Hashing algorithms, optimized mappings and massive parallelization of multi-configuration methods for bosons

Alex V. Andriati[a,*], Arnaldo Gammal[a]

[a] *Instituto de Física, University of São Paulo, CEP 05508-090, São Paulo-SP, Brazil*

Abstract

We report memory management routines to aid the manipulation of Fock states in multi-configuration methods and to use bosonic creation and annihilation operators in the spanned Hilbert space, defined by the total number of combinations of a finite number of particles in few individual particle states. From a basic combinatorial problem to map integer numbers to each Fock state, we build up routines to handle creation and annihilation operators, step by step, showing the performance gain and memory consumption for different implementations, highlighting the limits of applicability and time demanded for each one. We also exploited massive parallel processors from graphics processor units with CUDA to improve a routine to act with the many-body Hamiltonian on the spanned configurational space, which demonstrated quantitatively the scalability of the problem and showed a drastic performance gain for the number of particles bigger than the number of individual particle states.

Keywords: many-particle physics, multi-configuration, hashing, bosons, CUDA

PROGRAM SUMMARY

Program Title:

Licensing provisions: GPLv3

Programming language: C, OpenMP C, Cuda. The C programs were tested with the GNU and Intel compilers. The programs using graphics cards were tested with the NVIDIA Cuda compiler version 10.1

Nature of problem: The present programs measure time required to assemble the one- and two-body density matrices and the time to act with the Hamiltonian operator in a state represented in Fock state basis

Solution method: We employ hashing algorithms to map vectors of integers used in fock states to integer numbers, thus allowing an enumeration of the coefficients of an expansion of a many-particle state in the configurational basis. We also employ mappings to track the action of creation and annihilation operators among the Fock states to improve performance.

*corresponding author

Email addresses: andriati@if.usp.br (Alex V. Andriati), gammal@if.usp.br (Arnaldo Gammal)

1. Introduction

Quantum and statistical mechanics often resort to second quantized formalism, specially when dealing with a system of identical particles, which automatically takes into account the symmetry of the many-particle wave function. In this formalism, all observables can be expressed in terms of creation and annihilation operators, defined by two possible algebras

$$[\hat{a}_k, \hat{a}_l^\dagger] = \delta_{kl}, \tag{1}$$

for bosons and

$$\{\hat{c}_k, \hat{c}_l^\dagger\} = \delta_{kl}, \tag{2}$$

for fermions, where $[A, B] = AB - BC$ and $\{A, B\} = AB + BA$.

It is required in the formalism a complete set of individual particle states (IPS), whose the creation/annihilation index refers to. A generic set of single particle states $\{\phi_k\}_{k\in\mathbb{N}}$, allows us to write the expression for the non-interacting many-particle operators as

$$\hat{\mathcal{T}} = \sum_{k,l} \hat{a}_k^\dagger \hat{a}_l \langle \phi_k | T | \phi_l \rangle, \tag{3}$$

and the interacting many-particle operators as

$$\hat{\mathcal{V}} = \frac{1}{2} \sum_{k,l,q,s} \hat{a}_k^\dagger \hat{a}_s^\dagger \hat{a}_l \hat{a}_q \langle \phi_k, \phi_s | V | \phi_q, \phi_l \rangle, \tag{4}$$

or even more general operators that are a combination of interacting and non-interacting many-particle operators, as is the case of the Hamiltonian, which can be the sum of both $\hat{\mathcal{H}} = \hat{\mathcal{T}} + \hat{\mathcal{V}}$. Here and throughout this paper the many-body operators in second quantized form are denoted with a hat while uppercase letters without hat are used for ordinary one and two-body operators. Besides, to the IPS we use lowercase Greek letters while for many-body states the uppercase Greek letters are used. Eqs. (3) and (4) are valid for fermions and bosons. For now on we treat specifically the case of bosons.

Formally, the sums presented in Eqs. (3) and (4) run over an infinite set of indexes that enumerate the IPS, and it is where most physical approximations starts. As the main example, we have the mean field theory for bosons, which yields the Gross-Pitaevskii equation that is widely used to describe cold atomic clouds and Bose-Einstein condensates [1, 2, 3, 4]. In this mean field approximation, it is considered a single relevant mode of the creation/annihilation operators, that corresponds to a macroscopically occupied state, that satisfies $\langle \hat{a}_0^\dagger \hat{a}_0 \rangle / N \approx 1$.

Other relevant approaches admit more than a single mode as used in mean-field theory, like the Bose-Hubbard model [5], that has described cold atomic clouds loaded in optical lattices [6, 7], and explained rather well the insulator and superfluid phases [8, 9, 10] in these systems. Another important example is a two mode mean field approach, which is used for two interacting Bose-Einstein condensates, for instance to explain Josephson junction effects [11], and to add spin internal degree of freedom in multi-component Gross-Pitaevskii equations [12, 13].

The approximation techniques cited above, however, are exact in some specific limits [14, 15], or are reasonable for weak interactions where depletion from the modes considered are low [16], since terms in Eq. (4) couple all occupations in different IPS. However, in the past decade, the advances in probing correlation functions in cold atomic clouds [17, 18, 19, 20] and the lack of methods that systematically includes depletion from the condensate due to strong interactions, boosted the development of a numerical method that could include a variable number of IPS to describe the physical systems.

In this scenario, the many-particle state can in principle be written as a linear combination of Fock states, vectors of the Hilbert space in the second quantized formalism, which are simultaneous eigenstates of $\hat{a}_k^\dagger \hat{a}_k$ for every $k \in \{1, ..., M\}$ that label the IPS with the corresponding eigenvalue being the number of particles that are in the IPS k. They can be written based on what is called *configuration*, used to name a possible set of occupation numbers, by

$$|\vec{n}^{(\beta)}\rangle \doteq |n_1^{(\beta)} \; ... \; n_M^{(\beta)}\rangle \;, \text{ with } \sum_{j=1}^{M} n_j^{(\beta)} = N, \;\forall \beta, \tag{5}$$

where N is the total number of particles, M the number of IPS and $\vec{n}^{(\beta)}$ a possible configuration labeled by β.

The total number of configurations $\vec{n}^{(\beta)}$ is obtained from the combinatorial problem of how to fit N identical balls in M boxes, which yields for bosons

$$N_c(N, M) = \binom{N+M-1}{M-1} = \frac{(N+M-1)!}{N!(M-1)!}, \tag{6}$$

what implies $\beta \in \{1, 2, ..., N_c(N, M)\}$. Since there are many way to select the occupation numbers under the constraint of Eq. (5), expressing the many-body state in this basis is called a Multi-configuration method.

Well known methods that employ Multi-configurational spaces described above are the Exact Diagonalization (ED) [21, 22, 23, 24, 25] which most part of the time is used to find the low energy spectrum with Lanczos algorithm [26, 27], which consider the IPS fixed, and the Multi-configuration Time-Dependent Hartree method for Bosons (MCTDHB) [28, 29, 30, 31, 32], which determines the IPS variationally by action minimization. Both ED and MCTDHB have been used in different physical systems to study the ground state and dynamics as well.

In this basis of configurations, we can express the many-particle state of the system by a linear combination as

$$|\Psi(t)\rangle = \sum_{\beta=1}^{N_c(N,M)} C_\beta(t) |\vec{n}^{(\beta)}\rangle, \tag{7}$$

where $\vec{C}(t)$ is a complex vector of dimension $N_c(N, M)$.

Independently of how the IPS are picked up, based on ED or MCTDHB, the operators in Eq. (3) and (4) need to be represented in the configurational basis, which requires some way to act with creation/annihilation operators on the Fock states. This problem is studied in this article, starting from the fundamental question on how to establish the relation between β and its configuration $\vec{n}^{(\beta)}$. A function that does this job is called a hashing function and we have a perfect hashing if we get a one-to-one correspondence.

This problem has been studied in the last three decades at least, where some effort was done on developing hashing functions to search for configuration states [23, 33], although most part of more recent works have been directed to particles confined in sites of optical lattices [24], restricting to the Bose-Hubbard model [22], or for spin systems [23, 25]. A more recent proposal for hashing function was showed in [34], though most part of the results are for fermionic spin systems.

The operators in the second quantized formalism also demand a conversion between configurations due to the action of creation/annihilation operators, which induce rearrangements of the occupation numbers. We can define mappings to track every configuration to others, that are related by the rearrangement of 1 or 2 particles among the IPS. The number of possibilities can be very large depending on the size of the configurational space and to implement such structures for arbitrary number of IPS and particles can be challenging.

In Ref. [33] it was obtained a perfect hashing function to establish the connection of configurations and integer numbers for bosons, without any restriction on the Hamiltonian, like symmetries or only nearest neighbors interaction, for an arbitrary number of IPS. In the following, we present an alternative derivation of this hashing table directly for bosons from a cost function. We implement direct mappings from configuration to configuration, whose the occupation numbers differs from each other due to the action of one or two pairs of creation/annihilation operators and show a detailed discussion on the performance gain and memory requirement by this refinement. Finally, we study the impact of using parallel processors comparing a multi-threaded CPU with Graphics Processing Units (GPUs) with CUDA interface.

The physical operators chosen to illustrate the performance throughout this article are the one- an two-body density matrices and the Hamiltonian. These quantities are essential for the MCTDHB [28], which usually have to be evaluated several times for the same configurational space, thus emphasizing the relevance of our study on the problem. The one- and two-body matrices are given by the expectation values of a combination of two or four creation and annihilation operators as

$$\rho^{(1)}_{kl} = \langle \Psi | \hat{a}^\dagger_k \hat{a}_l | \Psi \rangle = \sum_{\gamma=1}^{N_c(N,M)} \sum_{\beta=1}^{N_c(N,M)} C^*_\gamma C_\beta \langle \vec{n}^{(\gamma)} | \hat{a}^\dagger_k \hat{a}_l | \vec{n}^{(\beta)} \rangle, \tag{8}$$

and

$$\rho^{(2)}_{klqs} = \langle \Psi | \hat{a}^\dagger_k \hat{a}^\dagger_l \hat{a}_q \hat{a}_s | \Psi \rangle = \sum_{\gamma=1}^{N_c(N,M)} \sum_{\beta=1}^{N_c(N,M)} C^*_\gamma C_\beta \langle \vec{n}^{(\gamma)} | \hat{a}^\dagger_k \hat{a}^\dagger_l \hat{a}_q \hat{a}_s | \vec{n}^{(\beta)} \rangle, \tag{9}$$

respectively. For the matrix elements of the Hamiltonian, that in general is a sum of parts given in Eqs. (3) and (4), we choose not to store the matrix, which is very sparse. In this way our routines only require the matrices elements $\langle \phi_k | T | \phi_l \rangle$ and $\langle \phi_k, \phi_s | V | \phi_q, \phi_l \rangle$ to evaluate the sum

$$\sum_{\beta=1}^{N_c(N,M)} \langle \vec{n}^{(\gamma)} | \hat{\mathcal{H}} | \vec{n}^{(\beta)} \rangle C_\beta, \tag{10}$$

for every γ, that represents the action of the Hamiltonian written in terms of the Fock states. This could be further used to address the time evolution in the configurational basis with the equation for the coefficients given by

$$i\hbar \frac{\mathrm{d}C_\gamma}{\mathrm{d}t} = \sum_{\beta=1}^{N_c(N,M)} \langle \vec{n}^{(\gamma)} | \hat{\mathcal{H}} | \vec{n}^{(\beta)} \rangle C_\beta(t). \tag{11}$$

The clear advantage in not storing the Hamiltonian matrix appears when dealing with a method that needs to update the IPS $\{\phi_k\}$, like in the MCTDHB because they are time-dependent, which would require to reassemble the matrix during the time evolution. There is also interest when the goal is to diagonalize using the Lanczos algorithm, where one only needs a routine to apply the matrix on a vector and not the matrix itself.

2. Mapping Fock states to integers

Following the theoretical framework developed in the previous section, as a first step, it is necessary to address an integer number for each configuration. The routine to perform this task assigns a cost for every IPS to be occupied, starting with all occupations zero. The problem can be depicted by a basket of balls (particles), that starts with N, the total number of particles, and is emptied particle by particle to fill the IPS.

Given an enumeration for the IPS from 0 to $M-1$, if we take one with number k where $0 \leq k \leq M-1$, there are other k IPS below it since we are counting the number zero. In this way, the cost to put one particle in the state k is defined by the total number of configurations of remaining particles in the basket over all previous IPS. In other words, the cost is all the combinations we could do with the lower number IPS and particles in the basket. When the basket has none particle left, the process is finished and the total cost will be the index of the configuration.

In Fig. 1 is depicted a practical example of the description above. The combination function defined in Eq. (6) plays a crucial role being used to compute the costs. For instance, if p_n is the IPS the n-th particle occupies, where by construction we have $0 \leq p_1 \leq ... \leq p_N < M$, then the total cost mentioned above can be compute by

$$I(\vec{p}) = \sum_{n=1}^{N} N_c(n, p_n) = \sum_{n=1}^{N} \binom{n + p_n - 1}{p_n - 1}, \ \forall p_n > 0; \quad N_c(n, 0) = 0, \forall n. \tag{12}$$

This relation is identical to the results in Ref. [33], though some conventions are changed and there the derivation follows an alternative way, using a correspondence to fermions. Moreover from [33], it is already known that this relation maps uniquely integers to configurations without any left number and therefore indicates a perfect hashing function.

Algorithm 1 Get configuration Index from a configuration

```
Require: $N > 0,\ M > 0,$ occupation vector $\vec{n}$
  $k \leftarrow 0$
  $s \leftarrow N$
  for $m = M - 1..1$ do
    $j \leftarrow n[m]$
    while $j > 0$ do
      $k \leftarrow k + N_c(s, m)$
      $s \leftarrow s - 1$
      $j \leftarrow j - 1$
    end while
  end for
  return $k$
```

Algorithm 2 Build configuration $\vec{n}$ from index β

```
Require: $N > 0,\ M > 0,\ N_c(N, M) > \beta \geq 0$
  for $i = 0..M - 1$ do
    $n[i] \leftarrow 0$
  end for
  $k \leftarrow \beta$
  $m \leftarrow M - 1$
  $s \leftarrow N$
  while $k > 0$ do
    while $k - N_c(s, m) < 0$ do
      $m \leftarrow m - 1$
    end while
    $k \leftarrow k - N_c(s, m)$
    $n[m] \leftarrow n[m] + 1$
    $s \leftarrow s - 1$
  end while
  if $s > 0$ then
    $n[0] \leftarrow n[0] + s$
  end if
```

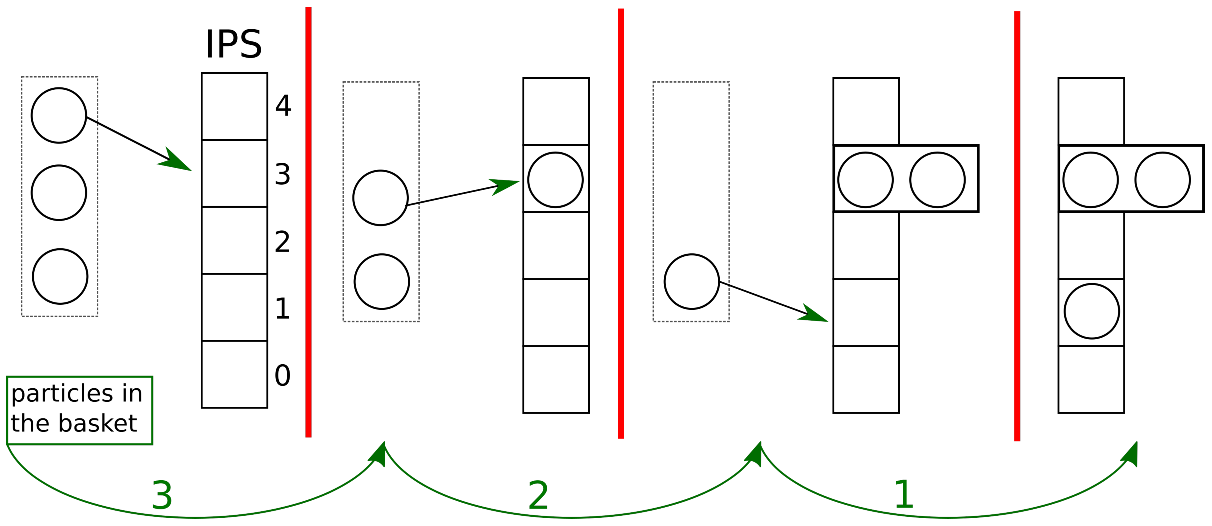

Figure 1: Example to illustrate the process of IPS occupation for $N = 3$ and $M = 5$ for a the specific configuration $|0, 1, 0, 2, 0\rangle$. The total cost of this configuration is $N_c(3, 3) + N_c(2, 3) + N_c(1, 1) = 17$ where the terms are presented following the order of arrows.

The prescription of the hashing can now be implemented. Given a configuration, to discover its index, we sum up the costs, removing particle by particle using Eq. (12). This procedure detailed in Algorithm 1. The reverse process is quite straightforward, to assemble the configuration given an index between 0 and $N_c(N, M) - 1$, we need to put all the particles in a basket and check, starting from the IPS $M - 1$, if the index is bigger than the cost to put a particle. In positive case, it transfers a particle from the basket to the IPS, otherwise move to lower cost IPS and try again. The routine to convert index to configuration is given in Algorithm 2. The algorithms 1 and 2 are the core functions to operate with creation and annihilation given a many-body state in the configurational basis. Explicit C code implementation for both are given in the attached file "configurationsMap.h", where the function "FockToIndex" corresponds to algorithm 1 and "IndexToFock" to algorithm 2.

In the computation of the density matrices Eqs. (8) and (9), we need to perform just the sum in β, whereas for each β there is a unique value for γ, the one corresponding to the configuration after replacing the particles due to the action of the creation and annihilation operators. Therefore, for β running from 0 to $N_c(N, M) - 1$, we need three steps to perform the operation required. First, obtain the configuration using algorithm 2. Second, reconfigure the occupation according to the action of creation/annihilation operators. Third, use this new occupation vector to compute the corresponding index γ using algorithm 1 and do the multiplication of coefficients.

Nevertheless, we must use the algorithms 2 and 1 $N_c(N, M)$ times for every element of the density matrices, that results in a total of $M^2 N_c(N, M)$ calls of both functions to setup all the elements of $\rho^{(1)}$ and $M^4 N_c(N, M)$ for $\rho^{(2)}$[1]. However, we may spend more memory

[1]Actually, this number can be halved if one use hermiticity, and reduced even more using the commutation relations in the case of $\rho^{(2)}$.

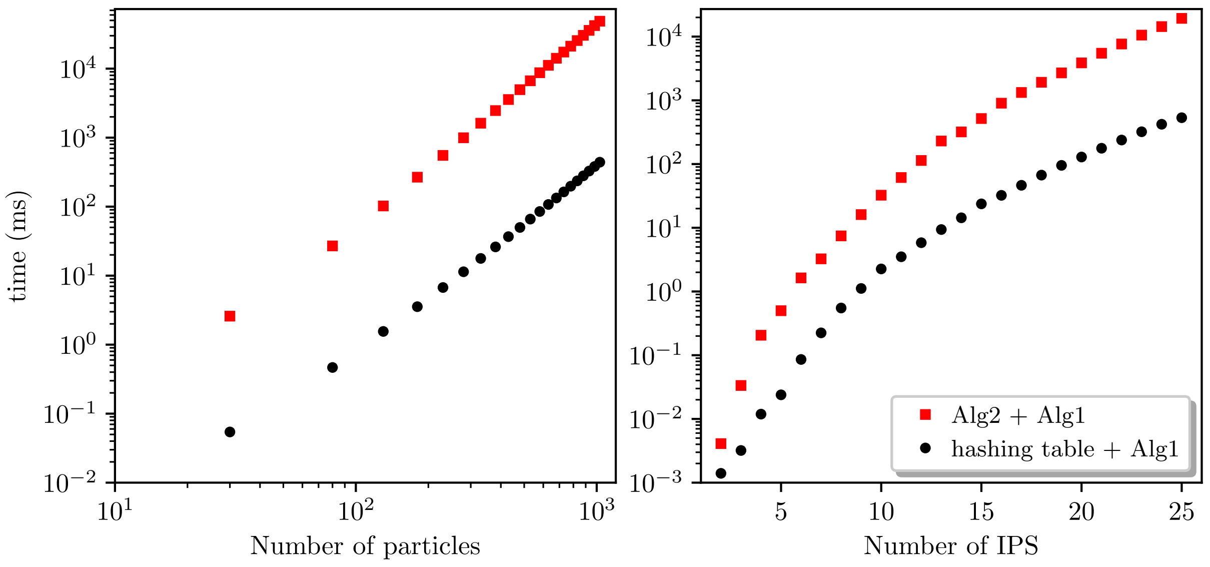

Figure 2: Time to compute all elements of $\rho^{(1)}$. The red squares correspond to an implementation that uses just the conversion algorithms 2 and 1 and the black circles make use of a hashing table to store and sort the configurations, which restrict to use only algorithm 1. In the left panel was fixed 3 IPS while varying the number of particles and in the right panel was varied the number of IPS with 5 particles.

creating some structures to avoid the number of calls of these functions, whose will improve performance as will be shown later. Surely, the setup of any new structure would demand some equivalent time but, we again emphasize that our goal are problems that need to compute these quantities several times for the same configurational space [28].

A first improvement is to build once all occupation vectors and maintain them stored during all operations, defining a hashing table. For instance, they can be stored along rows of a matrix of integers, with the row number being the index of the respective configuration. This hashing table would require to store $MN_c(N,M)$ integers in exchange of avoiding calls of algorithm 2 when computing $\rho^{(1)}$ and $\rho^{(2)}$. In the attached codes, the "setupFocks" function in "configurationsMap.h" file is a C code implementation that builds the hashing table.

In Fig. 2 we compute $\rho^{(1)}$ in an implementation, using both algorithms 2 and 1, and an improved one, that uses hashing table to setup and sort occupation vectors previously and require only algorithm 1. The basic difference between the two implementations is the call of algorithm 2 face to a memory access of the hashing table. As can be noted in Fig. 2, the performance gain is critical, highlighted by the logarithm scale, for both cases, when varying the number of particles or the number of IPS. Moreover for the left panel, the time required with respect to the number of particles has a clear linear relation in logarithmic scales, which indicates a power law.

The density matrix $\rho^{(1)}$ is computed in Fig. 2 by randomly generating the components of the vector $\vec{C}$ and then normalizing to one to use Eq. (8) and the rules detailed in Appendix B. The output of both implementations for each case provides a self-consistency check.

Other routines, to build the two-body density matrices and to apply the Hamiltonian using the configuration basis shall benefit even more from the hashing table, because they require much more operations. We thus focus on these two quantities for the next improvements.

3. Mapping routines

The next step is to set the routines to compute the observables of Eqs. (8), (9) and (10) completely free from calls of the algorithm 1 as well. For this aim, it is necessary to define a structure where given an index it has stored all possible jumps[2] of one and two particles among the IPS, corresponding to the action of one and two pairs of creation/annihilation operators respectively.

In a single particle jump, one has (at most)M different states to remove a particle and M different states to place it back, which implies that for every configuration there are at most M^2 possible transitions. Thus a straightforward way to map all these transitions is to define a triple indexed structure, which stores integers, where the first index is from the configuration number, and the other two are IPS numbers, one from where the particle is being destroyed and other where it is being created. This one-particle jump mappings would require $N_c(N, M)M^2$ new integers to store. The function "OneOneMap" in "configurationsMap.h" file implement in C code this mappings using an array of integers.

It is worth pointing out that the memory cost for this one-particle jump mapping is greater than the first improvement of the hashing table, where in that case was stored all the occupation numbers and therefore had a cost of $N_c(N, M)M$ integers. This justify why we did not mind about memory cost at that stage, though it will matter in the following. Moreover, the $N_c(N, M)M^2$ integers wastes some memory because there are configurations with some empty IPS, which actually do not have M^2 possible transitions. Nevertheless, this wasted memory here will not matter as well, since the two-particle jump mappings will require more elements than $N_c(N, M)M^2$, as will be shown later.

We now analyze how to implement a structure that maps double jumps, that is, two particles move to different IPS. If we follow the same idea presented for the one-particle jump, for each configuration there would be at most M^2 possibilities to take two particles from the occupation numbers and for each one of these possibilities there are again M^2 ways to replace them. Following this naive way, we would end up with the additional memory requirement of $N_c(N, M)M^4$ integers. Nevertheless, it is possible to reduce this number.

A thoroughly inspection over configurations show us that M^2 possibilities to take two particles from the IPS (equivalent to the action of two annihilation operators) may be not true for most part of the configurations, because there are many configurations with empty IPS. The real number of possibilities can be obtained as follows: for every non-empty IPS k we search for $s \geq k$ non-empty as well and, whenever we find such numbers, we will have M^2 possible IPS to replace these particle taken from k and s.

[2]Jump here means the simultaneous destruction and creation of particle in different states.

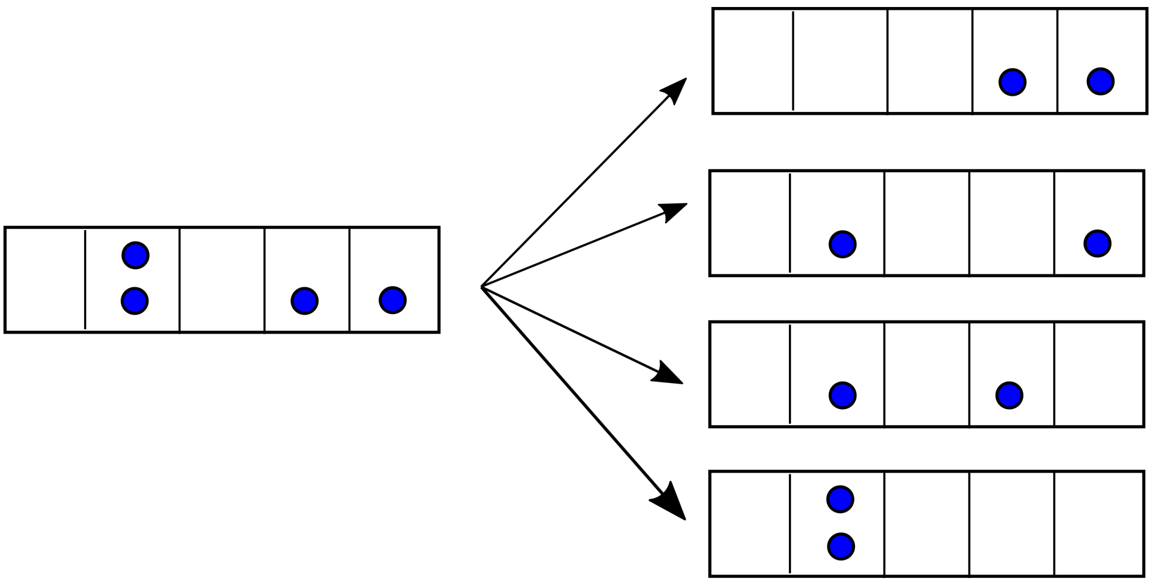

Figure 3: In the left a possible configuration for $M = 5$ and $N = 4$. In the right the arrows indicate all possible ways to remove 2 particles from the single particle states, that is 4.

In Fig. 3, it is illustrated for a simple case, given a specific configuration, the possible ways to remove simultaneously two particles by the action of two annihilation operators. The arrows conducts to the possible outcomes, where for each one, we have M^2 possibilities to replace the particles. Originally, the naive way would store a lot of useless information since it considers a bunch of forbidden transitions, that is, removal from empty states. For instance, in the case represented in Fig. 3, it would require $5^4 = 625$ possibilities, while there are only $4 \times 5^2 = 100$ real possibilities.

In summary, to save memory for this structure that we are going to define, we cannot allocate those forbidden transitions. In order to overcome the problem, we define a structure like a hashing table, though each line of the table has a variable number of elements, where the line number correspond to an index of a configuration. Its elements are integers, indexes of other configurations that are outcomes of all possible jumps of two particles.

A possible way to sort the elements for each line in the table is starting with the IPS $k = 0$ up to $k = M - 1$, we take $k \leq s < M$ and for each possible simultaneous removing of particles in k and s, we have a stride of M^2 integer numbers that corresponds to new configurations obtained for every possible way to replace the particles removed. Therefore, if one wants to know, the configuration index γ, that is a result of transferring particles of another configuration $\vec{n}^{(\beta)}$ from IPS i and $j > i$ to q and l IPS, it is required to check out how many strides must be ignored. In this case, the number of strides is the number of possible simultaneous removal from IPS k and s for every $k = 0, ..., i$ and $s = k, ..., j - 1$.

For example, suppose in Fig. 3 we are interested in the transitions that come from removing the last two particles, thus we need to skip 3 strides. In other words, in our table, in the line corresponding to the configuration in the figure, we need to skip the 3×5^2 elements to get the indexes of configurations we are interested.

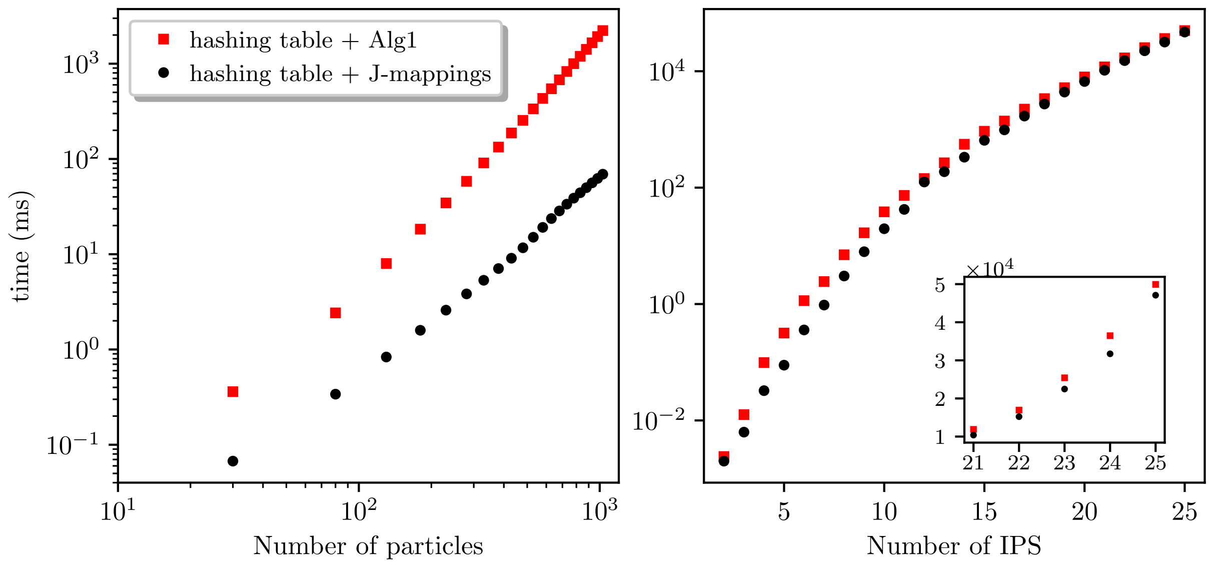

Figure 4: Time to compute all elements of $\rho^{(2)}$. The red squares correspond to an implementation that uses hashing table of configurations and calls of algorithm 1 while the black circles refers to one that uses direct jump mappings between configurations related by the action of the creation/annihilation operators and dismiss completely the use of algorithms 2 and 1. In the left while varying the number of particles it was taken 3 IPS fixed and in the right 5 particles was used throughout the curve with respect to IPS.

The implementation for the two-jump mapping described above are done in the codes splitting the problem in two parts. The first mappings refers to annihilation of two particles from the same IPS, and is implemented by the function “OneTwoMap” in “configurationsMap.h” file. The second refers to annihilation of 2 particle in necessarily different IPS and is implemented in “TwoTwoMap” function in the same file. This procedure avoids conditional statements when computing the number of strides, and is suitable for the different rules presented presented in the Appendix B

In the following we compare the performance between implementations that uses only hashing table and the ones that uses jump mapping. Before moving on, in the supplemental C code files, the “demonstrateFockMap.c” can be executed to see the basic functionality of the structures described so far. Its execution is rather simple and requires only the number of particles and number of IPS as command line arguments to print the hashing table and some random jumps, which are addressed to other configuration using the mappings.

In Fig. 4 the performance is compared between two routines that setup $\rho^{(2)}$. The first, uses the hashing table of configurations and algorithm 1. The second, does not use any of the algorithms of conversion between indexes and configurations, instead, uses the hashing table and jump mappings (both of one and two-particle jump) explained above. The hashing table is still required to exclude forbidden transition in the rules in Appendix B. For large number of particles, we see a good performance gain, while for high number of IPS, the gain is slight, indicating that the use of algorithm 1 is not the bottleneck in this case. The C code routines used to assemble $\rho^{(2)}$ are in “twobodyMatrix.h” file. Specifically, to generate

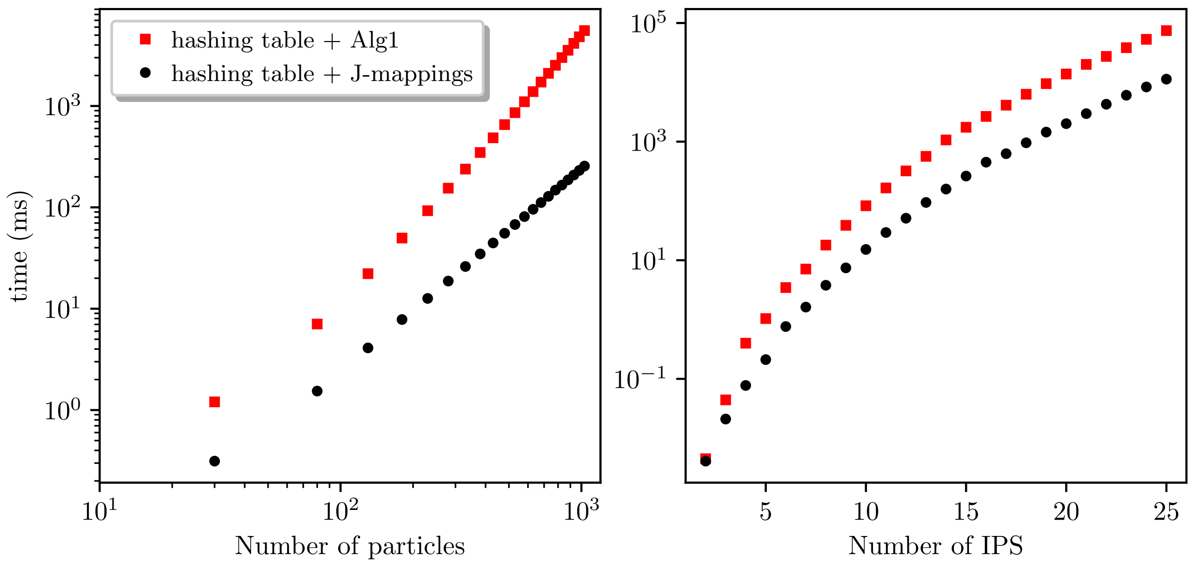

Figure 5: Time required to act with Hamiltonian operator in the configuration basis as in . As done for $\rho^{(2)}$ in Fig. 4 the red squares correspond to a routine that uses hashing table and algorithm 1 and the black circles to one that uses mappings instead of algorithm 1. In the left panel we vary the number of particles for 3 IPS and in the right panel the number of IPS for 5 particles.

the figure data, it was used "TBrho" that corresponds to the implementation that uses the hashing table + algorithm 1 and "TBrho_XX" that uses hashing table + jump mappings .

A careful inspection in algorithm 1 show us that it need to remove all particles from the configuration and thus demands the total number of particles as operations. Therefore, it is expected that the gain in performance using jump mappings is bigger for large number of particles than for large number of IPS. In other words, it is harder to empty many particles from few IPS than a few particles from many IPS.

Another very important routine to check the performance gain is the time to act with the Hamiltonian over a state expressed in the configurational basis, that is to evaluate the sum in Eq. (10). In the same way that was done for $\rho^{(2)}$, in Fig. 5 we compare the time demanded for evaluating the sum in Eq. (10) using two routines, again one using the hashing table and algorithm 1 and other using the hashing table and the jump mappings. Similarly there is a clear improvement varying the number of particle (left panel), but this time there is a substantial gain also varying the number of IPS. The C code function to act with the Hamiltonian in a vector of coefficients are in the file "hamiltonianMatrix.h", where the specific routines to generate Fig. 5 are "applyHconf" which uses the hashing table + algorithm 1 and "applyHconf_XX" which uses hashing table + jump mappings .

In all comparisons between the routines that used the algorithm 1 with the hashing table and those that use mappings, when varying the number of particles, there is an evident constant slope behaviour in the curve, at least, for large number of particles. This reveals that the time demanded respect to the number of particles can be written as a power law

	$\rho^{(2)}$		$\mathcal{H}$	
	a	b	a	b
hashing table	2.835(5)	$6.33(9) \times 10^{-6}$	2.791(5)	$2.13(3) \times 10^{-5}$
jump mappings	2.34(3)	$6.4(6) \times 10^{-6}$	2.000(2)	$2.40(2) \times 10^{-4}$

Table 1: Fitted parameters, for implementations using hashing table and jump mappings. As the only value for the number of IPS was 3, we dropped the subscript M from the parameters.

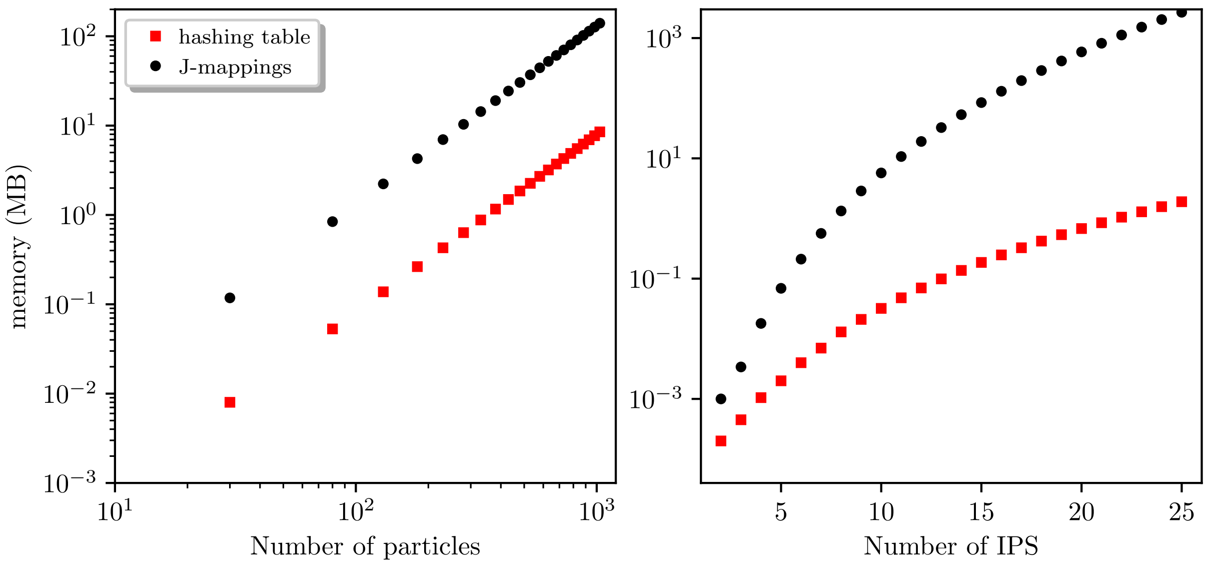

Figure 6: Memory allocation for the jump mappings structure of creation/annihilation operators and hashing table for the number of particles and IPS as used in previous figures, measured in megabytes (MB). In the left panel again we set 3 IPS fixed and in the right panel 5 particles were used.

in the form

$$\tau_M(N) = b_M N^{a_M}, \tag{13}$$

with M the number of IPS fixed. The parameters can be extracted from curve fitting, where a_M is the slope on the log scaled plot. Since the study varying the number of particles in all cases presented here were carried out with $M = 3$ then we dropped the index M from the parameters in the following.

For the more time demanding routines, that are for $\rho^{(2)}$ and $\mathcal{H}$, it was evaluated a linear curve fitting in the log scaled plots, which result in a power law of the form (13) for time as function of the number of particles. The values are showed in Tab. 1. The most important feature is that the mappings reduced the exponents for both cases of $\rho^{(2)}$ and $\mathcal{H}$, which shows that the improvement is more expressive as larger is the number of particles.

Nevertheless, despite we have emphasized how suitable was the introduction of the jump mappings structure, we need to check the limits of application in terms of the additional memory demanded. Indeed, all the gain in time had a cost in memory, as showed in Fig. 6. From the left panel, the case we vary the number of particles, we see that this cost is

	a	b
hashing table	1.9953(2)	$8.288(5) \times 10^{-6}$
jump mappings	2.0004(1)	$1.3158(1) \times 10^{-4}$

Table 2: Fitted parameters of power law for the memory consumption as function of N for $M = 3$ fixed.

relatively cheap, some hundreds of megabytes (MB), right the case the performance gain was more expressive. The case in the right panel shows that we can not ignore the memory consumption since it demanded up to some thousands of MB, which is not a problem for present workstations, but indicates that a possible limitation may come up if one extrapolate $M = 25$ IPS with $N = 5$ particles.

It is worth to highlight the very similar behaviour between the memory cost in Fig. 6 and time execution in Fig. 5, where again a constant slope can be identified in the log scale plot with respect to the number of particles. The results of the fitting parameters for memory consumption are shown in Tab. 3.

4. Massive parallel processors application

The use of GPUs to speed up numerical evaluations is not novel. In the past decade the interest for these tools has dragged attention due to their effectiveness in improvements for workstations, in some cases as good as non-GPU supercomputers. Here, in view of the Lanczos algorithm [26] for diagonalization and the time evolution Eq. (11) that is fundamental for the MCTDHB [28], we focus on an analysis based on the time required to apply the Hamiltonian in the configurational space.

Using the same parameters that were chosen throughout this article, in Fig. 7 we compare the time used for a routine to act with the Hamiltonian in the configuration space, with all mappings described in previous section, varying the number of threads in parallelization of the routine. For codes running in GPU we dynamically chose the number of blocks of threads, each one with 256 threads, for optimal usage of architecture using CUDA. Thus depending on the size of configurational space it adaptively setup the blocks.

Left panel in Fig. 7 shows that the function is highly scalable, since the presence of more threads improve dramatically the performance. Nevertheless, for the right panel we see that the scalability is much smaller than the first case. An explanation for this result lies in the amount of work each thread perform, since we have only 5 particles in this case, there are a lot of configurations that have forbidden transitions and thus some threads have much less operations to do, and this benefit the CPU cores that have higher clock frequency. Thus, the scalability in out implementation, depends on a filling factor, how many particles are by IPS, where as higher this filling factor is more equally will be the work among the threads.

The code of the parallelized routine to compute the action of the Hamiltonian is “applyHconf_omp” in “hamiltonianMatrix.h” file. The parallelization for multi CPU threads is done using the OpenMP API, as the name of the function already suggests. The codes that run on GPU are separated in the “cuda/” folder. The codes itself use specific technical details of the CUDA language, thus the file that implement the function to act with the Hamilto-

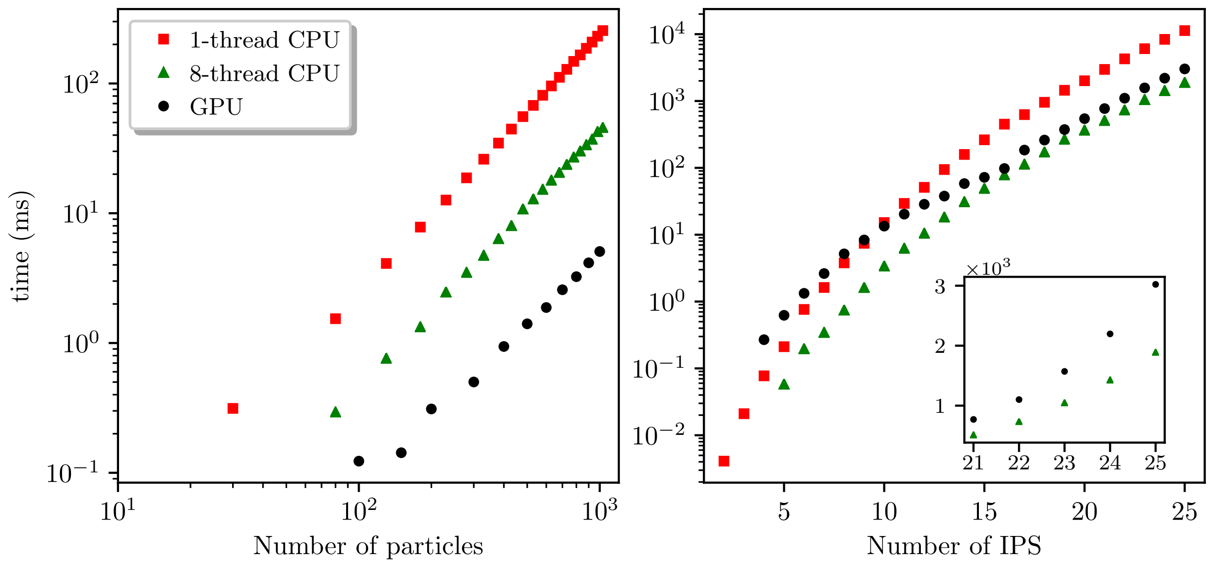

Figure 7: Performance experiment using single thread (red squares) and 8 threads (green triangles) of CPU and variable number of threads of GPU using CUDA (black circles). Just as before in the left panel was used 3 IPS and in the right 5 particles. The inset shows in normal scale the difference between CUDA and CPU parallelization for a region they are close in log scale plot.

nian "cuda/hamiltonianMatrix.cuh" is different from the routines in "hamiltonianMatrix.h", however is still readable for those experienced in C language.

The technical details of the workstations such as the CPU and GPU used in the simulations can be conferred in Appendix A.

5. Conclusion and Outlook

In this article, we brought different ways to implement an effective indexing of configurations to represent a many-particle state, which is of main concern for developing numerical multi-configuration methods. We generalize the problem assuming that all particles in any individual particle state interact with each other, without restricting to a lattice with just nearest neighbor interaction or focusing in spin system. Therefore the time demanded exposed here is an upper bound for any bosonic system.

It was discussed carefully the performance and limitations of the different ways to build routines for the main physical quantities, and how direct mappings of indexes can be done to track the action of creation/annihilation operators. The limits of applicability with the mappings structure is explored in terms of memory required and our results shows to be reproducible in fair workstations.

Beyond developing the algorithms, we carried out an study of the impact from massive parallelization using GPU and compared with CPU, revealing details about the scalability of the implementation as well. For the most demanding cases, for roughly 1000 particles the best improvement was achieved by the GPU with a time reduction by a factor 50 when compared to single thread, whereas for 25 IPS the best result was with 8-threaded CPU with approximately a time reduction by a factor 6 when compared to single thread execution.

All the codes used to present the result in this article are hosted in https://github.com/andriati-alex/mcpi.

Appendix A. Technical Details of the Resources

The CPU used was an Intel® Xeon® CPU E5-2620 v4 with clock rate 2.10GHz and 8 cores. The CPU parallelization were done using the OpenMP API version 4.5. The GPU used for simulation was a NVIDIA Tesla K40c, with CUDA compiler version 11.0. We stress that in the GPU architecture the threads are divided in blocks and we by default used 256 threads per block and define the number of blocks dynamically accordingly to the size of the configurational space, trying to keep one operation per thread up to the maximum number of threads available.

Appendix B. Matrix elements for the reduced two-body density matrices

We follow Ref. [35], and present the reduced one-body and two-body density matrices explicitly, addapted to our notation. Starting with a β configuration, β_a^b is a resulting configuration index where one particle from the a-th orbital is removed and then added to the b-th orbital. Analogously, starting from a β configuration, β_{ab}^{cd} is a resulting configuration index where, two particles are removed from the a-th and b-th orbitals and then added to the c-th and d-th orbital, respectively. Sums over β index ranges from 1 to N_c.

$$\rho_{kk} = \sum_{\beta}^{N_c} C_\beta^* C_\beta n_k, \qquad \rho_{ksks} = \sum_{\beta}^{N_c} C_\beta^* C_\beta n_k n_s,$$

$$\rho_{kl} = \sum_{\beta}^{N_c} C_\beta^* C_{\beta_k^l} \sqrt{(n_l+1)n_k}, \qquad \rho_{kkqq} = \sum_{\beta}^{N_c} C_\beta^* C_{\beta_{kk}^{qq}}$$

$$\rho_{kkkk} = \sum_{\beta}^{N_c} C_\beta^* C_\beta (n_k^2 - n_k), \qquad \rho_{kkql} = \sum_{\beta}^{N_c} C_\beta^* C_{\beta_{kk}^{ql}} \sqrt{(n_k-1)n_k(n_q+1)(n_l+1)},$$

$$\rho_{kkkl} = \sum_{\beta}^{N_c} C_\beta^* C_{\beta_k^l} (n_k-1)\sqrt{n_k(n_l+1)}, \qquad \rho_{ksqq} = \sum_{\beta}^{N_c} C_\beta^* C_{\beta_{ks}^{qq}} \sqrt{n_k n_s (n_q+1)(n_q+2)},$$

$$\rho_{ksss} = \sum_{\beta}^{N_c} C_\beta^* C_{\beta_k^s} n_s \sqrt{n_k(n_s+1)}, \qquad \rho_{kssl} = \sum_{\beta}^{N_c} C_\beta^* C_{\beta_k^l} n_s \sqrt{n_k(n_l+1)},$$

$$\rho_{ksql} = \sum_{\beta}^{N_c} C_\beta^* C_{\beta_{ks}^{ql}} \sqrt{n_k n_s (n_q+1)(n_l+1)}.$$

Aknowledgements

The authors thank the Brazilian agencies Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) and Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq). We are also grateful for NVIDIA for providing us the Tesla K40c GPU.

[1] Lev Pitaevskii and Sandro Stringari. *Bose-Einstein Condensation and Superfluidity, 2nd ed.* Oxford University Press, United Kingdom, 2016.

[2] M. H. Anderson, J. R. Ensher, M. R. Matthews, C. E. Wieman, and E. A. Cornell. Observation of Bose-Einstein condensation in a dilute atomic vapor. *Science*, 269(5221):198–201, 1995.

[3] Oliver Penrose and Lars Onsager. Bose-Einstein condensation and liquid helium. *Phys. Rev.*, 104:576–584, Nov 1956.

[4] Immanuel Bloch, Jean Dalibard, and Wilhelm Zwerger. Many-body physics with ultracold gases. *Rev. Mod. Phys.*, 80:885–964, Jul 2008.

[5] H. A. Gersch and G. C. Knollman. Quantum cell model for bosons. *Phys. Rev.*, 129:959–967, Jan 1963.

[6] D. Jaksch and P. Zoller. The cold atom hubbard toolbox. *Annals of Physics*, 315(1):52 – 79, 2005. Special Issue.

[7] D. Jaksch, C. Bruder, J. I. Cirac, C. W. Gardiner, and P. Zoller. Cold bosonic atoms in optical lattices. *Phys. Rev. Lett.*, 81:3108–3111, Oct 1998.

[8] Matthew P. A. Fisher, Peter B. Weichman, G. Grinstein, and Daniel S. Fisher. Boson localization and the superfluid-insulator transition. *Phys. Rev. B*, 40:546–570, Jul 1989.

[9] T. D. Kühner and H. Monien. Phases of the one-dimensional Bose-Hubbard model. *Phys. Rev. B*, 58:R14741–R14744, Dec 1998.

[10] C. Bruder, Rosario Fazio, and Gerd Schön. Superconductor–mott-insulator transition in Bose systems with finite-range interactions. *Phys. Rev. B*, 47:342–347, Jan 1993.

[11] Michael Albiez, Rudolf Gati, Jonas Fölling, Stefan Hunsmann, Matteo Cristiani, and Markus K. Oberthaler. Direct observation of tunneling and nonlinear self-trapping in a single bosonic josephson junction. *Phys. Rev. Lett.*, 95:010402, Jun 2005.

[12] M.-S. Chang, C. D. Hamley, M. D. Barrett, J. A. Sauer, K. M. Fortier, W. Zhang, L. You, and M. S. Chapman. Observation of spinor dynamics in optically trapped ^{87}Rb Bose-Einstein condensates. *Phys. Rev. Lett.*, 92:140403, Apr 2004.

[13] Chunji Wang, Chao Gao, Chao-Ming Jian, and Hui Zhai. Spin-orbit coupled spinor Bose-Einstein condensates. *Phys. Rev. Lett.*, 105:160403, Oct 2010.

[14] Elliott H. Lieb and Robert Seiringer. Proof of Bose-Einstein condensation for dilute trapped gases. *Phys. Rev. Lett.*, 88:170409, Apr 2002.

[15] Elliott H. Lieb, Robert Seiringer, and Jakob Yngvason. Bosons in a trap: A rigorous derivation of the Gross-Pitaevskii energy functional. *Phys. Rev. A*, 61:043602, Mar 2000.

[16] Raphael Lopes, Christoph Eigen, Nir Navon, David Clément, Robert P. Smith, and Zoran Hadzibabic. Quantum depletion of a homogeneous Bose-Einstein condensate. *Phys. Rev. Lett.*, 119:190404, Nov 2017.

[17] M. Naraschewski and R. J. Glauber. Spatial coherence and density correlations of trapped Bose gases. *Phys. Rev. A*, 59:4595–4607, Jun 1999.

[18] R. G. Dall, A. G. Manning, S. S. Hodgman, Wu RuGway, K. V. Kheruntsyan, and A. G. Truscott. Ideal n-body correlations with massive particles. *Nat. Phys.*, 9:341–344, May 2013.

[19] S. S. Hodgman, R. I. Khakimov, R. J. Lewis-Swan, A. G. Truscott, and K. V. Kheruntsyan. Solving the quantum many-body problem via correlations measured with a momentum microscope. *Phys. Rev. Lett.*, 118:240402, Jun 2017.

[20] Nir Navon, Alexander L. Gaunt, Robert P. Smith, and Zoran Hadzibabic. Critical dynamics of spontaneous symmetry breaking in a homogeneous Bose gas. *Science*, 347(6218):167–170, 2015.

[21] Alexander Weiße and Holger Fehske. Exact diagonalization techniques. In H. Fehske, R. Schneider, and A. Weiße, editors, *Computational Many-Particle Physics*, pages 529–544. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[22] J M Zhang and R X Dong. Exact diagonalization: the Bose–Hubbard model as an example. *European Journal of Physics*, 31(3):591–602, apr 2010.

[23] H. Q. Lin. Exact diagonalization of quantum-spin models. *Phys. Rev. B*, 42:6561–6567, Oct 1990.

[24] David Raventós, Tobias Graß, Maciej Lewenstein, and Bruno Juliá-Díaz. Cold bosons in optical lattices: a tutorial for exact diagonalization. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 50(11):113001, may 2017.

[25] Anders W. Sandvik. Computational studies of quantum spin systems. *AIP Conference Proceedings*, 1297(1):135–338, 2010.

[26] Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Natl. Bur. Stand. B*, 45:255–282, 1950.

[27] Charles F. Van Loan and Gene H. Golub. *Matrix Computations*. The Johns Hopkins University Press, London, 1996.

[28] Ofir E. Alon, Alexej I. Streltsov, and Lorenz S. Cederbaum. Multiconfigurational time-dependent hartree method for bosons: Many-body dynamics of bosonic systems. *Phys. Rev. A*, 77:033613, Mar 2008.

[29] R. Roy, A. Gammal, M. C. Tsatsos, B. Chatterjee, B. Chakrabarti, and A. U. J. Lode. Phases, many-body entropy measures, and coherence of interacting bosons in optical lattices. *Phys. Rev. A*, 97:043625, Apr 2018.

[30] Axel U. J. Lode, Barnali Chakrabarti, and Venkata K. B. Kota. Many-body entropies, correlations, and emergence of statistical relaxation in interaction quench dynamics of ultracold bosons. *Phys. Rev. A*, 92:033622, Sep 2015.

[31] J. H. V. Nguyen, M. C. Tsatsos, D. Luo, A. U. J. Lode, G. D. Telles, V. S. Bagnato, and R. G. Hulet. Parametric excitation of a Bose-Einstein condensate: From Faraday waves to granulation. *Phys. Rev. X*, 9:011052, Mar 2019.

[32] Kaspar Sakmann, Alexej I. Streltsov, Ofir E. Alon, and Lorenz S. Cederbaum. Reduced density matrices and coherence of trapped interacting bosons. *Phys. Rev. A*, 78:023615, Aug 2008.

[33] Shoudan Liang. A perfect hashing function for exact diagonalization of many-body systems of identical particles. *Comput. Phys. Commun.*, 92(1):11 – 15, 1995.

[34] C.J. Jia, Y. Wang, C.B. Mendl, B. Moritz, and T.P. Devereaux. Paradeisos: A perfect hashing algorithm for many-body eigenvalue problems. *Comput. Phys. Commun.*, 224:81 – 89, 2018.

[35] Axel U. J. Lode Lode, M. C. Tsatsos Tsatsos, E. Fasshauer, L. Papariello R. Lin, P. Molignini, C. Lévêque, and S. E. Weiner. MCTDH-X: The Time-dependent multiconfigurational Hartree for indistinguishable particles software. http://ultracold.org. Accessed: 2019-07-17.