K}$, $m = n$, $p = \mathrm{idx}$
  for $i = N-1$ to 0:
    if $p \geq C_i^m(\mathbf{k})$:
      set φ's ith bit to TRUE
      $p \leftarrow p - C_i^m(\mathbf{k})$
      $m \leftarrow m - 1$
      $\mathbf{k} \leftarrow \mathrm{mod}(\mathbf{k} - \mathbf{k}_i, \mathrm{BZ})$
    end
  end
  return φ

The spin-full restricted momentum subspace can no longer be regarded as a direct product space between $\uparrow$ and $\downarrow$ momentum subspaces. Instead, the index of a basis element depends explicitly on both the $\uparrow$ and $\downarrow$ configurations. Noting the normal ordering convention with the $\downarrow$ bits followed by those for $\uparrow$ right-to-left, the full forward mapping procedure reduces to first performing a forward mapping for the $\downarrow$ portion of the basis element. The pseudocode for treating the remainder of the forward mapping including the $\uparrow$ portion of a basis element can be written as

***Paradeisos* k-space forward map (spin-full)**
Goal: For $n_\uparrow + n_\downarrow$ electrons and N momenta, map

function *forward_map_k_spin* ($\varphi_\uparrow$, $\varphi_\downarrow$)
  declare idx = 0, $m = 0$, $\mathbf{k}$=0.
  call *forward_map_k* ($\varphi_\downarrow$): idx$_\downarrow$, $\mathbf{K}_\downarrow$; $\mathbf{k} \leftarrow \mathbf{K}_\downarrow$.
  for $i = 0$ to $N-1$:
    if ith bit in $\varphi_\uparrow$ is TRUE:
      $m \leftarrow m + 1$
      $\mathbf{k} \leftarrow \mathrm{mod}(\mathbf{k} + \mathbf{k}_i, \mathrm{BZ})$
      $\mathrm{idx} \leftarrow \mathrm{idx} + \widetilde{C}_i^m(\mathbf{k}; n_\downarrow)$
    end if
  end
  return $\mathrm{idx} + \mathrm{idx}_\downarrow$

Here one must work with an additional modification of the combinatorial function, which accounts for the spin-down combinations. This modified version takes the form

$$\widetilde{C}_i^m(\mathbf{k}; n_\downarrow) = \sum_{\mathbf{k}_\downarrow} C_i^m(\mathbf{k} - \mathbf{k}_\downarrow) C_N^{n_\downarrow}(\mathbf{k}_\downarrow). \tag{7}$$

Again, these quantities may be precalculated and stored in a look-up table. A similar modification applies for the spin-full backward mapping whose pseudocode becomes

***Paradeisos* k-space backward map (spin-full)**
Goal: For $n_\uparrow + n_\downarrow$ electrons, N momenta,
and total momentum $\mathbf{K}$, map idx(φ) $\mapsto \varphi_\uparrow \otimes \varphi_\downarrow$

function *backward_map_k_spin* (idx; $n_\uparrow$, $n_\downarrow$, N, $\mathbf{K}$)
  declare $\varphi_\uparrow = 00\dots00_2$, $\mathbf{k} = \mathbf{K}$, $m = n_\uparrow$, $p = \mathrm{idx}$
  for $i = N-1$ to 0:
    if $p \geq \widetilde{C}_i^m(\mathbf{k}; n_\downarrow)$:
      set $\varphi_\uparrow$'s ith bit to TRUE
      $p \leftarrow p - \widetilde{C}_i^m(\mathbf{k}; n_\downarrow)$
      $m \leftarrow m - 1$
      $\mathbf{k} \leftarrow \mathrm{mod}(\mathbf{k} - \mathbf{k}_i, \mathrm{BZ})$
    end if
  end
  call *backward_map_k* (p; $n_\downarrow$, $\mathbf{k}$): $\varphi_\downarrow$
  return $\varphi = \varphi_\uparrow \otimes \varphi_\downarrow$

4.2. Inversion symmetry

Inversion symmetry ($\mathcal{F}$) implies that a transformation $\mathbf{r} \to -\mathbf{r}$ (or $\mathbf{k} \to -\mathbf{k}$) leaves the Hamiltonian unchanged. To simplify the presentation and without loss of generality, we confine our discussion to one-dimension in real space for a lattice (chain) with N sites indexed sequentially $0, \dots, N-1$. Inversion acting on the creation and annihilation operators appearing in the Hamiltonian returns $\mathcal{F}\left(c_i^\dagger\right) \to c_{N-1-i}^\dagger$ and $\mathcal{F}(c_i) \to c_{N-1-i}$. Since $\mathcal{F}^2 = I$, $\mathcal{F}$ has eigenvalues ± 1, which decomposes the Hilbert space into two orthogonal subspaces, corresponding to symmetric and antisymmetric combinations of the basis elements. Note that certain basis elements may already be symmetric in $\mathcal{F}$. Based on our conventions, the goal will be to compute the index of a given bit sequence representing the *symmetric* subspace (a similar procedure can be employed for the antisymmetric subspace).

For n electrons on N sites, partition the representatives into three classes according to their bits in positions $N-1$ and 0:

$$\begin{aligned} &\text{A:}\ |\varphi_A\rangle := \overset{N-1}{0} \dots \overset{i=0}{1_2} \\ &\text{B:}\ |\varphi_B\rangle := \overset{N-1}{0} \dots \overset{i=0}{0_2} \\ &\text{C:}\ |\varphi_C\rangle := \overset{N-1}{1} \dots \overset{i=0}{1_2} \end{aligned}$$

The symmetric states in class A are of the form

$$|\varphi_A^s\rangle := \left(0\, a_{N-2} \dots a_1\, 1_2 + 1\, a_1 \dots a_{N-2}\, 0_2\right)/\sqrt{2},$$

which obviates the need for bit sequences of the form $\overset{N-1}{1} \dots \overset{i=0}{0_2}$ that are included by construction. Within class A, the index can be determined by standard *Paradeisos* (forward mapping without symmetry). The index for elements in classes B and C can be determined by recursive partitioning for bits in positions $N-d$ and $d-1$, where d is the recursion depth. For each recursion step the same classification scheme can be employed, which results in a tree with class A, B, or C nodes. Fig. 2 shows such a tree to illustrate this process.

There are a few things to note about this construction. First, each class A node is a leaf node (which contains a set of states) – the recursion ends and *Paradeisos* forward mapping can be used to determine the index at that depth. Leaf nodes can exist for class B and C, but only if the maximum recursion depth has been reached based on the number of bits N in each sequence. Those leaves that end with B or C have a palindrome pattern, containing only one state. The height of the tree is $N/2+1$ for N even or $(N+1)/2$ for N odd, given that the number of bits yet to be classified at each recursion step decreases by 2.

By convention, the indices for class A are the smallest, followed by those for class B, and lastly those for class C, at the same depth on the classification tree. In this way, the standard order has been destroyed, and replaced by a set of continuous indices for all the representative elements in the symmetric subspace. One observes that class A contains C_{i-2}^{m-1} representatives (the same number for either the symmetric or antisymmetric subspaces by construction) for i "active" bits and m non-zero bits at the current depth, as the inner bit sequence has no bearing on the A, B, or C classifications. Similarly, the number of representatives at a particular level in class B or C can be determined recursively by $S_i^m = C_{i-2}^{m-1} + S_{i-2}^m + S_{i-2}^{m-2}$, where S_i^m is the total number of representative elements in the symmetric subspace for i total bits and m non-zero bits. The three terms represent the number of elements for class A, B and C, respectively, at the next lower level. S_i^m can be calculated in a bottom-up manner using dynamical programming. In this way, the number of preceding basis elements can be calculated by a depth-first search of the tree, counting C_{i-2}^{m-1} elements for a class A leaf and a single basis element for class B or C leaves.

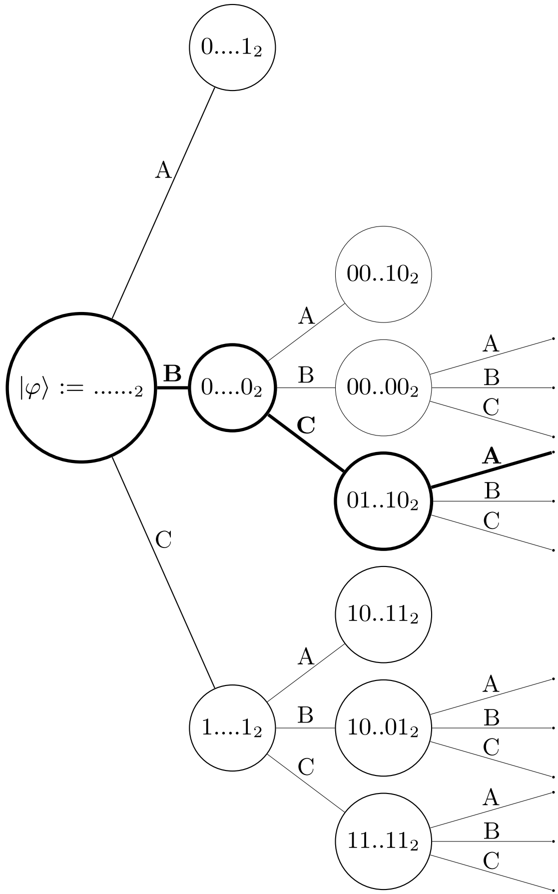

Fig. 2. A tree up to depth $d = 4$ (bits not shown for brevity) in the recursive classification of a bit sequence representing a basis element in the symmetric subspace. Branches typically terminate at an A class bit sequence from which the standard *Paradeisos* forward mapping can be used to obtain the index of the state at that recursion depth. The highlight shows the path to a state such as 010110_2, for example.

***Decision Tree* for inversion symmetry**
Goal: construct root and node *structure*

@ depth d:
  node.φ_L (d leftmost bits in φ)
  node.φ_R (d rightmost bits in φ)
  node.count (number of representatives)
  node.A (pointer to depth $d+1$, class A)
  node.B (pointer to depth $d+1$, class B)
  node.C (pointer to depth $d+1$, class C)
  node.parent (pointer to depth $d-1$)

function *decision_tree* (n, N)
  for $d = N/2$ to 0: (N even)
    construct *root* and *nodes* @ d
      assign node.$\varphi_L(\varphi_R)$
      assign node.A(B,C)
      assign node.parent
    calculate node.count
      C_i^m (class A)
      S_i^m (class B or C)
  end
  return *root*

Unfortunately, one cannot simultaneously obtain the benefits of using both translation and inversion symmetry in *Paradeisos*. Invoking translation symmetry typically reduces the effective Hilbert space dimension much more than the reduction obtained from the inversion symmetric subspace. However, the inversion symmetric algorithm may be employed for some problems that lack translation invariance, such those with open boundaries.

To be complete, we present pseudocode for the forward and backward mapping *Paradeisos* algorithms with inversion symmetry.

***Paradeisos* inversion forward map**
Goal: For n electrons and N sites, map $\varphi \mapsto \mathrm{idx}(\varphi)$

function *forward_map_inversion* (φ)
  node $\leftarrow$ *decision_tree* (n, N)
  for $d = 0$ to $N/2 - 1$: (N even)
    switch(($N - 1 - d$)-th bit in φ, dth bit in φ):
    case: `FALSE`, `TRUE`:
      $\phi \leftarrow$ (d+1)-th through ($N - 2 - d$)-th bits in φ
      idx $\leftarrow$ idx + *forward_map*(ϕ)
      return idx
    case: `FALSE`, `FALSE`:
      idx $\leftarrow$ idx + node.A.count
      node $\leftarrow$ node.B
    case: `TRUE`, `TRUE`:
      idx $\leftarrow$ idx + node.A.count + node.B.count
      node $\leftarrow$ node.C
  end
  return idx

***Paradeisos* inversion backward map**
Goal: For n electrons and N sites, map $\mathrm{idx}(\varphi) \mapsto \varphi$

function *backward_map_inversion* (idx; n, N)
  node $\leftarrow$ *decision_tree* (n, N), p = idx
  for $d = 0$ to $N/2 - 1$: (N even)
    if $p \geq$ node.A.count + node.B.count:
      $p \leftarrow p -$ node.A.count $-$ node.B.count
      node $\leftarrow$ node.C
    elseif $p \geq$ node.A.count:
      $p \leftarrow p -$ node.A.count
      node $\leftarrow$ node.B
    else:
      node $\leftarrow$ node.A
      $m \leftarrow n -$ bit_count(node.$\varphi_L \wedge$ node.φ_R, 1)
      $\phi \leftarrow$ *backward_map*(p; m, $N - 2d$)
      return node.$\varphi_L \wedge \phi \wedge$ node.φ_R
    end if
  end
  return node.$\varphi_L \wedge$ node.φ_R
  ($\wedge$ represents concatenation of binary sequences)

5. Numerical benchmarks

The goal in devising the *Paradeisos* algorithm was to address one of the remaining bottlenecks for sparse matrix eigensolvers — memory requirements for basis element hashing in traditional binary search that lead to a large communications overhead or significant aggregate memory consumption in large-scale parallel implementations. In this section we benchmark the performance of *Paradeisos* against binary search, and investigate additional benefits when coupling the algorithm with the checkerboard decomposition for data parallelism. We concentrate specifically on how memory usage scales with the number of processors for single-band Hubbard model calculations on clusters of varying size, as the scaling slope usually determines the quality of a parallelization scheme.

In a parallel matrix–vector algorithm, the storage of a single vector of size D on each processor characterizes the typical memory cost and can usually to be used as a reference to determine

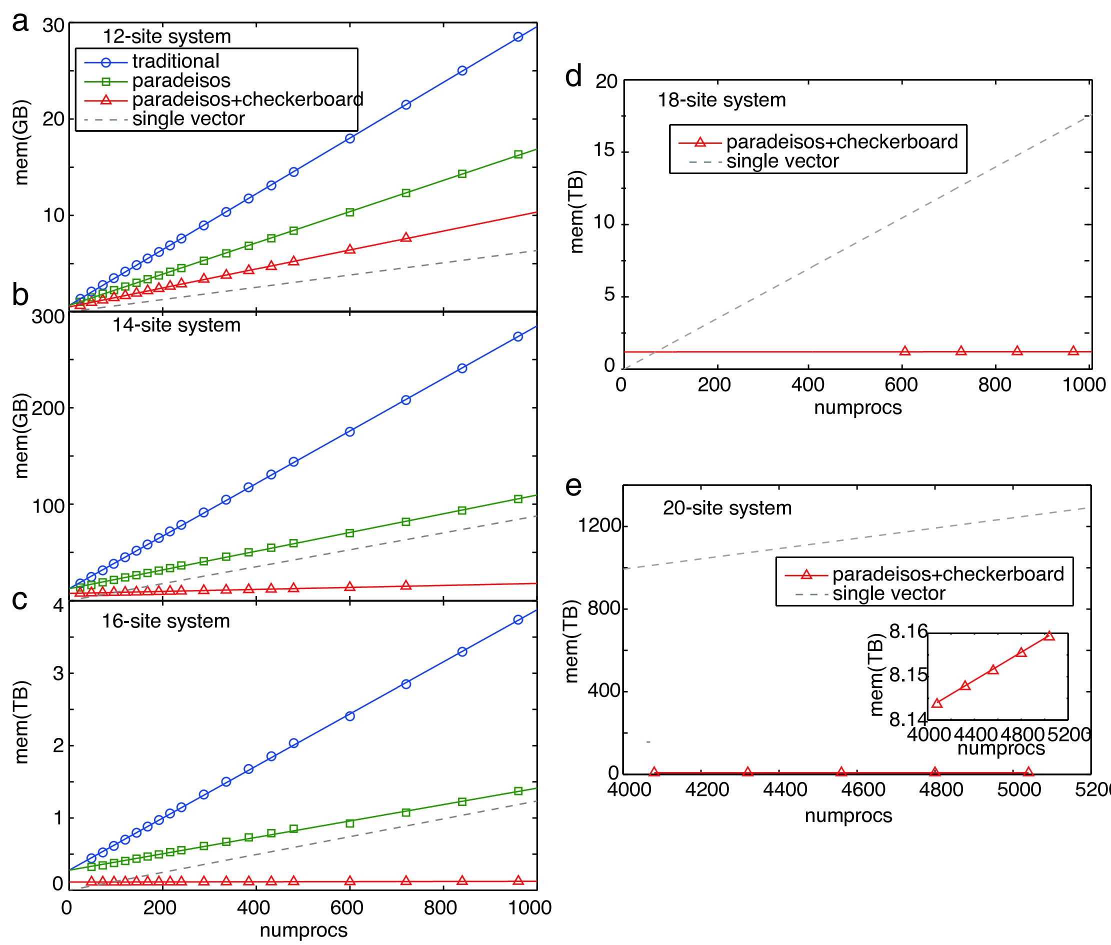

Fig. 3. Comparison of aggregate memory usage as a function of the number of processors for sparse matrix construction and diagonalization (obtaining the groundstate wavefunction), for a one-dimensional Hubbard model in real space without invoking any Hilbert space reductions due to symmetry. The blue curves correspond to traditional binary search, green curves correspond to *Paradeisos*, and red curves relate to *Paradeisos* combined with a checkboard decomposition. The dashed lines indicate the memory required to store a single vector within the Hilbert space serially for each process. Note significant differences in scale with Hilbert space dimension (lattice size). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the necessity of a sophisticated parallelization scheme like the "checkerboard" decomposition. Thus, we denote such a memory cost as gray lines in Figs. 3 and 4. As shown in Fig. 3, the traditional binary search method (blue) has a relatively large scaling slope with system size, making the parallelization cost extremely expensive for large cluster calculations. Once the overhead cost exceeds the benefits of parallelization, adding more processors would not help to solve the memory problem. *Paradeisos* (green) succeeds in reducing the slope. However, the steady increase of aggregate memory with the number of processors still limits parallelization. When one applies *Paradeisos* with the checkerboard decomposition (red), the situation changes dramatically. The scaling slope becomes relatively small and shows little change with growth in the dimension of the Hilbert space (dictated by the lattice size). The memory overhead remains effectively constant, an important consideration for parallel computations that then may be cheaply divided over many nodes or processors. For the largest problems, *Paradeisos* improves the scaling beyond that of even storing a single vector in the Hilbert space for each process.

As shown in Fig. 4, when applying symmetry reduction for small Hilbert space dimensions (the smallest lattice problems), *Paradeisos* alone performs marginally better than binary search and even when combined with the checkerboard decomposition, primarily due to residual memory overhead from other parts of the sparse matrix diagonalization code. With increasing Hilbert space dimension (lattice size), the results remain qualitatively similar to the those from the real space implementation of the algorithm without symmetry reduction. Fig. 5 compares the memory scaling slope for various lattice sizes, showing the significant improvements in memory consumption that can be realized by combining *Paradeisos* with the checkerboard decomposition.

6. Extension to spin and bosonic many-body systems

The algorithm can be extended or generalized to many-body spin and bosonic basis states, for which a particle number truncation is necessary, particularly local moment spin-$\frac{1}{2}$ systems, hard-core bosons, and phonons etc. These extensions are applicable because of the one-to-one mapping between the fermionic many-body states and spin/bosonic many-body states.

For a spin-$\frac{1}{2}$ system the spin-up (1) and spin-down (0) states for each site can be mapped directly to the occupied (1) and non-occupied (0) states of spinless fermonic system. Higher order spin systems require a more complicated generalization. For a bosonic system with N-modes (or sites) and M total bosons, any configuration can be constructed by distributing M objects among the N bins and assigning a string of 1s to represent the bosons in a bin and a 0 to represent a divider between neighboring bins. This can be mapped uniquely to a fermionic binary state of $(N+M-1)$-site M fermions. The *Paradeisos* algorithm can be employed directly by constructing these one-to-one mappings between spin/bosonic many-body states and fermionic many-body states.

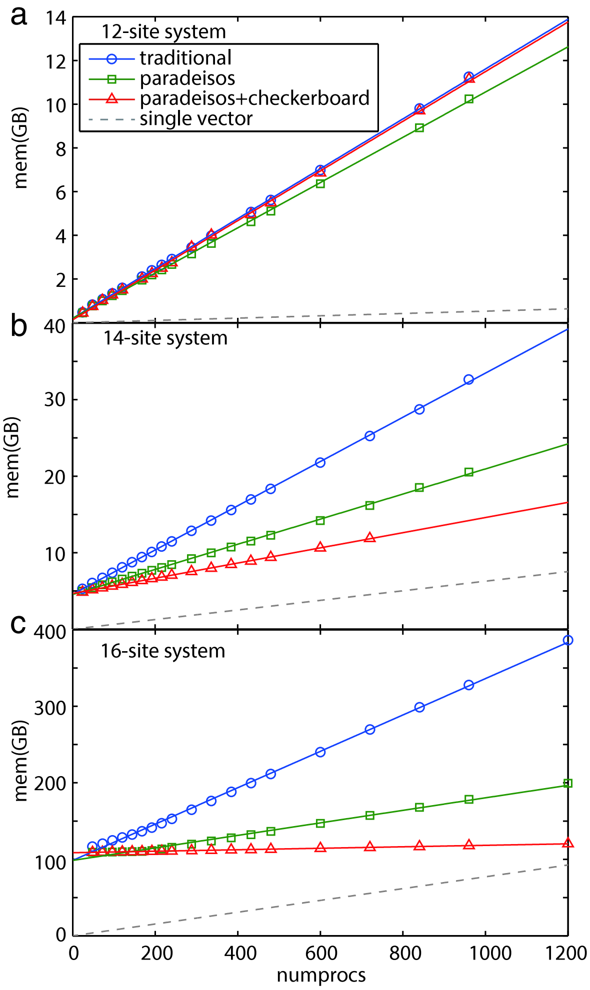

Fig. 4. Comparison between traditional binary search (blue), *Paradeisos* (green), and *Paradeisos* combined with a checkerboard decomposition (red). Each curve shows overall memory consumption for a one-dimensional Hubbard model on chains (lattices) of length (a) 12, (b) 14, and (c) 16 sites. Each one represents the memory required for solving the problem in the $\mathbf{K} = 0$ momentum subspace, which contains the groundstate wavefunction. The dashed lines indicate the memory required to store a single vector within the $\mathbf{K} = 0$ Hilbert space serially for each process. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

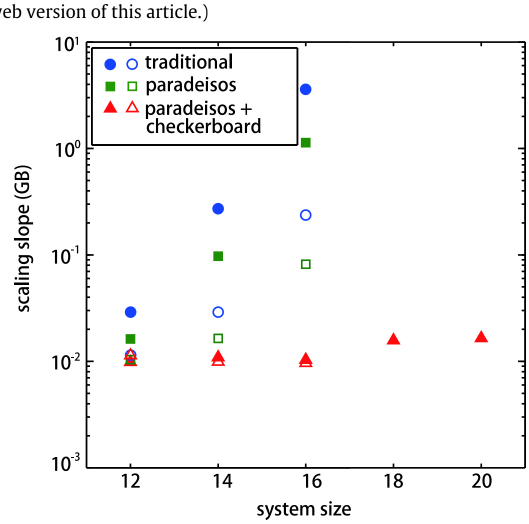

Fig. 5. Summary of the memory scaling slopes as a function of system size from Figs. 3 and 4. The solid dots represent results from the real space implementation, while the open symbols correspond to those obtained from invoking the translation symmetry reduction of the effective Hilbert space.

7. Summary and discussion

We have proposed a perfect hashing algorithm – a direct mapping between Hilbert space basis elements and their corresponding index – for use in sparse matrix eigenvalue problems. Compared to the traditional binary search, the present algorithm provides a considerable savings in aggregate memory usage without incurring additional penalties to the time complexity. Moreover, in concert with a checkerboard decomposition scheme, the memory overhead can be negligible (effectively independent of problem size), implying efficient parallelization for large size compute environment. The *Paradeisos* algorithm is also compatible with additional point group symmetries — which can highly reduce the Hilbert space dimension, and thus can be efficiently applied to most quantum many-body systems or models. Moreover, the algorithm eschews storage, and as such, may be utilized in *matrix-free* implementations of eigenvalue solvers.

Acknowledgments

This work was supported at SLAC and Stanford University by the U.S. Department of Energy, Office of Basic Energy Sciences, Division of Materials Sciences and Engineering, under Contract No. DE-AC02-76SF00515. Y.W. was supported by the Stanford Graduate Fellows in Science and Engineering, and C.M. acknowledges support from the Alexander von Humboldt foundation via a Feodor Lynen fellowship. The computational work was performed using the resources of the National Energy Research Scientific Computing Center supported by the U.S. Department of Energy, Office of Science, under Contract No. DE-AC02-05CH11231.

References

[1] H. Fehske, R. Schneider, A. Weiße (Eds.), Computational Many-Particle Physics, Springer, 2008.
[2] A.W. Sandvik, AIP Conf. Proc. 1297 (1) (2010) 135–338. http://dx.doi.org/10.1063/1.3518900. arXiv:http://aip.scitation.org/doi/pdf/10.1063/1.3518900. http://aip.scitation.org/doi/abs/10.1063/1.3518900.
[3] E. Dagotto, Rev. Modern Phys. 66 (1994) 763–840. http://dx.doi.org/10.1103/RevModPhys.66.763. https://link.aps.org/doi/10.1103/RevModPhys.66.763.
[4] A.M. Läuchli, J. Sudan, E.S. Sørensen, Phys. Rev. B 83 (2011) 212401. http://dx.doi.org/10.1103/PhysRevB.83.212401. https://link.aps.org/doi/10.1103/PhysRevB.83.212401.
[5] E. Dagotto, A. Moreo, F. Ortolani, D. Poilblanc, J. Riera, Phys. Rev. B 45 (1992) 10741–10760. http://dx.doi.org/10.1103/PhysRevB.45.10741. https://link.aps.org/doi/10.1103/PhysRevB.45.10741.
[6] D. Gotfryd, J. Rusnačko, K. Wohlfeld, G. Jackeli, J. c. v. Chaloupka, A.M. Oleś, Phys. Rev. B 95 (2017) 024426. http://dx.doi.org/10.1103/PhysRevB.95.024426. https://link.aps.org/doi/10.1103/PhysRevB.95.024426.
[7] P. Nataf, F. Mila, Phys. Rev. Lett. 113 (2014) 127204. http://dx.doi.org/10.1103/PhysRevLett.113.127204. https://link.aps.org/doi/10.1103/PhysRevLett.113.127204.
[8] J.M. Zhang, R.X. Dong, Eur. J. Phys. 31 (3) (2010) 591. http://stacks.iop.org/0143-0807/31/i=3/a=016.
[9] H.Q. Lin, Phys. Rev. B 42 (1990) 6561–6567. http://link.aps.org/doi/10.1103/PhysRevB.42.6561.
[10] C.-C. Chen, C.J. Jia, A.F. Kemper, R.R.P. Singh, T.P. Devereaux, Phys. Rev. Lett. 106 (2011) 067002. http://dx.doi.org/10.1103/PhysRevLett.106.067002. https://link.aps.org/doi/10.1103/PhysRevLett.106.067002.
[11] K. Tsutsui, T. Tohyama, S. Maekawa, Phys. Rev. Lett. 83 (1999) 3705–3708. http://dx.doi.org/10.1103/PhysRevLett.83.3705. https://link.aps.org/doi/10.1103/PhysRevLett.83.3705.
[12] X. Deng, C. Jia, C.-C. Chien, Phys. Rev. B 91 (2015) 054515. http://dx.doi.org/10.1103/PhysRevB.91.054515. https://link.aps.org/doi/10.1103/PhysRevB.91.054515.
[13] G.H. Golub, C.F. Van Loan, Matrix Computations, Vol. 3, 4th ed, JHU Press, 2013.
[14] C. Jia, K. Wohlfeld, Y. Wang, B. Moritz, T.P. Devereaux, Phys. Rev. X 6 (2016) 021020. http://dx.doi.org/10.1103/PhysRevX.6.021020. http://link.aps.org/doi/10.1103/PhysRevX.6.021020.
[15] Y. Wang, C. Jia, B. Moritz, T.P. Devereaux, Phys. Rev. Lett. 112 (15) (2014) 156402.
[16] H.A. van der Vorst, SIAM J. Sci. Stat. Comput. 13 (2) (1992) 631–644. http://dx.doi.org/10.1137/0913035.

[17] G. Karypis, V. Kumar, MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0, University of Minnesota, Minneapolis, MN, 2009 http://www.cs.umn.edu/~metis.
[18] S. Liang, Comput. Phys. Comm. 92 (1995) 11–15. http://dx.doi.org/10.1016/0010-4655(95)00108-R. http://www.sciencedirect.com/science/article/pii/0010465595500108R.
[19] C.B. Mendl, Comput. Phys. Comm. 182 (2011) 1327–1337. http://dx.doi.org/10.1016/j.cpc.2011.01.028.
[20] J. Henry S. Warren, Hacker's Delight, 2nd ed, Addison-Wesley Professional, 2012.
[21] S.E. Anderson, Bit twiddling hacks https://graphics.stanford.edu/~seander/bithacks.html.
[22] L.F. Williams Jr., Proceedings of the 14th Annual Southeast Regional Conference, ACM-SE 14, ACM, New York, NY, USA, 1976, pp. 95–101. http://dx.doi.org/10.1145/503561.503582. http://doi.acm.org/10.1145/503561.503582.
[23] G. Fano, F. Ortolani, A. Parola, Phys. Rev. B 46 (1992) 1048–1060. http://dx.doi.org/10.1103/PhysRevB.46.1048.