DanceQ: High-performance library for number-conserving bases

Robin Schäfer[1⋆], David J. Luitz[2†]

1 Department of Physics, Boston University, Boston, Massachusetts 02215, USA
2 Institute of Physics, University of Bonn, Nussallee 12, 53115 Bonn, Germany

⋆ rschaefe@bu.edu , † david.luitz@uni-bonn.de

Abstract

The complexity of quantum many-body problems scales exponentially with the size of the system, rendering any finite-size scaling analysis a formidable challenge. This is particularly true for methods based on the full representation of the wave function, where one simply accepts the enormous Hilbert space dimensions and performs linear algebra operations, e.g., for finding the ground state of the Hamiltonian. If the system satisfies an underlying symmetry where an operator with a degenerate spectrum commutes with the Hamiltonian, it can be block-diagonalized, thus reducing the complexity at the expense of additional bookkeeping. At the most basic level required for Krylov space techniques (like the Lanczos algorithm), it is necessary to implement a matrix-vector product of a block of the Hamiltonian with arbitrary block-wavefunctions, potentially without holding the Hamiltonian block in memory. An efficient implementation of this operation requires the calculation of the position of an arbitrary basis vector in the canonical ordering of the basis of the block. We present here an elegant and powerful, multi-dimensional approach to this problem for the $U(1)$ symmetry appearing in problems with particle number conservation. Our divide-and-conquer algorithm uses multiple subsystems and hence generalizes previous approaches to make them scalable. In addition to the theoretical presentation of our algorithm, we provide DanceQ, a flexible and modern – header only – `C++20` implementation to manipulate, enumerate, and map to its index any basis state in a given particle number sector as open source software under https://DanceQ.gitlab.io/danceq.

Contents

1 Introduction

For quantum many-body problems, the size of the Hilbert space grows exponentially with the size of the system. Since there are only a handful of exactly solvable and non-trivial interacting models [1–3], we have to rely on approximations of various degrees of sophistication [4–6] and numerical methods [7–10] to study interacting systems. Numerical approaches started to pick up momentum in the 1950s with the increasing availability of computational power motivating new algorithmic developments of particular relevance for condensed matter physics [11–14] and the subsequent birth of computational physics [15–19]. In particular, it became possible to compute the spectrum of small but generic interacting many-body systems [20]. In 1958, for example, R. Orbach used an *IBM 701* to compute eigenvalues of a chain of ten spins [21]. The success of numerical simulations of one-dimensional systems [22–25] quickly swapped over to higher dimensions [26–28] due to the exponentially growth in computer power [29, 30]. This steady growth of computer power makes it possible today to compute ground states for magnetic systems containing up to 50 spin-1/2 particles [31–35] with total Hilbert space dimensions exceeding 10^{15}.

Brute force methods directly tackle the exponentially increasing complexity of the Hilbert space by encoding all details of the wave function fall under the category of *exact diagonalization*. Compared to other computational techniques frequently used in the field [8, 9], their advantages are their wide applicability and unbiased nature, particularly for cases where wave functions are strongly entangled. Naturally, the exponential growth in complexity of the problem imposed by quantum mechanics is a major obstacle to the solution of larger systems, which, in turn, are required for a valid finite size scaling analysis to address the thermodynamic limit. Therefore, any reduction of the problem – such as by exploiting symmetries to block-diagonalize the Hamiltonian – should be employed. Besides lattice symmetries that depend on the precise geometry of the problem [33–38], more intrinsic properties independent of the spatial structure play a crucial role in many physical systems. One such property is the conservation of the particle number, which leads to the simplest scheme for block-diagonalization, which is the focus of this work. Number conservation naturally arises in simple tight-binding type models [39] and in magnetic spin systems where the equivalent symmetry is related to the total magnetization. While organizing and managing the basis states may seem straightforward at first glance, the task becomes increasingly complex as the number of particles and system size grows, as outlined below [7, 40, 41].

In this work, we present an efficient algorithm to handle and organize the basis states of number-conserving systems based on a general divide-and-conquer approach. Specifically, consider a system consisting of L individual sites, where each site hosts a quantum degree of freedom (qudit) with a local Hilbert space dimension Q with the basis states

$$|\sigma_i\rangle \in \{|0\rangle, |1\rangle, \dots, |Q-1\rangle\}. \tag{1}$$

The total *particle number* of the many-body system is

$$n = \sum_{i=1}^{L} \sigma_i\,. \tag{2}$$

In the language of Bosons, σ_i refers to the number of particles located at a discrete lattice site where the maximal number of particles per site is $Q-1$. Alternatively, we can think of the total magnetization of L spin-S instances with $2S+1=Q$. In this scenario, the total particle number is replaced by the total magnetization along the z-axis:

$$S^z_{\text{tot}} = \sum_{i=1}^{L} S^z_i\,. \tag{3}$$

Throughout the manuscript, we mainly use the Bosonic language and label the Hilbert space sectors by the particle number n or filling fraction $f := \frac{n}{L(Q-1)}$ defined with respect to the maximal particle number $L(Q-1)$. The filling fraction can be translated to the total magnetization in the spin language: $S^z_{\text{tot}} = SL(2f-1)$. The half-filling case refers to the zero-magnetization sector.

Our algorithm efficiently manages the comprehensive organization and manipulation of these basis states, which is a crucial element for methods based on exact diagonalization.

Naively organizing all basis states with a fixed particle number in a list, hash tables [42], or in lexicographical order [43] quickly suffers from an exponentially increasing overhead. For example, considering $L = 32$ spin-1/2 particles at half-filling allocates approximately 18 GiB of additional memory, which may be needed elsewhere. To overcome this barrier and to make larger systems accessible, Lin [7,40] proposed the decomposition into two subsystems reducing the memory consumption of lookup tables from $\mathcal{O}\left(e^L\right)$ to $\mathcal{O}\left(e^{L/2}\right)$ (a high-performance implementation is for example provided in Ref. [44]). However, with the advancement of technology and massive parallelization over the past decades, even larger systems have become accessible, necessitating an even greater compression of lookup tables in massively parallel codes. Inspired by Lin's approach, we have generalized this idea into a "divide-and-conquer" ansatz, allowing the decomposition into N subsystems yielding a reduction of $\mathcal{O}\left(e^{L/N}\right)$.

The newly achieved reduction is extremely important for large, dilute systems and for massively parallel *sparse* and *matrix-free* applications. In the former case, the critical bottleneck in state enumeration can be circumvented, while in the latter case, the reduced required storage for index lookup makes it possible for each worker, dealing with a part of the Hilbert space, to hold thread-local lookup tables for fast and synchronization-free state-to-index mapping (details below).

We have integrated our multi-dimensional search algorithm into a modern `C++20` implementation — **DanceQ** [45,46] available as open source software under https://gitlab.com/DanceQ/danceq — capable of generating arbitrary particle number preserving Hamiltonians for arbitrary Q.

It features both Message Passing Interface (MPI) and openMP implementations. Our implementation features a MPI-based, matrix-free version of the Lanczos algorithm for ground-state searches [11], along with a frontend for advanced parallel libraries such as `Petsc` [47,48] and `Slepc` [49,50]. It provides a user-friendly interface that is ready to exploit the full potential of current high-performance computing facilities.

While our motivation is driven by the application to physical systems, the problem of efficiently computing a lexicographic one-to-one mapping is a well-known problem in computer science and combinatorics and referred to as *enumerative encoding* [51,52]. Our generic algorithm and previous variants [7,40,53], can be derived from the general ansatz provided by Cover in 1973 [52], a link we establish in Sec. 3.4.

This paper is organized as follows: In Sec. 2, we introduce the problem and the desired features needed to efficiently construct an operator acting on a particle number sector. Then, Sec. 3 focuses on our divide-and-conquer algorithm. We start by discussing a concrete example followed by the general algorithm. For further clarification, we present two important limits: Lin's original proposal [7,40] with two subsystems and the limit dividing the system into L subsystems containing a single site each [53]. Next, we refer to Cover's formulation [52]. Sec. 4 introduces the core modules and usage of the DanceQ library. It discusses different implementations of the lookup tables and analyzes their performance in order to identify the optimal choice of partitioning. Next, we benchmark the performance of matrix-free matrix-vector multiplication. The extensive documentation [46] offers further information for using DanceQ and provides numerous examples. Lastly, Sec. 5 summarizes our work.

2 Overview

This section briefly introduces the problem and the most important features of the code necessary to carry out a (matrix-free) matrix-vector product using parallel working threads.

2.1 The problem

We begin with a single lattice site with Q degrees of freedom, corresponding to a local Hilbert space dimension Q, and label the basis states by $|0\rangle, \dots, |Q-1\rangle$. A product state of the full system composed of L such sites is represented by the tensor product of basis states of the individual sites:

$$|\vec{\sigma}\rangle := \bigotimes_{i=1}^{L} |\sigma_i\rangle = |\sigma_1; \dots; \sigma_L\rangle \text{ with } |\sigma_i\rangle \in \{|0\rangle, \dots, |Q-1\rangle\} \,. \tag{4}$$

This induces a total Hilbert space dimension of Q^L for the full system of L sites. For systems with particle number conservation, it is useful to systematically focus on states with a fixed particle number $n \in \{0, \dots, (Q-1)L\}$:

$$\hat{n}|\sigma_1; \dots; \sigma_L\rangle = \left(\sum_{i=1}^{L} \sigma_i\right)|\vec{\sigma}\rangle = n|\vec{\sigma}\rangle \,. \tag{5}$$

The number of such basis states with fixed particle number n is the dimension of the corresponding symmetry sector. For the case $Q = 2$, it is well known that the number of basis states for n particles on a total of L is given by

$$D_{Q=2}(L,n) = \binom{L}{n}, \tag{6}$$

since it corresponds to the number of distinct ways to distribute n indistinguishable items (particles) on L sites.

For the general case with arbitrary $Q \geq 2$, the dimension of the symmetry sector with n particles on L sites is given by

$$D_Q(L,n) = \sum_{k=0}^{\lfloor n/Q \rfloor} (-1)^k \binom{L}{k} \binom{L-1+n-Qk}{L-1}, \tag{7}$$

where $\lfloor \bullet \rfloor = \text{floor}(\bullet)$ is the lower Gauss bracket defined by the integer part of the argument. We provide an explicit elementary derivation of this result in Appendix A in the appendix. The result in Eq. (7) was proven by Ref. [54] (cf. Eqs. (11), (12) in [54]), where it was also traced back to early work by De Moivre. It has also been used to enumerate permanents in Bosonic systems [55, 56] and was derived in an alternative way by Ref. [57] in appendix B thereof.

In order to represent wave functions as vectors and operators as matrices on a computer, it is necessary to impose a *canonical order* of all $D_Q(L,n)$ basis states in a symmetry sector. This order can be arbitrary but must not be changed during the calculation. In condensed matter physics and chemistry, the regime of interest is typically large L and n, and hence, the goal is to obtain, for example, the low energy behavior of a model Hamiltonian, i.e., to calculate the ground state in a given particle number sector. This can be achieved using Krylov space techniques like the Lanczos algorithm [11, 12, 58], for which it is sufficient to be able to calculate the action of the Hamiltonian H on an arbitrary many-body wavefunction $H|\psi\rangle$, without storing the (large and usually very sparse) matrix representation of H.

To carry out the matrix vector product $H|\psi\rangle$ efficiently, it is crucial to be able to access basis states by their index, i.e., the forward map

$$\text{index} \rightarrow |\sigma_0, \sigma_1, \ldots, \sigma_{L-1}\rangle, \tag{8}$$

as well as to retrieve the index of a given basis state, i.e., the reverse map

$$|\sigma_0, \sigma_1, \ldots, \sigma_{L-1}\rangle \rightarrow \text{index}, \tag{9}$$

because the action of an off-diagonal matrix element of H effectively changes the basis state, and we have to determine the corresponding row index in the result vector. This task can in principle, be fulfilled by a lookup table of size $D_Q(L,n)$ for the forward lookup (state from index) and a lookup table of size Q^L for the reverse lookup (index to state), but this requires an exponential memory overhead (by far exceeding the memory needed for storing wavefunctions) and the goal of divide-and-conquer approaches as the one presented here is precisely to avoid this overhead. Note that even though a forward lookup table of size $D_Q(L,n)$ for the map

$$\text{index} \rightarrow |\sigma_0, \sigma_1, \ldots \sigma_{L-1}\rangle \tag{10}$$

can in principle be stored (its size is the size of a wavefunction in the n particle sector), a simple binary search in this table for reverse lookup of cost $O(\ln D_Q(L,n))$ (memory access) is expensive.

2.2 The code

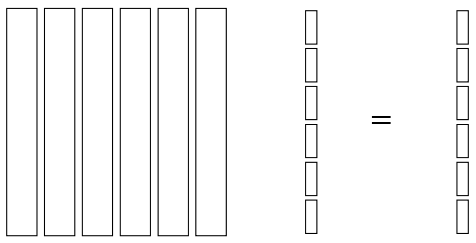
An efficient code requires a versatile enumeration scheme for the basis states. In the absence of number conservation, one can either use simple integer counting or implement more advanced schemes like Gray codes [59, 60]. The DanceQ library generates all basis states in any given particle number sector to represent the corresponding block of an operator in this sector. A parallelized program requires three different functions:

(i) `get_index`($|\vec{\sigma}\rangle$)
Maps a valid (correct particle number) basis state $|\vec{\sigma}\rangle$ to a unique index in the canonical basis order ranging from 0 to $D_Q(L,n)-1$.

(ii) `increment`($|\vec{\sigma}\rangle$)
Returns the next valid basis state in the canonical basis order such that
`get_index`($|\vec{\sigma}'\rangle$) = `get_index`($|\vec{\sigma}\rangle$)+1
with $|\vec{\sigma}\prime\rangle$ = `increment`($|\vec{\sigma}\rangle$).

(iii) `get_state`(k)
The reversed mapping of function (i), i.e., it returns the basis state with a given index k in the canonical basis order, such that
`get_index`(`get_state`(k)) = k.

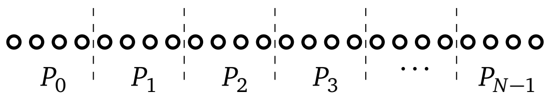
The matrix-free matrix-vector product $H|\psi\rangle$ for any wave function $|\psi\rangle$ with coefficients $\langle \vec{\sigma} \,|\, \psi \rangle$ is generated by iterating over all basis states of the particle number sector using function (ii). Function (iii) is only executed to obtain the initial state, which becomes non-trivial in parallel programs, in which each worker transverses a different segment of the basis. An operator in matrix form is obtained by applying it to a specific state defining the current row. Then, function (i) is applied to obtain the respective column indices. Pictorially, a parallel program would split the Hamiltonian matrix into rectangular blocks (left) and the input wave function vector (center) and output vector (right) into subvectors like this:



While functions (i) and (ii) are frequently executed during the construction of the operator, function (iii) is only used once per worker process. Each worker handling one consecutive part of the basis (consecutive rows in the input wave function) executes the function (iii) in the beginning to access its part. Pseudo codes for each function (Algo. 2, Algo. 3, Algo. 4) are attached in the Appendix B in the appendix.

3 The algorithm

The key idea to efficiently handle fixed-n basis states $|\sigma_1, \sigma_2 \dots \sigma_L\rangle$ and to overcome the exponential memory overhead is a “divide-and-conquer” ansatz where we divide the whole system of size L into a partition with N subsystems.



We note that despite the pictorial representation in one dimension, this technique can be used for any geometry of the physical system. It is, however, crucial to introduce an order of the sites in the system, and this is reflected in the partitioning. The index of a specific state is obtained by adding contributions from the individual subsystems, as we elaborate on in the following section.

We label the subsystems by P_k, where k indicates the *k-th* part. Because our basis states $|\vec{\sigma}\rangle$ are simple product states of single site states, they are also products of the individual subsystem states:

$$|\vec{\sigma}\rangle = |\vec{\sigma}^{(0)}\rangle_{P_0} \otimes |\vec{\sigma}^{(1)}\rangle_{P_1} \otimes \cdots \otimes |\vec{\sigma}^{(N-1)}\rangle_{P_{N-1}} . \qquad (11)$$

The remainder of this section is structured as follows. We begin by discussing an illustrative example using three subsystems, which are depicted in Fig. 1, Fig. 2, and Fig. 3. After the example, we discuss the generalization of the algorithm to N subsystems. To connect to prior work, we present two important limits, including Lin’s original approach [7], which corresponds to the case of two subsystems and the limit of $N = L$ subsystems [53] of size one which both follow trivially from the general formalism.

3.1 A concrete example

To understand the general idea of the multidimensional index lookup, it is useful to begin with an illustrative example. We consider a system of $L = 9$ sites with $Q = 2$ and a total of $n = 4$ particles, split into $N = 3$ subsystems which we label $A \equiv P_0$, $B \equiv P_1$, $C \equiv P_2$ for simplicity. The total number of states is $D_2(9,4) = 126$. We take all systems to have the same length $L_A = L_B = L_C = 3$. While the allowed states of the total system are limited by the fixed particle number $n = 4$, each subsystem can in principle be in any of the $Q^{L_A} = 2^3 = 8$ states: $|000\rangle$, $|001\rangle$, $|010\rangle$, $|100\rangle$, $|011\rangle$, $|101\rangle$, $|110\rangle$, and $|111\rangle$, however, if A is in state $|111\rangle$, B can only be in $|000\rangle$, $|001\rangle$, $|010\rangle$, or $|100\rangle$ due to the global constraint and it is precisely this kind of restriction which we need to deal with when enumerating all valid states.

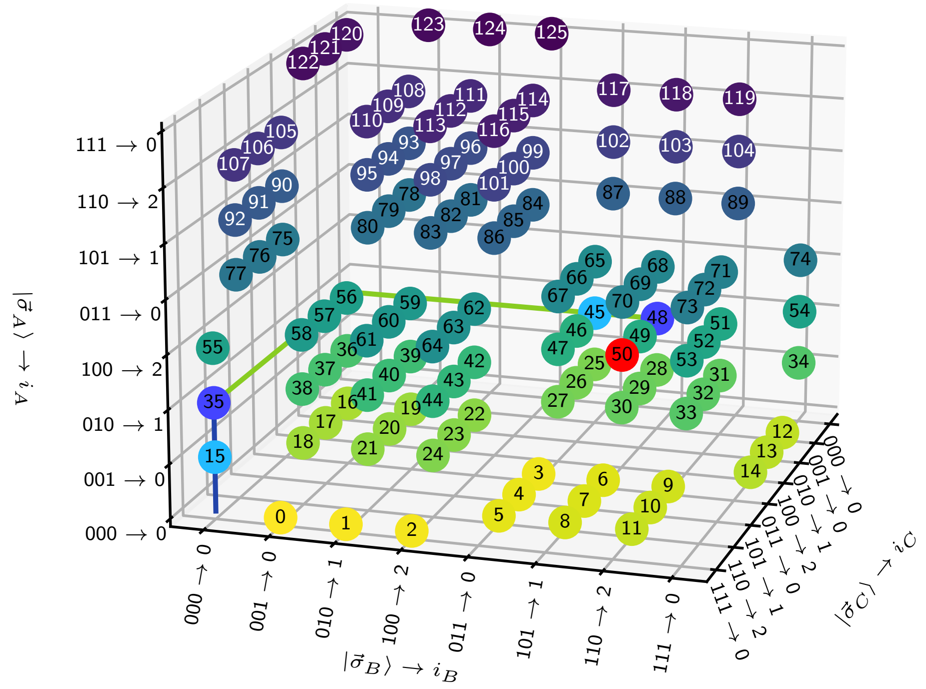


Figure 1: Illustration of the three-dimensional search structure emerging from three subsystems, A, B, and C. Each subsystem consists of three sites; the full system has hence length $L = 9$, and the figure shows all $D_2(9,4) = 126$ basis states for $Q = 2$ and $n = 4$ particles. (Light) blue and red colored balls connected by blue and green lines indicate the path to finding the index 50 of the state $|\vec{\sigma}\rangle = |010\rangle_A \otimes |101\rangle_B \otimes |100\rangle_C$ using the divide and conquer approach. We start with the state $|010\rangle_A$ on the A subsystem. It is in the $n_A = 1$ block, which has an offset of 15 (light blue). Within this block, $|010\rangle_A$ has the index $i_A = 1$, and the stride of this block is $\text{stride}_A = 20$. Hence, the contribution c_A to the final index is $c_A = \text{offset}_A + i_A\text{stride}_A = 35$ (dark blue). The state on subsystem B, $|101\rangle_B$ has the index $i_B = 1$ in the $n_B = 2$ block with $\text{offset}_B = 10$ and $\text{stride}_B = 3$, yielding $c_B = 13$. This brings us to the index $c_A + c_B = 48$ (dark blue) and by finally considering $|100\rangle_C$ with index 2 in the $n_C = 1$ block (with offset 0 and stride 1 since this is the last subsystem), we get $c_C = 1$. Hence, the final result for the desired index is $c_A + c_B + c_C = 50$ (red ball).

Suppose we order the states in each subsystem by the number of particles in the subsystem (the above list is already ordered in this way), and plot the subsystem states in the x, y, and z axes of the 3d plot in Fig. 1. In that case, we can enumerate all allowed states $|\vec{\sigma}\rangle_A \otimes |\vec{\sigma}\rangle_B \otimes |\vec{\sigma}\rangle_C$ and draw a point at the appropriate position of the coordinate system along with the corresponding index of the obtained state in the full basis. The emerging structure in Fig. 1 are dense blocks of states, while the voids between the blocks correspond to states that do not fulfill the global constraint $n = 4$. Each dense cuboid block is made from all states with fixed subsystem particle numbers, i.e., with fixed (n_A, n_B, n_C). This structure highlights the importance of ordering the subsystem bases by particle numbers and makes the key concept clear: We now have a structure of dense blocks of states in which we can efficiently retrieve the index of any state if we are able to skip all prior blocks in a straightforward way. To do this, we first explain how we organize the global basis states, i.e., in which sequence we walk through the structure shown in Fig. 1.

We choose to first increment the state of the C subsystem, keeping the order of states organized by the subsystem particle number n_C, as pointed out before. Once the subsystem

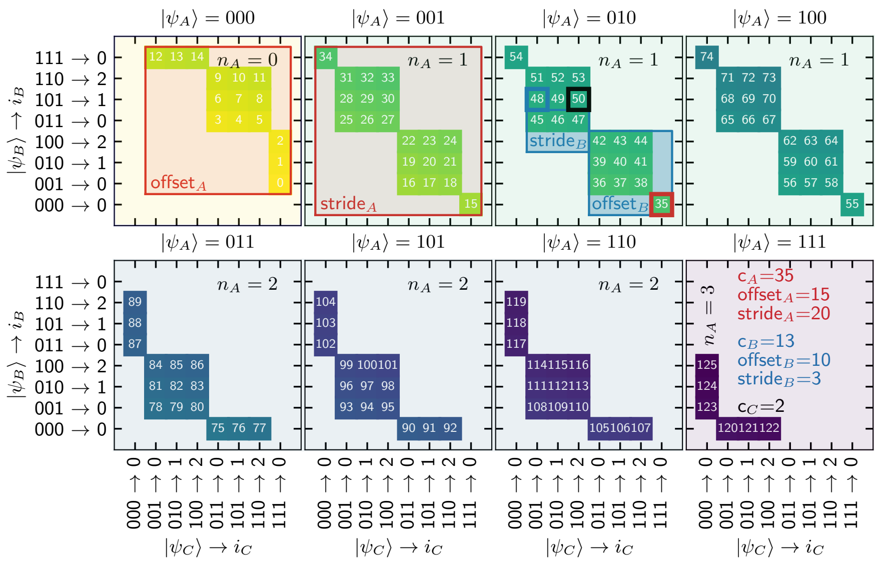


Figure 2: Indexing of all basis states in the $n = 4$ particles sector on $L = 9$ sites with $Q = 2$ states each based on partitioning of the system into three subsystems A, B, and C. The subsystem basis states are grouped by the subsystem particle number, and the indices within each subsystem particle number sector are illustrated by $100 \to 2$, which means that the state $|100\rangle$ has index 2 in the sector where the subsystem has one particle. The different panels correspond to horizontal slices (one for each basis state on the A subsystem in Fig. 1). The emerging block structure in this figure is the key concept behind the algorithm; each block corresponds to fixed particle numbers for all subsystems. Since each subsystem particle number may have a different size, there is a hierarchy of offsets (first index in the global ordering where the subsystem particle number sector begins) and strides (by how much the global index grows if a subsystem state is incremented to the next legal option within the sector).

state $|\vec{\sigma}\rangle_C$ has cycled through all possible states (which sometimes are single choices as visible for states 0, 1, and 2 in Fig. 1) for fixed states on A and B, we increment the B state and only once also B has exhausted its allowed states, the A state is incremented, moving up to the next 'layer' in the z direction in Fig. 1. This imposes a hierarchy where the state on subsystem C changes the fastest when we iterate through all global basis states. The subsystem state on A is the leading part and defines horizontal cuts perpendicular to the z-axis in Fig. 1. Within each layer, fixing the subsystem state B reduces accessible basis states to a *column*, which only differ by the C state. Finally, specifying the state on subsystem C fully determines the global state (which is a point) within the layer defined by A and the column additionally defined by B.

This structure is advantageous for retrieving the index of a particular state $|\vec{\sigma}\rangle_A \otimes |\vec{\sigma}\rangle_B \otimes |\vec{\sigma}\rangle_C$ with the help of a few lookup tables. Each subsystem contributes an additive part c_A, c_B, or c_C to the final index, which is then given by the sum of these parts. Here, c_A identifies the correct layer, $c_A + c_B$ points to the beginning of the column, and $c_A + c_B + c_C$ yields the final index. In Fig. 1, we discuss the example to retrieve the index of the state $|010\rangle_A \otimes |101\rangle_B \otimes |100\rangle_C$. For this case, the correct layer is the third from the bottom and determined by the state $|\vec{\sigma}\rangle_A = |010\rangle_A$.

The contribution c_A points to the first state (with index 35) in this layer, and it is clear that c_A therefore counts the number of all states *prior* to the target layer in the first and second layers in Fig. 1. In Fig. 2, we provide a more detailed view of the same structure by showing each of the eight layers in an individual panel. $c_A = 35$ then corresponds to the first state in

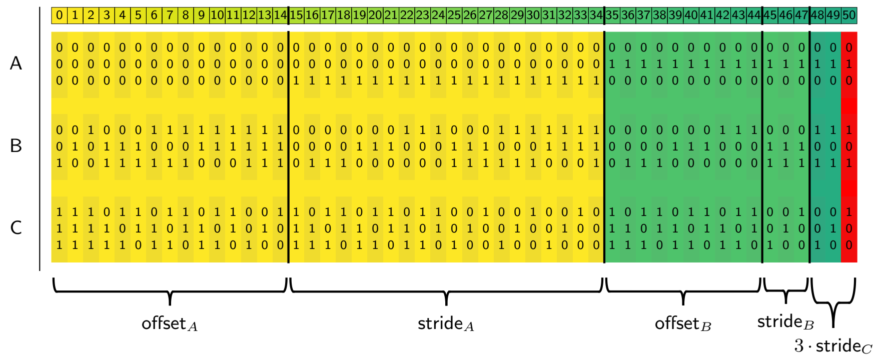


Figure 3: Example breakdown for finding the index of the 50-th state (red) $|010\rangle_A \otimes |101\rangle_B \otimes |100\rangle_C$ following the example Fig. 1 and Fig. 2. We explicitly show the first 50 states in the basis and highlight the contribution of the three subsystems A, B, and C in yellow, light green, and dark green, respectively. Beginning with $|010\rangle_A$, we find that it has $n_A = 1$, and we hence skip ahead to the first state with $n_A = 1$, which is state offset$_A(n_A, \lambda_A) = 15$. Incrementing the state in A in this sector increases the global index by 20; this is stride$_A(n_A, \lambda_A) = 20$. $|010\rangle_A$ has index$_A(\vec{\sigma}^{(A)}) = 1$, see Table 1, in the subsystem basis of the $n_A = 1$ sector and we hence skip forward to the global index $c_A = \text{offset}_A + \text{index}_A \cdot \text{stride}_A = 35$. We observe that in the slice ahead from global index 35 to our target index 50, the A state no longer changes and we can now move on to subsystem B. The state of subsystem B is $|101\rangle_B$, which is located in the $n_B = 2$ sector. We have to skip ahead the global index by offset$_B(n_B, \lambda_B) = 10$ to reach the first state in the $n_A = 1$, $n_B = 2$ sector. In this sector, we see that the global index increases by stride$_B(n_B, \lambda_B) = 3$ if we increment the B state. The state $|101\rangle_B$ has index$_B(\vec{\sigma}^{(B)}) = 1$ in the subsystem basis, again see Table 1, of the $n_B = 2$ sector and we hence have to increment the global index by $c_B = \text{offset}_B(n_B, \lambda_B) + \text{stride}_B(n_B, \lambda_B) \cdot \text{index}_B(\vec{\sigma}^{(B)}) = 13$ to reach the first state in the global basis with the correct subsystem states on A and B. This state has the index $c_A + c_B = 48$ and fulfills the constraint $|\vec{\sigma}^{(A)}\rangle = |010\rangle$ and $|\vec{\sigma}^{(B)}\rangle = |101\rangle_B$. Since C is the last subsystem, it does not have an offset, offset$_C(n_C, \lambda_C) = 0$, and its stride$_C(n_C, \lambda_C) = 1$. Therefore, incrementing the state on subsystem C directly increments the global basis index by one. Its index is index$_C(\vec{\sigma}^{(C)}) = 3$ yielding $c_C = 2$, and the final index is retrieved $c_A + c_B + c_C = 50$.

the third (target) panel in Fig. 2.

Similarly, we next consider the state on subsystem B, $|\vec{\sigma}\rangle_B = |101\rangle_B$, which allows us to skip forward in the global basis to the target column in Fig. 2 where our final state is located. The number of states to skip depends on the particles in B and A. The column determined by $|\vec{\sigma}\rangle_A$ and $|\vec{\sigma}\rangle_B$ starts with index $c_A + c_B = 48$. The third panel in Fig. 2 highlights the contribution $c_B = 13$, which can be illustrated as the number of states in the layer occurring before we reach the target state on B.

Finally, the state of the C subsystem $|\vec{\sigma}\rangle_C = |100\rangle_C$ determines the location within the column which corresponds to the index of the C state in the n_C sector of the C basis: $c_A + c_B + c_C = 50$ with $c_C = 2$.

3.2 General recipe

The idea illustrated in the previous section can be formalized and generalized to any number of subsystems and particles. Similarly to the example from Sec. 3.1, the global index of a basis state $|\vec{\sigma}\rangle$ is obtained by summing up contributions from all subsystems:

$$\text{index}(|\vec{\sigma}\rangle) = \sum_{k=0}^{N-1} c_k(n_k, \lambda_k, \vec{\sigma}^{(k)}) \,. \tag{12}$$

Each coefficient c_k is positive and depends on the number of particles n_k in the subsystem P_k, the number of particles λ_k in the *previous* subsystems $P_0 \ldots P_{k-1}$ and on the subsystem state on P_k, $|\vec{\sigma}^{(k)}\rangle$. Hence, the final index monotonically increases while traversing through the subsystems. It corresponds to the cumulative number of global basis states occurring in our chosen canonical order before the subsystem state reaches the target state. By cumulative, we mean here that we first count such states to fix the P_0 state and *from here* we start counting from zero again to determine the number of global states we have to increment before P_1 reaches the target state, i.e. we keep the state on P_0 fixed (analogous to first fixing the layer, and then counting the number of states to reach the target column in the three-dimensional example).

Therefore, we traverse through all subsystems, beginning with P_0, respecting the total particle number constraint of n. All allowed states[1] on P_0 are sorted with regard to their particle number which we denote n_0. For fixed subsystem particle number n_0, each subsystem state has a unique, zero based index $\text{index}_0(|\vec{\sigma}\rangle_0)$. An example of such an order is given in Table 1. Given a subsystem state $|\vec{\sigma}\rangle_0$ with n_0 particles on the first subsystem P_0, there are $D_Q(L-L_0, n-n_0)$ possible states in the *complement* of P_0 of size $L-L_0$ with $n-n_0$ particles that fulfill the global particle number constraint of n. The coefficient c_0 from Eq. (12) counts all states in the global basis that occur before P_0 reaches the target state. For each prior subsystem state on P_0, we hence have to take into account all configurations on the complement of P_0, which can be paired with the P_0 state while fulfilling the global constraint of fixed particle number n.

Once we have determined c_0, the P_0 state is fixed and the dimensionality of the problem is effectively reduced from N to $N-1$ subsystems. Now, the same strategy can be applied to P_1. Since we no longer have to worry about P_0, to determine c_1, we only have to count combinations with legal states in the remaining subsystems $P_2, P_3, \ldots P_{N-1}$ of total size $L-L_0-L_1$ and the effective particle number constraint is $n-n_0$ since n_0 particles are already bound to P_0. This recursive scheme is carried out through the entire system until the last subsystem P_{N-1} is reached, and the final index is recovered.

Each contribution c_k is composed of two parts: *(i)* an offset counting all basis states with subsystem particle number lower than n_k and *(ii)* a stride determined by how much the global index increases if the subsystem state is incremented within the particle number sector n_k. Together with the zero-based index of the subsystem state $|\vec{\sigma}^{(k)}\rangle$ in the subsystem particle number sector, we then have the explicit expression

$$c_k = \text{offset}_k(n_k, \lambda_k) + \text{stride}_k(n_k, \lambda_k) \cdot \text{index}_k(\vec{\sigma}^{(k)}) \,. \tag{13}$$

Here, n_k is the local particle number within the k-*th* subsystem, and λ_k is the total particle number contained in the subsystems P_0 to P_{k-1} to the left of P_k:

$$\lambda_k = \sum_{i=0}^{k-1} n_i \,. \tag{14}$$

[1] If a subsystem can host more than n particles, or if all other subsystems together can not host all n particles, the global constraint may disallow certain subsystem states and limit accessible subsystem states.

The state of the subsystem is $|\vec{\sigma}^{(k)}\rangle$ and has a zero based $\text{index}_k(\vec{\sigma}^{(k)})$ *in each* subsystem particle number sector. Similarly to the two-dimensional search [7], $\text{index}_k(\vec{\sigma}^{(k)})$ refers to a local lookup table of P_k that maps $|\vec{\sigma}^{(k)}\rangle$ to an integer running from zero to $D_Q(L_k, n_k) - 1$. The mapping within a particle number sector n_k can be arbitrary but has to be bijective such that each subsystem state maps to a unique number within the given interval. One possible lookup table for subsystems of length $L_k = 3$ with $Q = 2$ is listed in Table 1, which is used in the example Fig. 1 and Fig. 2.

n_A	$\lvert\vec{\sigma}_A\rangle$	$\text{index}_A\vec{\sigma}_A$
0	$\lvert 000\rangle$	0
1	$\lvert 001\rangle$	0
1	$\lvert 010\rangle$	1
1	$\lvert 100\rangle$	2

n_A	$\lvert\vec{\sigma}_A\rangle$	$\text{index}_A\vec{\sigma}_A$
2	$\lvert 011\rangle$	0
2	$\lvert 101\rangle$	1
2	$\lvert 110\rangle$	2
3	$\lvert 111\rangle$	0

Table 1: Example for a lookup table-for subsystem A from Fig. 1 and Fig. 2. The choice within each particle number sector can be arbitrary.

3.2.1 The Offset

To derive the expression for the required offsets, we start from the first subsystem P_0 and a given state $|\vec{\sigma}^{(0)}\rangle$ with n_0 particles. The offset counts all possible states with a lower particle number than n_0. Due to the globally fixed particle number n, there is a lower bound for the number of particles n_0^{low} that must be placed in the subsystem P_0. If the complement of P_0 is large enough to accommodate all n particles, $n_0^{\text{low}} = 0$, else, it has to reflect the fact that at least $n_0^{\text{low}} = \max(0, n - (Q-1)(L - L_0))$ need to be placed in the subsystem P_0 to satisfy the constraint. For each valid particle number k_0 on P_0, there are $D_Q(L_0, k_0)$ possible configurations for states on P_0. Each such state can be combined with any state in the *complement* (all other subsystems) of length $L - L_0$ with $n - k_0$ particles in it, and there are $D_Q(L - L_0, n - k_0)$ choices for this. Therefore, we find a total of $D_Q(L_0, k_0)D_Q(L - L_0, n - k_0)$ states with the constraints of k_0 particles in P_0 and n particles in total. In sum, to account for each *valid* subsystem particle number sector k_0 that is lower than n_0, we find

$$\text{offset}_0(n_0) = \sum_{k_0 = n_0^{\text{low}}}^{n_0 - 1} D_Q(L_0, k_0) D_Q(L - L_0, n - k_0)\,. \tag{15}$$

For the next subsystem, P_1, it is crucial to realize that the state and particle number on P_0 is already fixed, effectively reducing the dimensionality of the remaining problem by one. We, hence, only need to consider the remaining $n - n_0$ particles. The length of the complement $C_1 = \overline{P_0 \cup P_1}$ is

$$\Gamma_1 = L - L_0 - L_1\,, \tag{16}$$

and it needs to host $n - n_0 - k_1$ particles, if P_1 hosts k_1 particles. With the minimal allowed number of particles in P_1 given by $n_1^{\text{low}} = \max(0, n - n_0 - (Q-1)\Gamma_1)$, the offset for P_1 is given by

$$\text{offset}_1(n_1, \lambda_1) = \sum_{k_1 = n_1^{\text{low}}}^{n_1 - 1} D_Q(L_1, k_1) D_Q(\Gamma_1, n - \lambda_1 - k_1)\,, \tag{17}$$

where $\lambda_1 = n_0$, the number of particles already locked into P_0.

Taking the general form for the length of complement C_i of subsystem P_i,

$$\Gamma_i = L - \sum_{l=0}^{i-1} L_l\,, \tag{18}$$

we can generalize the offset for subsystem P_i to

$$\text{offset}_i(n_i, \lambda_i) = \sum_{k_i = n_i^{\text{low}}}^{n_i - 1} D_Q(L_i, k_i) D_Q(\Gamma_i, n - \lambda_i - k_i)\,. \tag{19}$$

Depending on the particle number and the subsystem length, there might be a minimal amount of n_i^{low} particles that must be placed in P_i to match the global constraint of n particles. The general form is given by $n_i^{\text{low}} = \max(0, n - \lambda_i - (Q-1)\Gamma_i)$. Importantly, for the last subsystem P_{N-1}, since the global number of particles is fixed, its particle number equals the lower bound n_{N-1}^{low} yielding $\text{offset}_{N-1}(n_{N-1}, \lambda_{N-1}) = 0$.

3.2.2 The Stride

In addition to the offsets, which bring us to the beginning of the relevant subsystem particle number sectors, we need to determine the increase in the global index if the subsystem state is incremented within the particle number sector n_k.

To understand the stride, let us start again with the first state $|\vec{\sigma}^{(0)}\rangle$ on P_0 with n_0 particles. The lookup table for P_0 assigns a unique index $i_0 = \text{index}_0(\vec{\sigma}^{(0)})$ to $|\vec{\sigma}^{(0)}\rangle$, which means that i_0 subsystem states are ranked lower than $|\vec{\sigma}^{(0)}\rangle$ within the same particle number sector n_0. Sec. 4.2 discusses the tables and their construction in more detail. As pointed out in the previous paragraph, the complement C_0 of P_0 contains all subsystems P_1 to P_{N-1} and is of size $\Gamma_0 = L - L_0$. The stride is the number of states in the complement such that the total particle number constraint n is fulfilled. For any state in P_0 with n_0 particles, this number is

$$\text{stride}_0(n_0, \lambda_0) = D_Q(L - L_0, n - n_0)\,. \tag{20}$$

Hence, the number of all possible basis states that can be constructed with the i_0 subsystem states that are ranked lower than $|\vec{\sigma}^{(0)}\rangle$ is simply $D_Q(L - L_0, n - n_0) \cdot i_0$.

Similarly to the offset, this reduces the dimensionality of the problem when we move to the second subsystem P_1. Again, we use its lookup table to obtain the index $i_1 = \text{index}_1(\vec{\sigma}^{(1)})$ of $|\vec{\sigma}^{(1)}\rangle$ with n_1 particles. Since n_0 particles are a already placed in P_1 its complement C_1 of size $\Gamma_1 = L - L_0 - L_1$ has to contain $n - n_0 - n_1$ particles leading to $(\lambda_1 = n_0)$

$$\text{stride}_1(n_1, \lambda_1) = D_Q(L - L_0 - L_1, n - n_0 - n_1) \tag{21}$$

possibilities for *each* state with n_1 particles in P_1. Therefore, the stride contribution, counting all states with the constraint $|\vec{\sigma}^{(0)}\rangle$ on P_0 and a lower index than i_1 on P_1, is

$$D_Q(L - L_0 - L_1, n - n_0 - n_1) \cdot i_1\,. \tag{22}$$

Following this scheme, we can generalize the stride contribution for the i-th subsystem with the state $|\vec{\sigma}^{(i)}\rangle$ and n_i particles. Its index is again retrieved from P_i's lookup table: $i_i = \text{index}_i(\vec{\sigma}^{(i)})$. The previous subsystems P_0 to P_{i-1} contain $\lambda_i = \sum_{i=0}^{i-1} n_i$ particles reducing the global constraint to $n - \lambda_i$ particles on P_i and its complement C_i. The general form of stride counting the number of possible states with $n - \lambda_i - n_i$ in the complement C_i of size $\Gamma_i = L - \sum_{k=0}^{i} L_k$ is

$$\text{stride}_i(n_i, \lambda_i) = D_Q(\Gamma_i, n - \lambda_i - n_i)\,. \tag{23}$$

Hence, the number of states with the constraints $|\vec{\sigma}^{(i)}\rangle$ on P_i for $i = 0, \dots, i-1$ and lower ranked subsystem states on P_i with n_i particles is $D_Q(\Gamma_i, n-\lambda_i-n_i)\cdot i_i$. Since the last subsystem does not have a complement, its stride is simply one: $\text{stride}_N\,(n_N, \lambda_N) = 1$.

We have transformed the three-dimensional example from Fig. 1 into a list shown in Fig. 3, which highlights the individual contributions in the form of offsets and strides. A detailed explanation is given in the caption.

3.3 Two important limits

Next, we want to discuss two important limits of the algorithm: $N = 2$ and $N = L$. We start by discussing the original approach by Lin [7] that is based on two subsystems. Then, we illustrate the opposite limit [53], which consists of $N = L$ subsystems of size one.

3.3.1 Two subsystems ($N = 2$)

The case with two subsystems is special as a state in P_0 with n_0 particles fixes the number of particles in P_1 due to the global constraint: $n_1 = n - n_0$. In this case, we can store the individual contributions c_0 and c_1 directly into two lookup tables that label the local basis states as shown in Table 1. Since P_1 is the last subsystem, the offset is zero, and the stride is always one. Hence, c_1 reduces to $\text{index}_1(\vec{\sigma}^{(1)})$, simply the bare lookup table we discussed. This corresponds to system A with $J_a(I_a)$ in table II of Ref. [7]. Note that Ref. [7] does *not* work with zero-based indexing, which is used throughout this manuscript and the accompanying code.

Now, to incorporate the contribution of the first subsystem P_0, we overwrite its original lookup table – which maps $|\vec{\sigma}^{(0)}\rangle$ to a unique index $\text{index}_0(\vec{\sigma}^{(0)})$ – simply by its total contribution:

$$c_0 = \text{offset}_0\,(n_0, \lambda_0) + \text{stride}_0\,(n_0, \lambda_0)\,\text{index}_0(\vec{\sigma}^{(0)})\,. \tag{24}$$

The offset and stride are given by Eq. (15) and Eq. (20) :

$$\text{offset}_0\,(n_0, \lambda_0) = \sum_{k_0 = n_0^{\text{low}}}^{n_0 - 1} D_Q(L_0, k_0) D_Q(L_1, n - k_0) \tag{25}$$

$$\text{stride}_0\,(n_0, \lambda_0) = D_Q(L_1, n - n_0)\,. \tag{26}$$

The newly overwritten table corresponds to part B with $J_b(I_b)$ in the table II from Ref. [7].

While the trick to store the coefficients directly into the lookup table works for $N = 2$ due to the global constraint, the scheme is not possible for $N > 2$, and we have to account for this by tracking the particle number using λ_i.

3.3.2 L subsystems ($N = L$)

The opposite limit, evaluating $N = L$ subsystems of size one, can be done "on-the-fly" as it does not require the use of lookup tables. Since each system is of size one, it can have at most Q different states $|q_i\rangle$ with $q_i = 0, \dots, Q-1$. Therefore, there is only one state in each particle number sector in P_i inducing $\text{index}_i(\vec{\sigma}^{(i)}) = 0$ Then, the contribution of the i-th subsystem simplifies to:

$$c_i = \text{offset}_i(q_i, \lambda_i) = \sum_{k_i = n_i^{\text{low}}}^{q_i - 1} D_Q(\Gamma_i, n - \lambda_i - k_i)\,. \tag{27}$$

We have outlined the algorithm for $N = L$ in Algo. 1 and refer it as the "on-the-fly" implementation throughout the rest of the manuscript.

In the binary case ($Q = 2$), the formula to compute the index was already derived in Ref. [51,52]. Ref. [53] explicitly applied this scheme to enumerate product states in physical systems for arbitrary Q.

Algorithm 1: On-the-fly

```
**Data:** $|\vec{\sigma}\rangle = |q_0; \dots; q_{L-1}\rangle$
index, $\lambda = 0$                                  /* initializing variables */
$\Gamma = L - 1$
**for** $0 \le i < L - 1$ **do**
    **for** $0 \le k < q_i$ **do**
        **if** $n - \lambda - 1 - k \le (Q-1)\Gamma$ **then**
            index = index + $D_Q(\Gamma, n - \lambda)$
        **end**
        $\lambda = \lambda + 1$
    **end**
    $\Gamma = \Gamma - 1$
**end**
**return** index;
```

3.4 Enumerative encoding

The presented enumeration of basis states is an old problem in computer science and combinatorics [61]. In particular, Cover presented a generic ansatz in 1973 to compute the lexicographic one-to-one mapping and its inverse [52]. The idea behind his approach reflects the divide-and-conquer ansatz used in the derivation of our multidimensional search algorithm. In fact, we can use his formulation to derive our algorithm.

To formulate the problem in a computer science language, let $\vec{x} = (x_0, \dots, x_{N-1})$ be a *word* of length N and $x_i \in \{0, \dots, Q-1\}$ the *letters* from an alphabet of size Q. Then, the lexicographic order, $\vec{x} < \vec{y}$, is defined by $x_i < y_i$ where i is the smallest index with $x_i \neq y_i$.

Given any *arbitrary* subset $\mathcal{S}$ of all possible words of length N, we can use Cover's formula given in proposition 2 in Ref. [52] to find the lexicographic one-to-one mapping:

$$\mathcal{S} \to \{0, \dots, |S| - 1\}. \tag{28}$$

There, he defines the number of elements in $\mathcal{S}$ for which the first k letters are $(x_0, \dots, x_k)$ by $n_{\mathcal{S}}(x_0, \dots, x_k)$. The general formula that provides the desired mapping for $\vec{x}$ is:

$$\text{index}(\vec{x}) = \sum_{k=0}^{N-1} \sum_{l=0}^{x_k - 1} n_{\mathcal{S}}(x_0, \dots, x_{k-1}, l). \tag{29}$$

To demonstrate the generality of this ansatz, we have chosen a generic – not number conserving – set:

$$\mathcal{S} = \{(0,2,0), (0,2,1), (1,0,1), (2,0,0), (2,2,0), (2,2,1)\}.$$

The set is already lexicographically ordered, and we can illustrate the counting of $n_{\mathcal{S}}$. For example, the number of elements starting with (1) is $n_{\mathcal{S}}(1) = 1$ and with (0,2) is $n_{\mathcal{S}}(0,2) = 2$. Following the Eq. (29) , we derive the index of the last element $\vec{x} = (2,2,1)$ which is 5:

$$\text{index}(\vec{x}) = \underbrace{n_{\mathcal{S}}(0) + n_{\mathcal{S}}(1)}_{k=0} + \underbrace{n_{\mathcal{S}}(2,0) + n_{\mathcal{S}}(2,1)}_{k=1} + \underbrace{n_{\mathcal{S}}(2,2,0)}_{k=2} = 2 + 1 + 1 + 0 + 1 = 5.$$

Similarly to our multidimensional search algorithm, the first contribution, $k = 0$, takes care of all elements in $\mathcal{S}$ that have a smaller letter than x_0. This refers to the first contribution c_A in Fig. 1 that identifies the correct plane. The second part, $k = 1$, refers to c_B and jumps to the correct column. Lastly, $k = 2$ takes care of the last part and refers to the contribution c_C.

To relate this ansatz to our number constraint, we first use Eq. (29) to derive the $N = L$ limit with arbitrary Q. The contribution of the k-th subsystem is

$$c_k = \sum_{l_k=0}^{x_k-1} n_{\mathcal{S}}(x_0, \dots, x_{k-1}, l_k) . \tag{30}$$

The number of possible configurations in $\mathcal{S}$ that begin with $(x_0, \dots, x_{k-1})$ and fulfill the particle number constraint $n = \sum_k x_k$ are

$$n_{\mathcal{S}}(x_0, \dots, x_{k-1}, l_k) = D_Q(L-1-k, n-\lambda_k - l_k) .$$

$L-1-k$ is the length of the complement defined earlier by Γ_k. λ_k is the number of particles contained up to subsystem P_k: $\lambda_k = \sum_{s=0}^{k-1} x_s$. As we discussed in the preceding section, there might be a constraint on l_k restricting the sum in Eq. (30) to $l_k \in \{n_k^{\text{low}}, \dots, x_k - 1\}$. This can be extracted from the definition of $n_{\mathcal{S}}(\dots)$ which is simply zero if $l_k < n_k^{\text{low}}$. We have derived the same contribution for $N = L$ given in Eq. (27) using the general formalism from Cover.

Similarly, we can derive the offset and stride for the generic case. For simplicity, we choose an equal partitioning where all subsystem sizes are identical. In this case, the alphabet is growing exponentially with system size and each subsystem can have $M = 2^{L/N}$ states. Therefore, each $x_i = 0, \dots, M-1$ can take exponentially many values. To define a lexicographical order, we first have to impose a canonical ordering within each subsystem. Following the previous section, all M states are ordered by particle their number (lower number first), and we use a lookup table, *cf.* Table 1, to impose the order within each particle number sector. Each letter x_i refers to a substate on P_k and has an associated particle number $n(x_i) = 0, \dots, L/N(Q-1)$. The subset $\mathcal{S}$ is defined by the global particle number constraint n, and we use Eq. (29) to derive the index of $\vec{x} \in \mathcal{S}$. The contribution of the k-th subsystem is:

$$c_k = \sum_{l_k=0}^{x_k-1} n_{\mathcal{S}}(x_1, \dots, x_{k-1}, l_k) . \tag{31}$$

Note that the sum runs over exponentially many letters. To avoid adding this exponential overhead, we simply group letter $l_k \in \{0, \dots, x_k - 1\}$ into particle number sectors m_k on P_k:

$$\sum_{l_k=0}^{x_k-1} \to \sum_{m_k=0}^{n(x_k)} \sum_{l_k=0}^{x_k-1} \delta_{m_k, n(l_k)} .$$

$n(l_k)$ is the number of particles of the l_k-th state on P_k. For a given particle number $m_k < n(x_k)$, the number of states contained in the sum are $D_Q(L/N, m_k)$. Crucially, note that the number of words in $\mathcal{S}$ starting with $(x_0, \dots, x_{k-1}, l_k)$ only depends on number of particles contained in subsystem P_0 to P_k: $\lambda_k + n(l_k)$. Therefore, grouping the states according to their particle number greatly simplifies the equation as we can replace $n_{\mathcal{S}}(x_1, \dots, x_{k-1}, l_k)$ by $n_{\mathcal{S}}(\lambda_k, m_k)$:

$$c_k = \sum_{m_k=0}^{n(x_k)-1} D_Q(L/N, m_k) n_{\mathcal{S}}(\lambda_k, m_k) + \text{index}_k(\vec{\sigma}^{(k)}) n_{\mathcal{S}}(\lambda_k, n(x_k)) .$$

Here, $|\vec{\sigma}^{(k)}\rangle$ refers to the state associated with the letter x_k and $\text{index}_k(\vec{\sigma}^{(k)})$ is the index in the particle number sector $n_k = n(x_k)$. This form makes the origin of the offset and stride clear. To finally determine $n_{\mathcal{S}}(\lambda_k, m_k)$, we can use the same argument as in the previous section. Given the total constraint n, we already have $n - \lambda_k - m_k$ particles distributed on subsystems P_0 to P_k. Therefore, the number of possible configurations in $\mathcal{S}$ starting with any string $(x_0, \ldots, x_k)$ that contains $\lambda_k + m_k$ particles is $n_{\mathcal{S}}(\lambda_k, m_k) = D_Q(\Gamma_k, n - \lambda_k - m_k)$ where Γ_k is the length of the complement. Note that Cover's formula implicitly includes the lower bound on the particles on P_k as $n_{\mathcal{S}}(\lambda_k, m_k) = 0$ of $m_k < n_k^{\text{low}}$. Hence, we have derived our expression for c_k from Eq. (13) with the same offsets Eq. (19) and strides Eq. (23) .

4 DanceQ

DanceQ [45] is a high-performance library designed for a wide range of exact diagonalization techniques, serving as a frontend to state-of-the-art numerical libraries like Intel's `Math Kernel Library` or `Petsc` [47,48] and `Slepc` [49,50]. It achieves scalability and efficiency by leveraging the divide-and-conquer algorithm outlined before.

This section provides an overview of DanceQ's core modules, usage, and performance benchmarks. Sec. 4.1 begins by introducing the general layout and its usage. Sec. 4.2 introduces different implementations of the lookup tables and their performance is evaluated in Sec. 4.3. Finally, in Sec. 4.4, we explore DanceQ's performance in executing MPI-based matrix-free matrix-vector multiplication – a key task of the library. For further details, please refer to the full documentation [46].

4.1 Core Modules and Usage

DanceQ's architecture is built on a hierarchy of interconnected core modules. These include the State, Basis, and Operator classes, each playing a pivotal role in enabling high-performance computations.

State class The State class forms the inner core of DanceQ and uses a primitive bit-level structure for efficient storage and manipulation of product states. It provides the full functionality needed by more abstract modules such as the Basis and Operator class. To allow for maximal efficiency, the integer length representing a product state must be specified at compile time. This is done by defining the maximum number of sites potentially used via **MaxSites** and the local Hilbert Space dimension **Q**. When executed, smaller system sizes can be used without any concern.

```
#include "State.h"

/* Maximal system size */
#define MaxSites 128

/* Hilbert space dimension */
#define Q 2

/* Definition of the State class */
using State = danceq::internal::State<MaxSites,Q>;
```

Basis Class The BasisU1 class, referred to as Basis class, builds upon the State class and implements our number-conserving algorithm. It manages the lookup tables through a Container class, which comes in three implementations described in the next subsection: (i) memory-aligned list (default, ContainerTable), (ii) lexicographical order (ContainerDict), and (iii) combinatorial on-the-fly approach (ContainerFly). Users can specify the number of sites and particles at runtime, provided the number of sites does not exceed **MaxSites** and offers sufficient space to host the requested particle number.

```
#include "BasisU1.h"

/* Definition of the Container class */
using Container = danceq::internal::ContainerTable<State>;

/* Definition of the Basis class */
using Basis = danceq::internal::BasisU1<Container>;
```

Operator Class The Operator Class serves as the interface for the user. It can be constructed from the Basis class and supports the input of generic operator strings. Besides its number-conserving implementation, it allows for not number-considering systems when built from the State class. It further supports different float types by using **ScalarType**.

```
#include "Operator.h"

/* Scalar type for computation */
using ScalarType = std::complex<double>

/* Definition of the Hamiltonian class from the BasisU1 class
   */
using Hamiltonian_U1 = danceq::internal::Operator<Basis,
   ScalarType,danceq::Hamiltonian>;

/* Definition of the Hamiltonian class from the State class */
using Hamiltonian_NoU1 = danceq::internal::Operator<State,
   ScalarType,danceq::Hamiltonian>;
```

This class handles Hamiltonian operators acting on the Hilbert space, as well as Lindbladians operating on the space of density matrices. This flexibility applies to both number-conserving and non-number-conserving systems.

```
/* Definition of the Lindbladian class from the BasisU1 class
   */
using Lindbladian_U1 = danceq::internal::Operator<Basis,
   ScalarType,danceq::Lindbladian>;

/* Definition of the Lindbladian class from the State class */
using Lindbladian_NoU1 = danceq::internal::Operator<State,
   ScalarType,danceq::Lindbladian>;
```

The interface is inspired by the ITensor library [62, 63], which provides a straightforward input via strings. It comes with pre-implemented local operators like S^x, S^y, and S^z, but also allows for the definition of custom local operators. Below you can find an example for a Lindbladian operator with a dephasing term acting on site zero.

```
/* System size and particle number */
uint64_t N = 10;
uint64_t n = 5;

/* Lindbladian with number conservation */
Linbladian_U1 L(N,n);

/* Heisenberg model with OBC */
for(uint64_t i = 0; i < N-1; i++){
    L.add_operator(1., {i,(i+1)}, {"Sx","Sx"});
    L.add_operator(1., {i,(i+1)}, {"Sy","Sy"});
    L.add_operator(1., {i,(i+1)}, {"Sz","Sz"});
}

/* Jump operator acting on the first site */
L.add_jump_operator(1., {0}, {"Sz"});
```

The Operator class is a versatile tool, offering multiple formats for retrieving the matrix. In addition to supporting third-party matrix formats, it provides a custom-built sparse matrix and shell matrix. The latter one is used for matrix-free applications.

```
/* Dense matrix using std::vector<std::vector<std::complex<
   double>>> */
auto L_dense = L.create_DenseMatrix();

/* Dense matrix with Eigen using the type Operator::
   EigenMatrixType */
auto L_eigen = L.create_EigenDense();

/* Sparse matrix using the SparseMatrix class */
auto L_sparse = L.create_SparseMatrix();

/* Shell matrix using the ShellMatrix class for matrix-free
   multiplication */
auto L_shell = L.create_ShellMatrix();

/* Sparse matrix with Petsc MatType MATMPIAIJ */
auto L_sparse_petsc = L.create_PetscSparseMatrix();

/* Shell matrix with Petsc using the ShellMatrix class */
auto L_shell_petsc = L.create_PetscShellMatrix();
```

Internally, the Operator class uses a *sparse tensor storage* to act on any product state. Details about the idea and the implementation can be found in Appendix C.

Setup The core modules are bundled in the header file DanceQ.h. By defining **MaxSites** and **Q** (the maximum number of sites and the local Hilbert space dimension) before including the file, all classes become predefined and ready for use. A full example using the header file is given in Appendix D.

We strongly recommend using CMake to compile the code. Once the necessary paths are set correctly (a simple configuration script is included), building and utilizing DanceQ is straightforward. To fully leverage advanced C++ features, such as large-scale sparse MPI operators provided by `Petsc`, additional dependencies are required. We provide preconfigured Docker containers with all necessary software installed for various platforms to simplify the installation process. This allows for a straightforward setup.

The repository further includes several physical examples and test cases, exploring Lindbladian and Hamiltonian systems likewise. These examples demonstrate how to use the different frontends and features provided by DanceQ. Once the paths are set correctly—either by editing the configuration file or using Docker—the examples can be compiled and executed easily. A detailed list of examples, along with instructions on how to run them, can be found in the documentation [46].

4.2 Lookup tables

An efficient implementation of our multidimensional search algorithm uses two kinds of lookup tables. One is used to store the offsets and strides that are computed with Eq. (19) and Eq. (23)

. Both the offsets and strides depend on n_i and λ_i that can not be greater than $n \leq (Q-1)L$. Therefore, the memory required to store all possible coefficients for all N subsystems is smaller than Nn^2 and fits easily on any hardware.

However, the size of the other type of lookup table scales exponentially with the subsystem size, and reducing its volume is the motivation behind our work by introducing more subsystems. To recall, each subsystem has a lookup table that defines a canonical order within P_i, ignoring the rest of the system: For each subsystem particle number sector n_i, the table provides a *one-to-one* mapping between subsystem states $|\vec{\sigma}^{(i)}\rangle$ and an index, $\text{index}_i(\vec{\sigma}^{(i)})$, from zero to $D_Q(L_i, n_i) - 1$. Note that the system P_i has different particle number sectors where each has its own zero-based labeling. An example is shown Table 1. The index, together with the offset and stride, defines the subsystem contribution c_i.

The size of the lookup table $\text{index}_i(\vec{\sigma}^{(i)})$ scales exponentially with the subsystem size: Q^{L_i}. While the overhead coming from this table is manageable and does not hamper performance for $L_i \sim 10$, it quickly becomes a bottleneck for matrix-free applications in large eigenvalue problems. Therefore, in order to break the exponential increase, the system is split into multiple parts, keeping the individual subsystem sizes small. Splitting the system into $N = 2$ parts, as proposed by Ref. [7], helps to delay the problem, but it is an unsatisfying approach for $L \gtrsim 30$. In these cases, partitioning the system in more than two subsystems is required to reduce the memory overhead.

Implementation

We have implemented and tested three approaches to encode the lookup table $\text{index}_i(\vec{\sigma}^{(i)})$:

(i) memory-aligned list

(ii) lexicographical order in a tree-based associative map

(iii) combinatorial *on-the-fly*

The first option uses memory-aligned indices that are accessed using the integer representation of the state $|\vec{\sigma}^{(i)}\rangle$ similar to Ref. [7]. For example, we require two bits to encode a single state $|q\rangle$ for $q = 0, \ldots, 3$ with $Q = 4$. We denote the number of bits necessary to store a single state by $\texttt{Nbits}_Q = \texttt{ceil}(\log(Q)/\log(2))$. The state $|3;2;1;0\rangle$ with $L = 4$ spans over eight bits: (11100100) where two consecutive bits refer to a single qudit state. The bit string encodes the integer 228 and we, therefore, store the index of the state $|3;2;1;0\rangle$ at the 228-th position.

The second implementation (ii) might be advantageous in the limit of small filling fractions $n \ll L$. A table encoding a system of size L using the (i) requires $2^{\texttt{Nbits}_Q \cdot L}$ entries. However, in the limit of small fillings, most entries will never be used. By using a lexicographical order that only includes the valid states, the subsystem length can be chosen significantly bigger than in the first case.

Lastly, we can simply exploit Algo. 1 to compute the indices on-the-fly without actually storing the subsystem states. We actually use the algorithm to assign the unique indices to the subsystem states when tables in form (i) and (ii) are constructed. Again, the precise order is arbitrary as long as the mapping is one-to-one.

4.3 Performance

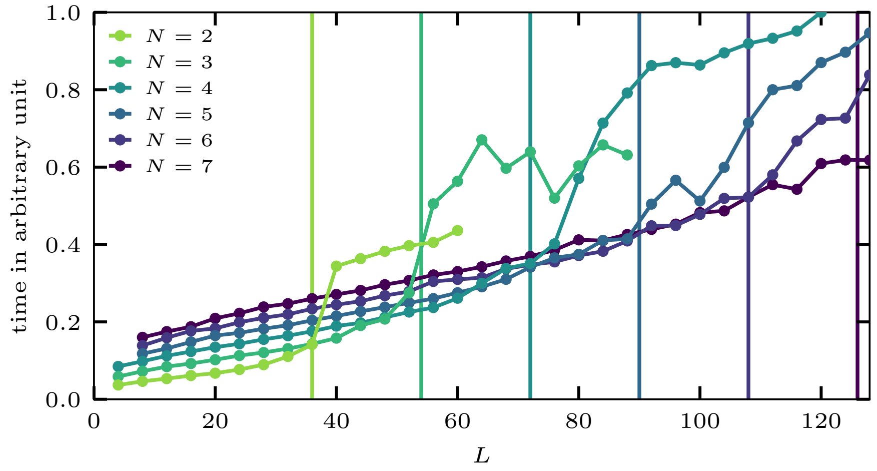


Figure 4: Runtime in arbitrary units to enumerate randomly generated trial states for a system of size L with $Q = 2$. N refers to the number of subsystems that are distributed "most" equally. The vertical lines mark the system size where the memory of the lookup table with size $2^{\texttt{ceil}(L/N)}$ exceeds the cache, here 4MB, of the processor. All computations were done using two unsigned integers with 64 bits each for state representation.

For a given length and filling fraction, the performance of the algorithm depends on the number of subsystems and their partitioning. To find the optimal choice, we randomly generate a fixed number of trial states and benchmark the time it takes to retrieve their basis index following the general recipe implemented in [46]. The number of states is of order 10^6. We used an *Intel i7-7500U* (2.70 GHz) processor with a cache size of 4 MB for these benchmarks.

As a first observation, we find that the performance drops significantly when the memory of the lookup tables exceeds the L3 cache of the processor. This is demonstrated in Fig. 4, which shows the runtime versus system size for different N. The vertical lines mark the point where the table exceeds the cache size. Hence, for optimal performance, the required memory should not exceed this limit.

Given the number of subsystems N, the partitioning with the lowest memory usage is the one that divides the whole system into the "most" equal parts. At most two different subsystem sizes L_i are present: $\texttt{ceil}(L/N)$ and $\texttt{floor}(L/N)$. This comes with another advantage, as we can use the same lookup tables for all subsystems of equal length, reducing the memory consumption further. Hence, we only consider the most equal partitioning of the system for the rest of the manuscript.

We find that the first container option (memory-aligned list) is in almost all cases the best choice. This is also true for dilute systems containing only a few particles. Fig. 5 displays the optimal number of subsystems for the first two lookup table implementations using the uniform partition. The upper panel refers to the memory-aligned list (i), and the lower panel refers to the lexicographical order (ii). By an optimal number of subsystems N_Q^{opt}, we mean that an equally sized partition with $N = N_Q^{\text{opt}}$ has the lowest runtime for our randomly generated test setup. We have evaluated the optimal number of subsystems in both cases for fixed filling fraction $f = \frac{n}{(Q-1)L}$ and length. In both cases, we see a clear trend that larger systems and larger filling fractions require more subsystems for an ideal performance. We note that the figure shows some deviations from this trend for larger system sizes for $Q > 2$.

To understand the scaling of the N_Q^{opt} with Q, we look into the most prominent case at half-filling using the memory-aligned list. Fig. 6 displays the N_Q^{opt} versus $L \cdot \texttt{Nbits}_Q$. To recall,

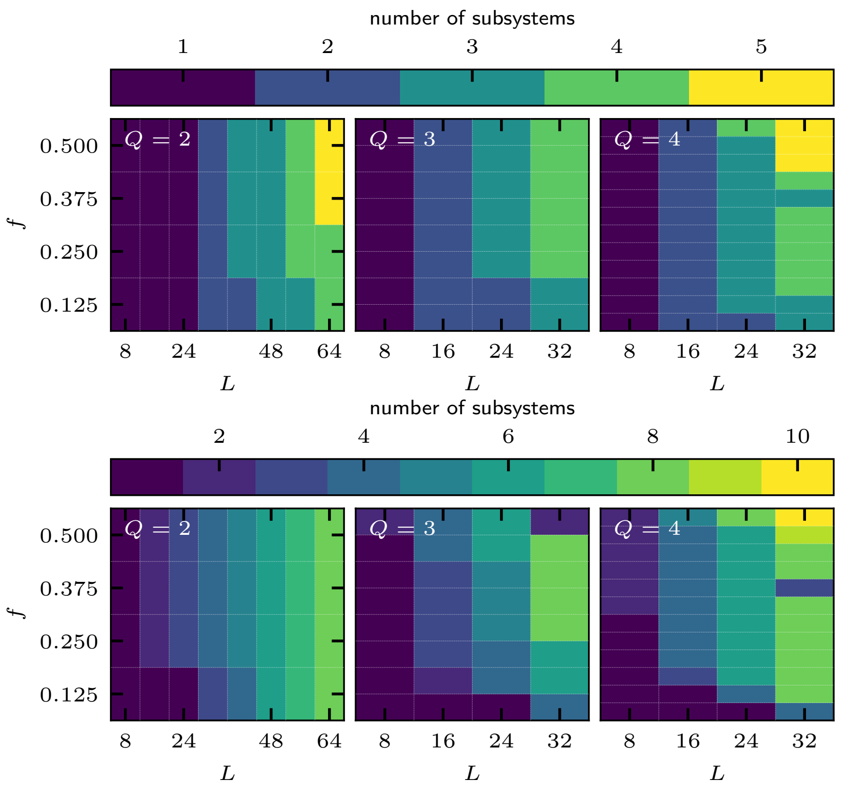


Figure 5: Optimal number of subsystems for two different implementations of the lookup tables using the memory-aligned list (left) and a lexicographical order (right). To determine the optimal N_Q^{opt}, we have chosen the most equal partition and measured the time it takes to retrieve indices of randomly generated trial states. The optimal N_Q^{opt} has the lowest runtime. L refers to the total system size, and Q to the local Hilbert space dimension. The filling fraction is defined by the number of particles in the total system divided by the maximal number of particles possible: $f = \frac{n}{(Q-1)L}$. We have not shown a computation for the on-the-fly approach (iii), as the optimal number of subsystems is simply $N = L$ for all cases. We further find that option (i) is superior for all filling fractions considered here. All computations were done using a single unsigned integer with 64 bits for state representation.

$\mathtt{Nbits}_Q = \mathtt{ceil}(\log(Q)/\log(2))$ is the number of bits required to encode a single state of dimension Q. We find that the optimal number of subsystems scales linearly with $L \cdot \mathtt{Nbits}_Q$:

$$N_Q^{\text{opt}} = \mathtt{ceil}\left(m_Q(L \cdot \mathtt{Nbits}_Q) + b_Q\right) . \tag{32}$$

We find good agreement for different values of Q and $m_Q \sim 0.05$. This can be understood as an optimal subsystem length L/N_Q^{opt} such that the table can be stored in the cache:

$$2^{\mathtt{Nbits}_Q \frac{L}{N_Q^{\text{opt}}}} \approx 2^{1/m_Q} \text{ for } b \ll m_Q(L \cdot \mathtt{Nbits}_Q) . \tag{33}$$

To summarize, we recommend using the most equal partition such that at most two tables have to be stored. Eq. (32) can be used to determine the optimal number of subsystems. However, in practice, the length should be chosen such that the lookup table footprint is smaller since other data needs to be stored in the L3 cache as well.

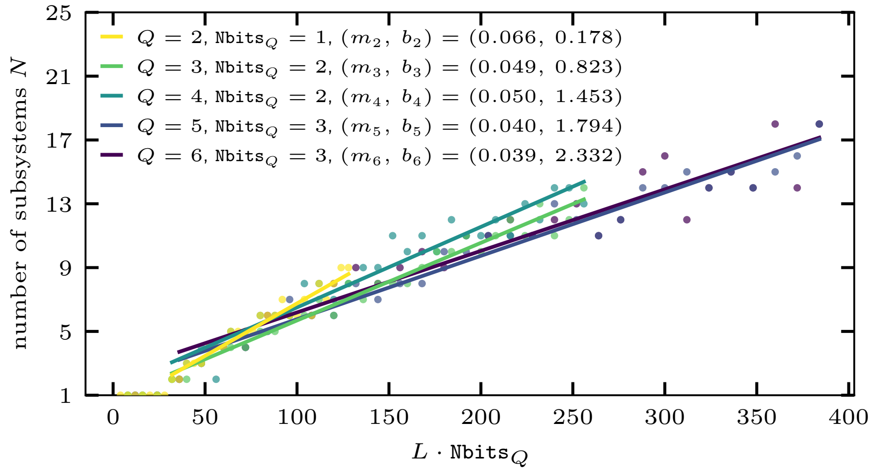


Figure 6: Optimal number of subsystems for different Q at half filling ($n = L(Q-1)/2$) versus $L\mathtt{Nbits}_Q$. For each system size, we use our test setup with randomly generated trial states and identify the optimal system size plotted on the y-axis. We find a linear scaling and fit $N_Q^{\text{opt}} = m_Q(L \cdot \mathtt{Nbits}_Q + b_Q)$ to extract the optimal scaling. $\mathtt{Nbits}_Q = \mathtt{ceil}(\log(Q)/\log(2))$ refers to the number of bits required to encode a single site with local Hilbert space dimension Q.

4.4 Matrix-free multiplication

Many algorithms in computational quantum many-body physics rely solely on matrix-vector multiplications to build, for example, a Krylov subspace which can used to perform real-time evolution, to calculate equilibrium properties via canonical typicality, or to compute ground states and excitations (e.g., by deflation techniques [58]), or other eigenvectors using spectral transformations [64–66]. Krylov space methods are particularly powerful and frequently applied to many physical problems due to the sparseness of the Hamiltonian as it reduces the complexity from a *cubic* for a full diagonalization to an often *linear* scaling with the Hilbert space dimension (which itself remains of course exponential in L).

The bottleneck for exact methods is usually the memory requirement to store the sparse Hamiltonian matrix, which scales with the Hilbert space dimension times the number of off-diagonal matrix elements per row (which is typical of order L in the case of nearest-neighbor interactions) for sparse matrix-vector multiplication. Therefore, to reduce the memory further, state-of-the-art computations [31–33,38,67–69] do not store this matrix and instead compute the action of its elements on the input vector on the fly in a massively parallel way. However, to ensure fast computations, each worker process must know the basis states and their associated indices. This is the main contribution of our algorithm, as it allows a memory-efficient way to perform this type of bookkeeping.

To understand the scaling of the subsystem size within the full matrix-free multiplication [46], we monitored the time it take to perform one such matrix-vector operation. While the performance depended crucially on N and the available cache in the last subsection, where we only focused on the lookup, we do not observe this behavior in this case. In fact, we find that the time depends only slightly on the number of subsystems, and the best performance was achieved by using a single "subsystem" of size L ($N = 1$) – if it fits in the RAM. The dependence of the runtime for a single matrix-free matrix-multiplication is shown in Fig. 7. We interpret this finding to indicate strong cache interference between data required for the actual multiplications of matrix elements on the vector and data for lookup tables, which means that the

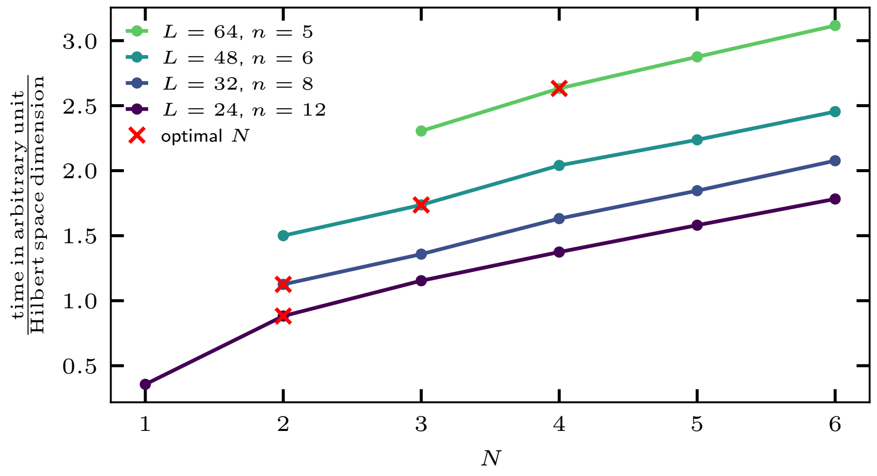


Figure 7: Runtime for a single matrix-free matrix-vector multiplication (isotropic Heisenberg chain) divided by the dimensionality of the problem versus the number of subsystems. Computations are carried out with the two MPI threads. The red crosses mark the optimal number of subsystems, as stated in Eq. (34) .

lookup and the retrieval of the indices play only a secondary role during the full matrix-vector multiplication and other operations that take place have to be considered. For example, the output wave function (which usually fills up the whole RAM) is constantly edited, and states have to be incremented and manipulated throughout the process. Therefore, to obtain the best performance, we recommend choosing N small but without consuming any meaningful memory. In other words, the memory footprint of each worker process should be the guiding principle when choosing N since the computing time in real-world applications only depends weakly on N.

The memory-core ratio is of the order of 4 GiB on modern platforms, and we will use a 4 GB limit per core as an example for the following discussion. Since memory is the constraining part for Krylov space techniques we do not want to block any significant amount of it. However, storing the lookup table for a single subsystem $N = 1$ blocks the available memory, which should be used by the wave function and is quickly exhausted ($L = 27$ for $Q = 2$). In Fig. 8, we show the memory required by the lookup table. In MPI-based programs in this scenario, each worker process is in charge of 4 GiB and has to store its own table. Therefore, it is not possible that the lookup table takes more than 4 GiB. This is indicated by the red-shaded area where more subsystems are required to reduce the memory consumption. Blue (yellow) color refers to large (exponentially small) fractions of the 4 GiB limits used by the table. The default setting of our code and our recommendation is 512 kiB, which refers to a subsystem length of $L_i = 16$ for $Q = 2$ [46]:

$$N_Q^{\text{opt}} = \texttt{ceil}\left(\frac{\texttt{Nbits}_Q \cdot L}{16}\right). \tag{34}$$

This choice is also in agreement with Eq. (32) ($b_Q = 0$) and the linear scaling from Fig. 6 with $m_2 \sim 1/16$. Note that, at most, two lookup tables are required for the most equal partition. Red crosses in Fig. 7 mark the optimal number of subsystems for the respective system sizes tested.

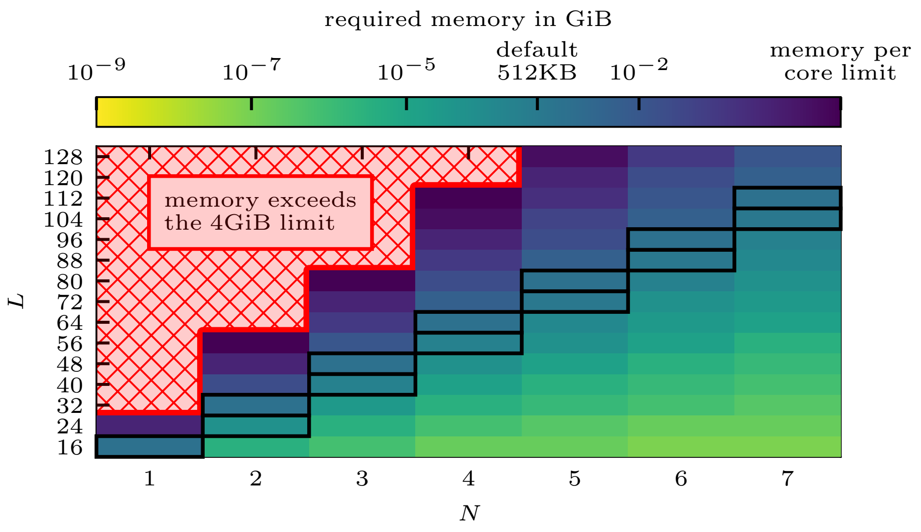


Figure 8: The figure displays the fraction of memory used to store the largest lookup table to the available memory per processor, which we set to 4 GiB here for different system sizes L and number of equally sized subsystems N ($Q = 2$). The blue (yellow) color indicates that a large (exponentially small) portion is used by the lookup table. The red-shaded area indicates system sizes that require more subsystems in order to fit the table within the memory of the processor. The bottleneck of exact diagonalization is usually memory, and the fraction of memory associated with the table should be chosen rather small. For each system size L, we have marked the optimal number of subsystems with a black box where the memory required by the lookup table does not exceed our default setting of 512 kiB [46]. Note that the memory consumption of the table is independent of the particle sector.

5 Conclusion

We presented an solution to efficiently deal with number-conserving systems in very large-scale, massively parallel calculations where the available memory per core limits space available for lookup tables to map basis states to their index. While an on-the-fly algorithm Algo. 1 exists as the extreme limit with negligible memory requirements, it is the slowest solution. The traditional approach [7] using two subsystems is much faster but requires too much memory for system sizes coming within reach on exascale machines. Our general divide-and-conquer algorithm interpolates between these two limits and provides an optimal balance between computational cost and available memory to overcome these limitations.

We have implemented this algorithm in a general, state-of-the-art, and header-only C++20 library available at Ref. [46]. The data used in the figures is publicly accessible at Ref. [70]. The code is user-friendly and allows the exploitation of the full power of large-scale computing facilities, making ground-state searches and time evolution for large systems possible. By combining several nodes via MPI, our implementation is capable of computing ground states for systems containing 46 spins ($Q = 2$) within the zero-magnetization sector. The required memory to store the necessary two wave functions is about 120 TiB which can be provided by ~ 256 nodes with 512GiB each. Similar to SPINPACK [34] and XDiag [35,38], the forthcoming version of DanceQ [45] will support the use and automatic detection of spatial symmetries following the ansatz developed in Ref. [37]. This makes larger systems accessible as the complexity is typically reduced by the system size, requiring only ~ 6 nodes in the example above.

While the focus of this paper and the accompanying code is on quantum magnetism, it is applicable to other problems of many Fermions or Bosons with conserved total particle number.

The problem of efficiently enumerating states or sequences in lexicographical order extends beyond physics and is important in various areas of computer science [51,52].

We note in closing that our method is formulated for L identical qudits with Q states per site. At the expense of additional bookkeeping, it is straightforward to generalize our approach to different Q for each site, which is relevant for systems of mixed spin S or, for example, Bose-Fermi mixtures [55].

Acknowledgements

This work was financially supported by the Deutsche Forschungsgemeinschaft through the cluster of excellence ML4Q (EXC 2004, project-id 390534769). DJL acknowledges support from the QuantERA II Programme that has received funding from the European Union's Horizon 2020 research innovation programme (GA 101017733), and from the Deutsche Forschungsgemeinschaft through the project DQUANT (project-id 499347025). We further acknowledge support by the Deutsche Forschungsgemeinschaft through CRC 1639 NuMeriQS (project-id 511713970). RS acknowledges the AFOSR Grant No. FA9550-20-1-0235.

A Hilbert space dimension

We consider a tensor product Hilbert space of local Q-dimensional spaces, subject to the constraint that the sum of local excitations n is fixed.

For $Q = 2$, the Hilbert space dimension of a sector $n = 0, \ldots, L$ can be derived combinatorially and is well known to be determined by the binomial coefficient:

$$D_2(L,n) = \binom{L}{n}. \tag{A.1}$$

However, determining the dimension of each sector for larger local dimension Q is more involved. It is related to the probability of scoring a fixed sum in the throw of L dices with Q faces, *cf.* p. 284, problem 18 in Ref. [71]. In the context of Hilbert space dimensions, one of the early applications can be found in Refs. [54,57].

Here, we provide an elementary derivation of this closed-form equation. We approach this problem by defining an equal superposition of all possible computational states

$$|\Psi\rangle = \bigotimes_{i=1}^{L} \left(\sum_{j=0}^{Q-1} |j\rangle \right). \tag{A.2}$$

Now, to determine the dimension of a sector with a certain magnetization n, we need to identify all states exhibiting the correct magnetization. This problem is equivalent to determining the coefficient of x^n of the polynomial $f(x) = \left(1 + x + \cdots + x^{Q-1}\right)^L$:

$$D_Q(L,n) = \mathrm{coef}_{x^n}\left[\left(1 + x + \cdots + x^{Q-1}\right)^L\right]. \tag{A.3}$$

Here, we identified the state $|k\rangle_1$ with x^k. Each computational state exhibiting the correct magnetization contributes to the coefficient of x^n.

We evaluate the polynomial using the finite geometric sum

$$f(x) = \left(\sum_{i=0}^{Q-1} x^i \right)^L = \left(\frac{x^Q - 1}{x - 1} \right)^L = \frac{\left(x^Q - 1\right)^L}{(x-1)^L}. \tag{A.4}$$

Then, the denominator is expanded using its Taylor series around $x = 0$:

$$(x-1)^{-L} = (-1)^{-L} \sum_{k=0}^{\infty} \frac{1}{k!} \left[\prod_{s=0}^{k-1} (L+s) \right] x^k = (-1)^{-L} \sum_{k=0}^{\infty} \binom{L-1+k}{L-1} x^k . \tag{A.5}$$

and the nominator is evaluated using the binomial coefficients:

$$\left(x^Q - 1\right)^L = \sum_{k=0}^{L} \binom{L}{k} x^{Qk} (-1)^{L-k}. \tag{A.6}$$

To obtain the dimension of the sector, Eq. (A.5) and Eq. (A.6) are multiplied and we evaluate the coefficient of x^n:

$$D_Q(L,n) = \sum_{k=0}^{\lfloor n/Q \rfloor} (-1)^k \binom{L}{k} \binom{L-1+n-Qk}{L-1}. \tag{A.7}$$

$\lfloor\,\rfloor$ is the lower Gauss bracket.

B Pseudo code

This section presents the pseudo-code of the most important functions (i), (ii), and (iii) from Sec. 2.2. Our DanceQ library initiates the lookup tables that provide the index within a particle number sector and all necessary offsets and strides from Eq. (19) and Eq. (23) . The following functions are implemented by an underlying `State` class:

- `get_n`$(|\vec{\sigma}\rangle, k)$:
  Returns the number of particles in the subsystem P_k.

- `get_minimal_state`(l, n):
  Returns the state with index 0 for a system with l sites and n particles. It has to be consistent with the lookup tables.

- `is_maximal`$(|\vec{\sigma}\rangle, k)$:
  Returns *True* if the subsystem state on P_k is the last state for its particle number sector in P. It has to be consistent with the lookup tables.

- `increment_local`$(|\vec{\sigma}\rangle, k)$:
  Returns the next state within the same particle number sector of $|\vec{\sigma}^{(k)}\rangle$ on subsystem P_k according to the lookup table.

Note that all functions have to be consistent with the chosen lookup table. A possible implementation to derive lookup tables and the required functions is as follows: We iterate from the "right" side to the "left" side of the respective subsystem. If the local state at site i is not maximal ($\neq |Q-1\rangle$) and the number of excitations n_{prev} on previous sites is greater than one, we can increase the state at site i and set the previous sites to the right of i to its minimal state defined by $n_{\text{prev}} - 1$. This is obtained by setting the remaining excitations $n_{\text{prev}} - 1$ as much to the "right" as possible.

We further defined a "container class" that is in charge of the lookup table.

- `get_local_index`$(|\vec{\sigma}\rangle, k)$:
  Returns the subsystem index for subsystem P_k: $\text{index}_k(\vec{\sigma}^{(k)})$.

- get_local_state(index, k):
  Returns the subsystem $|\vec{\sigma}^{(k)}\rangle$ on subsystem P_k:
  $$\texttt{index} = \text{index}_k(\vec{\sigma}^{(k)}) = \texttt{get\_local\_index}(|\vec{\sigma}\rangle, k)\,.$$
  This is the reverse function of the previous one.

Algorithm 2: Function (i): get_index

```
Data: $|\vec{\sigma}\rangle$
index $= 0$                                              /* initializing variables */
$\lambda = 0$
for $0 \le k < N$ do
    $n_k = $ get_n($|\vec{\sigma}\rangle, k$)                    /* local particle number in $P_k$ */
    $i_k = $ get_local_index($|\vec{\sigma}\rangle, k$)          /* index from the lookup table */
    $c_k = \text{offset}_k(n_k, \lambda) + i_k \cdot \text{stride}_k(n_k, \lambda)$   /* contribution of $P_k$ as defined in Eq. (13) */
    index $=$ index $+ c_k$
    $\lambda = \lambda + n_k$
end
return index;
```

C Sparse tensor storage

A core module of the DanceQ library is the Operator class, which provides an easy interface to handle arbitrary tensor products defined on a system consisting of L sites with a local Hilbert space dimension Q. Besides handling and organizing any input, it allows for a highly optimized on-the-fly matrix-vector multiplication without storing the exponentially large matrix. For optimal performance, the Operator class employs a similar divide-and-conquer approach. This involves merging several local terms that act on the same sites, a strategy that enhances efficiency and reduces computational complexity. In particular, we identify subclusters of N_{tensor} sites of the system and merge all local operators fully supported in this subcluster into a single sparse matrix of size $Q^{N_{\text{tensor}}}$.

Consider for example a one-dimensional spin chain of length $L = 30$ (we use periodic boundary conditions where we identify site 30 refers site 0):

$$H = \sum_{i=0}^{29} \left(S_i^x S_{i+1}^x + S_i^y S_{i+1}^y + S_i^z S_{i+1}^z\right) + \sum_{i=0}^{29} S_i^z\,. \tag{C.1}$$

To apply the Hamiltonian to a product state, we have to execute all $4 \cdot 30$ local operators. In order to reduce this complexity that scales with L, we assign three *overlapping* subclusters of size $N_{\text{tensor}} = 11$:

$$\mathcal{C}_0 = \{0, \ldots, 10\},\, \mathcal{C}_1 = \{10, \ldots, 20\},\, \mathcal{C}_2 = \{0, 20, \ldots, 29\}\,.$$

Note that the subclusters need to overlap to encompass all terms. This allows us only to store three sparse matrices $\mathcal{S}_i$ of size 2^{11}, each containing all operators fully supported on the individual clusters C_i. For example, all terms that act solely on $\mathcal{C}_0$,

$$H_{\mathcal{C}_0} = \sum_{i=0}^{9} \left(S_i^x S_{i+1}^x + S_i^y S_{i+1}^y + S_i^z S_{i+1}^z\right) + \sum_{i=0}^{10} S_i^z \tag{C.2}$$

Algorithm 3: Function (ii): `increment`

```
Data: $|\vec{\sigma}\rangle$
$\lambda = 0$
$\Gamma = 0$
for $1 \leq j \leq N$ do
    $k = N - j$    /* iterate backwards through all subsystems starting with the last */
    $n_k = \texttt{get\_n}(|\vec{\sigma}\rangle, k)$
    if not $\texttt{is\_maximal}(|\vec{\sigma}\rangle, k)$ then    /* increase state while persevering the particle number $n_k$ */
        $|\vec{\gamma}^{(k)}\rangle = \texttt{increment\_local}(|\vec{\sigma}\rangle, k)$    /* increase the state $|\Psi\rangle$ locally on $P_k$ within the sector $n_k$ */
        $|\vec{\gamma}^{(k+1,\ldots,N-1)}\rangle = \texttt{get\_minimal\_state}(\Gamma, \lambda)$    /* minimal state on $P_{k+1}$ to $P_{N-1}$ with length $\Gamma$ and $\lambda$ particles */
        $|\vec{\gamma}\rangle = |\vec{\sigma}^{(0,\ldots,k-1)}\rangle \bigotimes |\vec{\gamma}^{(k)}\rangle \bigotimes |\vec{\gamma}^{(k+1,\ldots,N-1)}\rangle$
        return $|\vec{\gamma}\rangle$
    else if $\lambda > 0$ and $n_k < (Q-1)L_k$ then /* increase state particle number in $P_k$ */
        $|\vec{\gamma}^{(k)}\rangle = \texttt{get\_minimal\_state}(L_k, n_k + 1)$    /* get the minimal state on $P_k$ with length $L_k$ and $n_k + 1$ particles */
        $|\vec{\gamma}^{(k+1,\ldots,N-1)}\rangle = \texttt{get\_minimal\_state}(\Gamma, \lambda - 1)$    /* minimal state on $P_{k+1}$ to $P_{N-1}$ with length $\Gamma$ and $\lambda - 1$ particles */
        $|\vec{\gamma}\rangle = |\vec{\sigma}^{(0,\ldots,k-1)}\rangle \bigotimes |\vec{\gamma}^{(k)}\rangle \bigotimes |\vec{\gamma}^{(k+1,\ldots,N-1)}\rangle$
        return $|\vec{\gamma}\rangle$
    end
    $\Gamma = \Gamma + L_k$
    $\lambda = \lambda + n_k$
end
return $\texttt{get\_minimal\_state}(L, n)$    /* the input state is maximal; return the minimal state */
```

Algorithm 4: Function (iii): `get_state`

```
Data: index
$\lambda = 0$
$\Gamma = L$
for $0 \leq k < N - 1$ do
    Determine $n_k$ s.t. $\text{offset}_k(n_k, \lambda) \leq \texttt{index} < \text{offset}_k(n_k + 1, \lambda)$    /* determine the correct particle number on $P_k$ */
    $\texttt{index} = \texttt{index} - \text{offset}_k(n_k, \lambda)$
    $i_k = \texttt{index}/\text{stride}_k(n_k, \lambda_k)$    /* determine the local index in the particle number sector $n_k$ */
    $|\vec{\sigma}^{(k)}\rangle = \texttt{get\_local\_state}(i_k, k)$    /* reverse lookup table */
    $\texttt{index} = \texttt{index} - i_k \cdot \text{stride}_k(n_k, \lambda_k)$
    $\lambda = \lambda + n_k$
end
$|\vec{\sigma}^{(N-1)}\rangle = \texttt{get\_local\_state}(\texttt{index}, k)$    /* last subsystem */
$|\vec{\sigma}\rangle = \bigotimes_k |\vec{\sigma}^{(k)}\rangle$
return $|\vec{\sigma}\rangle$
```

are compressed into $\mathcal{S}_0$. Thus, applying the large tensor matrix reduces the complexity of iterating over $4 \cdot 30$ local operators to only three operators resulting in less state manipulations and computational overhead. Despite the enhanced dimension of the matrices $\mathcal{S}_i$ compared to the two-body terms in Eq. (C.1) , it is bounded by N_{tensor} and can be chosen such it easily fits in the cache of the processor. We have chosen the default such that the dimension does not exceed $Q^{N_{\text{tensor}}} = 2048$.

Now, given an input state ψ, we can extract the corresponding `columnindex` of the sparse matrix for a given cluster $\mathcal{C}_i$ by:

$$\texttt{columnindex} = \sum_{k=0}^{|\mathcal{C}_k|-1} Q^k \psi[k]. \tag{C.3}$$

In our implementation, this index points directly to the memory-aligned coefficients and elements of $\mathcal{S}_i$. To further enhance the computation, the class works with statemasks which are stored within the sparse matrix. Instead of storing the sparse matrix of size $Q^{N_{\text{tensor}}} \times Q^{N_{\text{tensor}}}$, we directly store each *column* of this matrix as a *sparse vector*.

While the above description refers to only nearest-neighbor operators acting on two sites, its generalization is straightforward and implemented in the class. Note that the choice of subclusters does not correspond the partition of our multidimensional search algorithm.

To apply a column of the cluster-local operator to an element of the input vector (with a corresponding basis state), we effectively iterate over all configurations on the complement of the cluster for each nonzero element of the sparse matrix to calculate the contributions to the result vector. The bookkeeping in the innermost loop is performed using cheap bitwise logical operations.

D Example code

This section presents a brief example demonstrating how to compute the ground state and first excited state of a spin chain in a tilted field, along with the magnetization measurement at the first site. The code utilizes MPI and our native Lanczos implementation. The code prints the energies E_k together with their expectation value $\langle \Psi_k | O | \Psi_k \rangle$. The Hamiltonian and observable are defined as follows:

$$H = \sum_{k=0}^{L-1} \frac{1}{2}\left(S_k^+ S_{k+1}^- + S_k^- S_{k+1}^+\right) + S_k^z S_{k+1}^z + k\, S_k^z \tag{D.1}$$

$$O = S_0^z . \tag{D.2}$$

```
#include <iostream>
#include <mpi.h>

/* Eigen is required to diagonalize the projection onto the
   Krylov subspace */
#include <Eigen/Dense>
#include <Eigen/Eigenvalues>

/* Maximal system size */
#define MaxSites 64

/* Hilbert space dimension */
#define Q 2
```

```
/* ScalarType */
#define ScalarType double

#include "DanceQ.h"
using namespace danceq;


int main(int argc, char *argv[]) {

    /* Rank number */
    int myrank = 0;

    /* Initializes MPI */
    MPI_Init(&argc, &argv);

    /* Number of MPI ranks */
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);

    /* Rank number */
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

    /* Number of particles */
    uint64_t n = 14;

    /* System size */
    uint64_t L = 28;

    /* number of states */
    uint64_t number_of_states = 4;

    /* Hamiltonian */
    Hamiltonian_U1 H(L,n);

    /* XXX-Heisenberg model with open boundary conditions and a
    tilted field */
    for(uint64_t i = 0; i < L-1; i++){
        H.add_operator(.5, {i,(i+1)%L}, {"S+","S-"});
        H.add_operator(.5, {i,(i+1)%L}, {"S-","S+"});
        H.add_operator(1., {i,(i+1)%L}, {"Sz","Sz"});
        H.add_operator(static_cast<double>(i), {i}, {"Sz"});
    }

    /* Information */
    H.info();

    /* Observable */
    Hamiltonian_U1 O(L,n);

    /* Operator measuring the magnetization of the first site
   */
    O.add_operator(1., {0}, {"Sz"});

    auto O_matrix = O.create_ShellMatrix();

    std::vector<Vector> states;
    auto data = lanczos(H, number_of_states, &states /* returns
    eigen states if this is not nullptr */);
    for(uint64_t s_for = 0UL; s_for < data.size(); s_for++){
        auto obs = O_matrix.get_expectation_value(states[s_for
   ]);
        if(myrank == 0){
            std::cout << "[main] - E_" << s_for << " = " <<
   data[s_for] << ", O_" << s_for << " = " << obs << std::endl;
        }
    }

    /* Finalizes MPI */
    MPI_Finalize();

    return 0;
}
```

References

[1] H. Bethe, *Zur Theorie der Metalle*, Zeitschrift für Physik **71**(3), 205 (1931), doi:10.1007/BF01341708.

[2] A. J. Kitaev, *Fault-tolerant quantum computation by anyons*, Annals of Physics **303**(1), 2 (2003), doi:https://doi.org/10.1016/S0003-4916(02)00018-0.

[3] A. J. Kitaev, *Anyons in an exactly solved model and beyond*, Annals of Physics **321**(1), 2 (2006), doi:https://doi.org/10.1016/j.aop.2005.10.005.

[4] A. Georges, G. Kotliar, W. Krauth and M. J. Rozenberg, *Dynamical mean-field theory of strongly correlated fermion systems and the limit of infinite dimensions*, Reviews of Modern Physics **68**(1), 13 (1996), doi:10.1103/RevModPhys.68.13.

[5] T. Giamarchi, *Quantum Physics in One Dimension*, Oxford University Press, ISBN 9780198525004, doi:10.1093/acprof:oso/9780198525004.001.0001 (2003).

[6] M. Hermele, M. P. A. Fisher and L. Balents, *Pyrochlore photons: The $U(1)$ spin liquid in a $S = \frac{1}{2}$ three-dimensional frustrated magnet*, Phys. Rev. B **69**, 064404 (2004), doi:10.1103/PhysRevB.69.064404.

[7] H. Q. Lin, *Exact diagonalization of quantum-spin models*, Phys. Rev. B **42**, 6561 (1990), doi:10.1103/PhysRevB.42.6561.

[8] A. W. Sandvik and J. Kurkijärvi, *Quantum Monte Carlo simulation method for spin systems*, Phys. Rev. B **43**, 5950 (1991), doi:10.1103/PhysRevB.43.5950.

[9] S. R. White, *Density matrix formulation for quantum renormalization groups*, Phys. Rev. Lett. **69**, 2863 (1992), doi:10.1103/PhysRevLett.69.2863.

[10] U. Schollwöck, *The density-matrix renormalization group in the age of matrix product states*, Annals of Physics **326**(1), 96 (2011), doi:10.1016/j.aop.2010.09.012.

[11] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Natl. Bur. Stand. B **45**, 255 (1950), doi:10.6028/jres.045.026.

[12] W. E. Arnoldi, *The principle of minimized iterations in the solution of the matrix eigenvalue problem*, Quarterly of Applied Mathematics **9**(1), 17 (1951).

[13] A. S. Householder, *Unitary triangularization of a nonsymmetric matrix*, J. ACM **5**(4), 339–342 (1958), doi:10.1145/320941.320947.

[14] G. H. Golub and H. A. van der Vorst, *Eigenvalue computation in the 20th century*, Journal of Computational and Applied Mathematics **123**(1), 35 (2000), doi:https://doi.org/10.1016/S0377-0427(00)00413-1.

[15] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller, *Equation of State Calculations by Fast Computing Machines*, The Journal of Chemical Physics **21**(6), 1087 (1953), doi:10.1063/1.1699114.

[16] E. Fermi, J. R. Pasta and S. M. Ulam, *Studies of nonlinear problems I*, Tech. rep., Los Alamos Report LA-1940, doi:10.2172/4376203 (1955).

[17] B. J. Alder and T. E. Wainwright, *Studies in Molecular Dynamics. I. General Method*, The Journal of Chemical Physics **31**(2), 459 (1959), doi:10.1063/1.1730376.

[18] E. N. Lorenz, *Deterministic nonperiodic flow*, Journal of Atmospheric Sciences **20**(2), 130 (1963), doi:10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2.

[19] W. K. Hastings, *Monte Carlo sampling methods using Markov chains and their applications*, Biometrika **57**(1), 97 (1970), doi:10.1093/biomet/57.1.97.

[20] H. O. Pritchard, F. H. Sumner and G. Gee, *The application of electronic digital computers to molecular orbital problems I. The calculation of bond lengths in aromatic hydrocarbons*, Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences **226**(1164), 128 (1954), doi:10.1098/rspa.1954.0243.

[21] R. Orbach, *Antiferromagnetic Magnon Dispersion Law and Bloch Wall Energies in Ferromagnets and Antiferromagnets*, Phys. Rev. **115**, 1181 (1959), doi:10.1103/PhysRev.115.1181.

[22] L. F. Mattheiss, *Antiferromagnetic linear chain*, Phys. Rev. **123**, 1209 (1961), doi:10.1103/PhysRev.123.1209.

[23] G. Dresselhaus, *Ferro- and Antiferromagnetism in a Cubic Cluster of Spins*, Phys. Rev. **126**, 1664 (1962), doi:10.1103/PhysRev.126.1664.

[24] J. C. Bonner and M. E. Fisher, *The entropy of an antiferromagnet in a magnetic field*, Proceedings of the Physical Society **80**(2), 508 (1962), doi:10.1088/0370-1328/80/2/318.

[25] J. C. Bonner and M. E. Fisher, *Linear magnetic chains with anisotropic coupling*, Phys. Rev. **135**, A640 (1964), doi:10.1103/PhysRev.135.A640.

[26] C. Kawabata, *Statistical Mechanics of the Finite Heisenberg Model. II*, Journal of the Physical Society of Japan **28**(4), 861 (1970), doi:10.1143/JPSJ.28.861.

[27] V. Mubayi, C. K. Majumdar and K. Krishan, *Distribution of Zeros of the Partition Function for the Finite Two-Dimensional Heisenberg Model*, Phys. Rev. B **8**, 3305 (1973), doi:10.1103/PhysRevB.8.3305.

[28] J. Oitmaa and D. D. Betts, *The ground state of two quantum models of magnetism*, Canadian Journal of Physics **56**(7), 897 (1978), doi:10.1139/p78-120.

[29] G. E. Moore, *Cramming more components onto integrated circuits*, Electronics Magazine **38** (1965).

[30] G. E. Moore, *Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff.*, IEEE Solid-State Circuits Society Newsletter **11**(3), 33 (2006), doi:10.1109/N-SSC.2006.4785860.

[31] A. Wietek, *Topological states of matter in frustrated quantum magnetism*, Universität Innsbruck, Ph.D. thesis (2017).

[32] A. M. Läuchli, J. Sudan and R. Moessner, *$S = \frac{1}{2}$ kagome Heisenberg antiferromagnet revisited*, Phys. Rev. B **100**, 155142 (2019), doi:10.1103/PhysRevB.100.155142.

[33] R. Schäfer, *Magnetic Frustration in Three Dimensions*, TU Dresden, Ph.D. thesis (2022).

[34] Joerg Schulenburg, *Spinpack*, https://www-e.uni-magdeburg.de/jschulen/spin/ (2016).

[35] Alexander Wietek, *XDiag*, https://github.com/awietek/xdiag (2024).

[36] A. Weiße, *Divide and conquer the Hilbert space of translation-symmetric spin systems*, Phys. Rev. E **87**, 043305 (2013), doi:10.1103/PhysRevE.87.043305.

[37] R. Schäfer, I. Hagymási, R. Moessner and D. J. Luitz, *Pyrochlore $S = \frac{1}{2}$ Heisenberg antiferromagnet at finite temperature*, Phys. Rev. B **102**, 054408 (2020), doi:10.1103/PhysRevB.102.054408.

[38] A. Wietek and A. M. Läuchli, *Sublattice coding algorithm and distributed memory parallelization for large-scale exact diagonalizations of quantum many-body systems*, Phys. Rev. E **98**, 033309 (2018), doi:10.1103/PhysRevE.98.033309.

[39] J. Hubbard and B. H. Flowers, *Electron correlations in narrow energy bands*, Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences **276**(1365), 238 (1963), doi:10.1098/rspa.1963.0204.

[40] H. Lin, J. Gubernatis, H. Gould and J. Tobochnik, *Exact Diagonalization Methods for Quantum Systems*, Computer in Physics **7**(4), 400 (1993), doi:10.1063/1.4823192.

[41] M. Sharma and M. Ahsan, *Organization of the Hilbert space for exact diagonalization of Hubbard model*, Computer Physics Communications **193**, 19 (2015), doi:https://doi.org/10.1016/j.cpc.2015.03.014.

[42] E. R. Gagliano, E. Dagotto, A. Moreo and F. C. Alcaraz, *Correlation functions of the antiferromagnetic Heisenberg model using a modified Lanczos method*, Phys. Rev. B **34**, 1677 (1986), doi:10.1103/PhysRevB.34.1677.

[43] J. M. Zhang and R. X. Dong, *Exact diagonalization: the Bose–Hubbard model as an example*, European Journal of Physics **31**(3), 591 (2010), doi:10.1088/0143-0807/31/3/016.

[44] M. Kawamura, K. Yoshimi, T. Misawa, Y. Yamaji, S. Todo and N. Kawashima, *Quantum lattice model solver $H\phi$*, Computer Physics Communications **217**, 180 (2017), doi:10.1016/j.cpc.2017.04.006.

[45] *Source code of DanceQ*, https://gitlab.com/DanceQ/danceq.

[46] *Documentation of DanceQ*, https://DanceQ.gitlab.io/danceq.

[47] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, J. Faibussowitsch *et al.*, *PETSc/TAO users manual*, Tech. Rep. ANL-21/39 - Revision 3.21, Argonne National Laboratory, doi:10.2172/2205494 (2024).

[48] S. Balay, S. Abhyankar, M. F. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. M. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, J. Faibussowitsch *et al.*, *PETSc Web page*, https://petsc.org/ (2024).

[49] V. Hernández, J. E. Román and V. Vidal, *Slepc: Scalable library for eigenvalue problem computations*, In J. M. L. M. Palma, A. A. Sousa, J. Dongarra and V. Hernández, eds., *High Performance Computing for Computational Science — VECPAR 2002*, pp. 377–391. Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-540-36569-3, doi:https://doi.org/10.1007/3-540-36569-9_25 (2003).

[50] V. Hernandez, J. E. Roman and V. Vidal, *Slepc: A scalable and flexible toolkit for the solution of eigenvalue problems*, ACM Trans. Math. Softw. **31**(3), 351–362 (2005), doi:10.1145/1089014.1089019.

[51] J. P. M. Schalkwijk, *An algorithm for source coding*, IEEE Transactions on Information Theory **18**(3), 395 (1972), doi:10.1109/TIT.1972.1054832.

[52] T. M. Cover, *Enumerative source encoding*, IEEE Transactions on Information Theory **19**(1), 73 (1973), doi:10.1109/TIT.1973.1054929.

[53] J. Schnack, P. Hage and H.-J. Schmidt, *Efficient implementation of the Lanczos method for magnetic systems*, Journal of Computational Physics **227**(9), 4512 (2008), doi:10.1016/j.jcp.2008.01.027.

[54] K. Bärwinkel, H. J. Schmidt and J. Schnack, *Structure and relevant dimension of the Heisenberg model and applications to spin rings*, Journal of Magnetism and Magnetic Materials **212**(1), 240 (2000), doi:10.1016/S0304-8853(99)00579-X.

[55] A. I. Streltsov, O. E. Alon and L. S. Cederbaum, *General mapping for bosonic and fermionic operators in Fock space*, Phys. Rev. A **81**, 022124 (2010), doi:10.1103/PhysRevA.81.022124.

[56] A. Szabados, P. Jeszenszki and P. R. Surján, *Efficient iterative diagonalization of the Bose–Hubbard model for ultracold bosons in a periodic optical trap*, Chemical Physics **401**, 208 (2012), doi:https://doi.org/10.1016/j.chemphys.2011.10.003, Recent advances in electron correlation methods and applications.

[57] L. Maciej, S. Anna, A. Veronica, D. Bogdan, S. Aditi and S. Ujjwal, *Ultracold atomic gases in optical lattices: mimicking condensed matter physics and beyond*, Advances in Physics **56**(2), 243 (2007), doi:10.1080/00018730701223200.

[58] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, Society for Industrial and Applied Mathematics, doi:10.1137/1.9781611970739 (2011).

[59] F. Gray, *Pulse code communication.*, United States Patent Number 2632058. (1953).

[60] O. Di Matteo, A. McCoy, P. Gysbers, T. Miyagi, R. M. Woloshyn and P. Navrátil, *Improving Hamiltonian encodings with the Gray code*, Phys. Rev. A **103**, 042405 (2021), doi:10.1103/PhysRevA.103.042405.

[61] D. H. Lehmer, *Teaching combinatorial tricks to a computer* (1960).

[62] M. Fishman, S. R. White and E. M. Stoudenmire, *The ITensor Software Library for Tensor Network Calculations*, SciPost Phys. Codebases p. 4 (2022), doi:10.21468/SciPostPhysCodeb.4.

[63] M. Fishman, S. R. White and E. M. Stoudenmire, *Codebase release 0.3 for ITensor*, SciPost Phys. Codebases pp. 4–r0.3 (2022), doi:10.21468/SciPostPhysCodeb.4-r0.3.

[64] D. J. Luitz, N. Laflorencie and F. Alet, *Many-body localization edge in the random-field Heisenberg chain*, Phys. Rev. B **91**(8), 081103 (2015), doi:10.1103/PhysRevB.91.081103.

[65] P. Sierant, M. Lewenstein and J. Zakrzewski, *Polynomially Filtered Exact Diagonalization Approach to Many-Body Localization*, Physical Review Letters **125**(15), 156601 (2020), doi:10.1103/PhysRevLett.125.156601.

[66] D. J. Luitz, *Polynomial filter diagonalization of large Floquet unitary operators*, SciPost Phys. **11**(2), 021 (2021), doi:10.21468/SciPostPhys.11.2.021.

[67] K. De Raedt, K. Michielsen, H. De Raedt, B. Trieu, G. Arnold, M. Richter, T. Lippert, H. Watanabe and N. Ito, *Massively parallel quantum computer simulator*, Computer Physics Communications **176**(2), 121 (2007), doi:https://doi.org/10.1016/j.cpc.2006.08.007.

[68] H. De Raedt, F. Jin, D. Willsch, M. Willsch, N. Yoshioka, N. Ito, S. Yuan and K. Michielsen, *Massively parallel quantum computer simulator, eleven years later*, Computer Physics Communications **237**, 47 (2019), doi:https://doi.org/10.1016/j.cpc.2018.11.005.

[69] R. Schäfer, B. Placke, O. Benton and R. Moessner, *Abundance of Hard-Hexagon Crystals in the Quantum Pyrochlore Antiferromagnet*, Phys. Rev. Lett. **131**, 096702 (2023), doi:10.1103/PhysRevLett.131.096702.

[70] R. Schäfer and D. J. Luitz, *Data for "DanceQ: High-performance library for number conserving bases" [arXiv:2407.14591]*, doi:10.5281/zenodo.12798598 (2024).

[71] W. Feller, *An Introduction to Probability Theory and Its Applications*, Bd. 1-2. Wiley, ISBN 9780471257097 (1957).