Quantum Algorithms to compute the neighbour list of N-body Simulations

E. Fernández Combarro[1], I. Fernández Rua[2], F. Orts[3], G. Ortega[3], A.M. Puertas[4], E.M. Garzón[3]

Abstract

One of the strategies to reduce the complexity of $N-$body simulations is the computation of the neighbour list. However, this list needs to be updated from time to time, with a high computational cost. This paper focuses on the use of quantum computing to accelerate such a computation. Our proposal is based on a well-known oracular quantum algorithm (Grover). We introduce an efficient quantum circuit to build the oracle that marks pairs of closed bodies, and we provide three novel algorithms to calculate the neighbour list under several hypotheses which take into account a-priori information of the system. We also describe a decision methodology for the actual use of the proposed quantum algorithms. The performance of the algorithms is tested with a statistical simulation of the oracle, where a fixed number of pairs of bodies are set as neighbours. A statistical analysis of the number of oracle queries is carried out. The obtained results indicate that our algorithms can clearly outperform the best classical algorithm in terms of oracle queries, when the density of bodies is low.

Key words: quantum computing, quantum algorithm, neighbour list, N-body simulations

[1]Department of Computer Science, University of Oviedo, Spain Openlab, CERN, Switzerland
[2]Department of Mathematics, University of Oviedo, C/ Federico García Lorca 18, 33007, Oviedo, Spain
[3]Supercomputation-Algorithms Group, Department of Informatics, University of Almería, ceiA3, Ctra. Sacramento s/n, 04120, Almería, Spain
[4]Group of Complex Fluids Physics, Department of Applied Physics, University of Almería, Ctra. Sacramento s/n, 04120, Almería, Spain

2010 MSC: 00-01, 99-00

1. Introduction

The N-body problem is widely used in simulations in a large variety of fields, from material science, statistical physics, to astrophysics [1–3]. However, the high computational load of N-body simulations is well-known. When the number of particles, N, is not too large, the interactions can be computed by a brute-force approach, with complexity order $O(N^2)$ [1, 2, 4]. Nevertheless, when N increases it is necessary to reduce the complexity.

Barnes & Hut defined a hierarchical tree cells scheme to locate the particles and an algorithm to compute the interactions with a complexity of $O(Nlog(N))$. It is widely applied to a large number of long-range interactions ranging from stellar dynamical applications [5] to material science or molecular dynamics [1]. Moreover, an adaption of Barnes & Hut' scheme has also been simplified for the approximate computation of long-range forces between mutually interacting bodies with a complexity of $O(N)$ [6].

In the context of short-range interactions, the main approach to get a complexity of $O(N)$ is to define a neighbour list, where the interactions are only computed among neighbour particles. However, the neighbour list has to be updated after several time steps and its complexity is $O(N^2)$. The frequency of such computation can be reduced if the neighbourhood radius is optimized [2, 7].

Our interest is the acceleration of simulations related to N-body systems with short-range interactions by the fast computation of neighbour lists. This technique is commonly used in computer simulations in many different fields, such as phase equilibria, equilibrium or out-of-equilibrium molecular dynamics, or soft-matter systems [8]. Particularly in suspensions of macromolecules or colloids, the interaction among the particles is of a much shorter range than the radius or typical length, making the use of neighbour lists very convenient. This has allowed the experimental realization of the paradigmatic hard-sphere

model, or the attractive square-well with controllable range, in addition to the Lennard-Jones potential typical of atoms of molecules.

Quantum computing [9] can be considered as a strategy to predictably accelerate these computationally expensive simulations. Quantum computing relies on the basic quantum principles of superposition and entanglement, which make it suitable for accelerating parallel and distributed applications and also for improving networks and communications.

Previous works exploit the quantum parallelism in many-body system simulations based on adiabatic quantum computation [10–12]. In contrast, this paper addresses the N-body simulations considering quantum circuit algorithms to accelerate the computation of neighbour lists. It is designed using Grover's Algorithm, the main oracular quantum search algorithm [9].

The aim of this paper is two-fold. Firstly, to propose several comprehensive solutions to the computation of the neighbour list with quantum computing under different alternative hypothesis. The algorithms proposed here are tested with a simplified oracle, where a fixed number of pairs of particles are set as neighbours. The circuits obtained from this study are freely available at https://github.com/2forts/qsec. Secondly, to set a decision methodology for the actual use of the proposed quantum algorithms. And, additionally, to set a design methodology for the development of quantum algorithms, taking into account a comprehensive design that supplies both algorithms and related circuits.

The manuscript is organized as follows. In Section 2 an overview about quantum computing is established. Section 3 is devoted to describing the three proposed quantum algorithms for finding pairs of close particles and the selection criteria. Furthermore, details about the oracle design as a reversible quantum circuit are discussed. In section 4 statistical simulations to test the proposed algorithms with a simplification of the oracle are carried out. Finally, the conclusions are presented.

2. Quantum Computing Background

Quantum computers have been considered a promising technology from its introduction to our days. These computers benefit from the special and counter-intuitive properties of quantum mechanics, like superposition. Superposition allows a qubit (a quantum bit, the basic unit of the quantum computers) to be in the states $|0\rangle$ and $|1\rangle$ simultaneously. Thanks to this feature, quantum computers can evaluate a function $f(x)$ at many values of x at once, what is known as quantum parallelism [9].

Since their introduction, quantum algorithms have outperformed classical ones in several problems. Grover and Shor algorithms are the two best-known examples. In fact, most of the current quantum algorithms are based on the methodology of one of these two [9]. Focusing on Grover's algorithm, it performs a search through an unstructured space, achieving a quadratic speedup with respect to classic search algorithms. Among other quantum properties, Grover's algorithm is based on the concepts of superposition and quantum parallelism to compute several evaluations of a function as one [13]. The algorithm obtains a solution with a certain probability, being necessary a minimum of iterations of the algorithm to get the solution with the desired probability. The estimation of the necessary number of iterations is one of the most important parts in the algorithm.

Grover's algorithm needs a black box oracle O as an input. This oracle has to check if a value x is (or not) a solution to the search problem. Therefore, to apply Grover's algorithms to a real problem it is necessary to build an oracle with the capacity to recognize if a given value is a valid solution to that problem. It is just as important to use the algorithm in the correct context, as it is to build an efficient oracle for it. The circuits paradigm is the most usual methodology to design and implement quantum algorithms, where an oracle based on the design of reversible quantum circuits is required. In the literature, it is a common practice to mathematically define an oracle for the problem. However, without a real implementation, the algorithm is not functional on a quantum computer

or simulator.

So, the methodology widely used to design quantum applications involves the combination of: (1) the design of quantum algorithms based on well-known quantum procedures (for example, Grover) bearing in mind the statistical computation provided by them and (2) and the use of a particular reversible quantum circuit that implements the specific oracle use in such design. In this work we provide a whole design of quantum algorithms to compute the neighbour list.

In the rest of this paper, we introduce a quantum algorithm based on Grover's algorithm, showing that it involves fewer queries than classical alternatives. Moreover, we present the complete design of the oracle for our algorithm, ready for its use in quantum simulators.

3. Quantum algorithms for finding pairs of close particles

In this section, we propose three quantum algorithms that can be used to find all the pairs of particles that are closer than a given threshold distance. For this, we will assume, as it is customary in this kind of problem [9, 13], that we are given a quantum circuit implementing an oracle O such that

$$O(|x\rangle |0\rangle) = \begin{cases} |x\rangle |1\rangle & \text{if } x \text{ satisfies certain conditions} \\ |x\rangle |0\rangle & \text{otherwise} \end{cases}$$

Notice that this is a completely general situation and can be applied not only for the case of finding all the pairs of particles that are close (in which case $|x\rangle = |x_1\rangle |x_2\rangle$, with x_1 and x_2 indices of two particles), but to any setting in which we have to find all the elements in a set that satisfy a certain condition. This is closely related to the Coupon Collector Problem [14], that has been recently studied in a quantum context [15] but with an important difference: in general, we do not know how many pairs of particles are closer than the threshold, so we are not able to use the methods presented in that work. Another important feature is the fact that, for a given particle, the number of close

particles is upper bounded by a constant independent of the total number of particles.

The availability of the oracle O allows us to use Grover's search algorithm [13], that will be central to our methods. It is important to note that the success probability of Grover's algorithm and the number of times it consults the oracle are completely determined by the number of elements ν in the set and by the number μ of *marked* elements (i.e., elements that satisfy the condition). For that reason, in our algorithms we will consider oracles $O = O_\nu^\mu$ that mark exactly μ elements from a set of size ν. This general setting allows us to consider two different situations: we can search among all the pairs of particles at once (i.e., $\nu = N^2$, and μ is the number of pairs of close particles) or we can fix one of the particles and search for the close ones (i.e., $\nu = N$, and μ is the number of close neighbour). This will prove useful in certain situations, as we explain below, but from the point of view of the analysis of our quantum algorithms we can consider both cases in just one abstract setting, with the only difference being the values of the parameters ν and μ.

3.1. Oracle Construction

In this subsection we discuss the construction of a quantum circuit implementing the oracle O for the particular case of marking pairs of particles that are below a given distance. In this paper, we will consider that all our algorithms use that circuit as an instantiation of the oracle. Therefore, we want to demonstrate the feasibility of building such an oracle.

A circuit implementing the oracle must return 1 if the distance between two particles i and j is less than or equal to a threshold value δ, and 0 otherwise. That procedure can be divided into two operations: the computation of the distances between i and j, and the comparison between that distance and δ. Additionally, as required in two of the proposed algorithms, we will need to modify the oracle O so that, once found a marked element x_0, it is excluded from being marked by a new oracle O':

$$O'(|x\rangle\,|0\rangle) = \begin{cases} |x\rangle\,|1\rangle & \text{if } x \text{ is marked and } x \neq x_0 \\ |x\rangle\,|0\rangle & \text{otherwise} \end{cases}$$

Focusing on the arithmetic part, the process supports some simplifications. On the one hand, it is possible to work with the squared distances. Therefore, the square root of the distances between particles is not necessary. Then, the distances can be computed using subtractors, adders, and squaring circuits. On the other hand, the comparison can be computed using a half comparator instead of a full comparator since it is only necessary to identify if the distance is, or is not, less than or equal to the threshold. Half comparators involve less resources than full ones. Focusing now on the modification proposed in the previous equation, it can be achieved by standard procedures, such as for instance the use of X gates and a multi-controlled Toffoli gates. We will repeatedly use these modifications of the original oracles in our algorithms.

It is important to note that this oracle will not provide any quantum advantage. However, even quantum circuits that does not provide quantum advantages can be useful as part of larger circuits if they involve an small number of resources [16]. In our case, the oracle must use the least possible number of resources to be efficiently used by our algorithms. In terms of quantum circuits, resource optimization is commonly measured using the number of involved qubits. It is also important to avoid the so-called garbage outputs: qubits that are not part of the result and whose value is not restored to the initial one, so they cannot be used in other circuits. A reduction in the number of operations (represented by the so-called quantum cost) is also desirable [17, 18].

Table 3.1 shows some of the most prominent adders, subtractors, squaring circuits, and half-comparators available in the literature. The table shows their quantum cost, their number of ancilla inputs, and the number of garbage outputs, according to the definitions given by Mohammadi et al. [17]. To carry out a complete analysis of the available circuits in the state-of-the-art is out of the scope of this article. However, we have studied a few selection of them in order

	Circuit	Quantum cost	Ancilla inputs	Garbage outputs
Adders and subtractors	[23] (full subtractors)	$6n$	n	0
	[23] (full and half subtractors)	$6n-2$	n	0
	[24] + [23]	$6n$	$n+1$	0
	[20](input carry) $(\overline{a+b})$	$18n-6$	2	0
	[20](input carry) $(a+\overline{b}+1)$	$16n-4$	2	0
	[20](no input carry) $(\overline{a+b})$	$16n-8$	1	0
	[25] $(\overline{a+b})$	$31n-15W(n)-15log(n)-6$	$5n/4$	0
	[26] $(a+\overline{b}+1)$	$30n-15W(n)-15log(n)-4$	$5n/4$	0
Squaring circuits	[27]	$36n$	$7n$	$7n$
	[28]	$35n$	$10n$	$10n$
	[29]	$36n$	$7n$	$13n$
	[30]	$38n$	$13n$	$13n$
	[21]	$32n$	$6n-3$	0
Half comparators	[31]	$O(n^2)$	$2n$	0
	[32]	$39n+9$	$6n+1$	0
	[18]	$18n+9$	$4n-3$	0
	[33]	$14n$	$4n-2$	0
	[34]	$28n$	2	0
	[20] $(\overline{a+b})$	$32n-18$	3	0
	[20] $(a+\overline{b}+1)$	$30n-10$	3	0
	[23] (full and half subtractors)	$12n$	$2n-3$	0
	[22]	$16n-8$	2	0

Table 1: Evaluation of most optimized circuits which can be used as part of the oracle O for the general n-digit case, in terms of quantum cost, ancilla inputs and number of garbage outputs.

to implement a functional oracle. We have followed the methodology described in [19] to measure and to test these circuits. We have chosen the best circuits of each category to build the oracle, prioritizing the absence of garbage outputs and the number of ancilla inputs since their optimization involves less qubits. In particular, we have built and tested a prototype of the oracle in ProjectQ simulator using the circuits proposed in [20] (computing $\overline{a+b}$), [21], and [22]. The source code is freely available in https://github.com/2forts/qsec.

3.2. The algorithmic methodology

All our algorithms are based on the use of Grover's search [13]. This quantum algorithm allows, given an oracle O_ν^μ that marks μ elements from a set of

size ν, to find, with high probability, a marked element with $O\left(\sqrt{\frac{\nu}{\mu}}\right)$ consults to the oracle, compared to the $\Omega\left(\frac{\nu}{\mu}\right)$ that would be needed with a classical algorithm. This means that there is a quadratic gap between the upper-bound of the quantum algorithm, and the lower-bound of the classical ones. We will exploit this quadratic speed-up to obtain algorithms that are asymptotically faster than any possible classical algorithm that also uses a black-box oracle. Namely, this allows to beat the $\Omega(N^2)$ bound for the search of pairs of closed particles, in a non-quantum setting. Because of the intrinsic probabilistic nature of quantum computing, our algorithms will provide a right answer with probability at least $1 - w$, where w is a chosen input parameter.

We first consider the situation in which the number of marked elements μ is known. This case will be rarely encountered in practice (when our algorithms are used to find the pairs of particles that are below a given threshold), but we present it here anyway for two reasons. First, it is closely related to the Quantum Coupon Collector Problem, that has recently attracted some attention [15]. Second, it will provide a useful benchmark for the more realistic algorithms we present later, as an ideal minimal bound on the number of oracle consults.

Since we are assuming that we know μ, we can simply run Grover's algorithm, checking every time if we have obtained a new marked element, until all of them have been found. However, since Grover's algorithm only returns a marked element with certain probability, there is no upper bound to the number of required oracle consults. For that reason, we propose first to compute a number R of Grover iterations that guarantees finding all marked elements with probability of failure at most w (see the details in Appendix A). The complete procedure is, then, the one presented in Algorithm 1.

Algorithm 1.

INPUT:

- *An oracle* O_ν^μ *marking a* known *number of* μ *elements in a database of* ν *elements* ($0 < \mu \leq \frac{\nu}{2}$).
- *A desired error bound probability* $0 < w < 1$.

OUTPUT:

- *A set of* r *marked database elements* $L = \{x_1, \ldots, x_r\}$. *With probability at least* $1 - w$, *we will have* $r = \mu$.

PROCEDURE:

1. *Set* $L = \emptyset$; $R = \left\lceil \frac{\log\left(\frac{w}{\mu}\right)}{\log\left(1-\frac{1}{2\mu}\right)} \right\rceil$
2. *FOR* l *from 1 to* R *do*
   (a) *Run Grover's algorithm with* $\left\lceil \frac{\pi}{4}\sqrt{\frac{\nu}{\mu}} \right\rceil$ *iterations*
   (b) *If a marked element* x *is found, set* $L = L \cup \{x\}$
   (c) *If* $|L| = \mu$ *GO TO 3.*
3. *Return* L

In practice, however, μ will be unknown to us. This affects our application of Grover's search in two different ways. On the one hand, we can never be sure that we have already found all the marked elements and this affects the stopping conditions (cf. line 2(c) of Algorithm 1). On the other, we do not know what is the optimal number of iterations in Grover's algorithm (cf. line 2(a) of Algorithm 1). Of course, not knowing μ, also prevents us from computing R.

To overcome these difficulties, we adopt a strategy similar to the one proposed in [35]. For the number of iterations in Grover's search, we select a random number in $\{0, \ldots, \lfloor\sqrt{\nu}\rfloor - 1\}$. For the stopping condition, we compute a value R that will guarantee that if after R executions of Grover's search no marked element has been found, then the probability that indeed there are marked elements is below w, an error bound selected by the user. The mathematical

derivation of R is given in Appendix A. Note that this bound is very conservative and that, in practice, errors much smaller than w will be usually obtained, as shown in the numerical simulations that we have conducted (see Section 4).

The complete procedure is described in Algorithm 2. Notice that in line 3(b), after a new element has been found, we modify the oracle so that this element is not considered again. For that, we use the construction of oracle the O' mentioned above (Subsection 3.1).

Algorithm 2.

INPUT:

- *An oracle* O_ν^μ *marking an* unknown *number of* μ *elements (upper bounded by a* known or estimated B*) in a database of* ν *elements* ($0 \leq \mu \leq B \leq \frac{3\nu}{4}$).
- *A desired error bound probability* $0 < w < 1$.

OUTPUT:

- *A set of* r *marked database elements* $L = \{x_1, \ldots, x_r\}$. *With probability at least* $1 - w$, *we will have* $r = \mu$.

PROCEDURE:

1. Set $L = \emptyset$; $R = \left\lceil \frac{\log\left(1-(1-w)^{\frac{1}{B}}\right)}{\log\left(\frac{3}{4}\right)} \right\rceil$; $FOUND = FALSE$
2. FOR l from 1 to R do
   (a) Choose j uniformly at random from the set $\{0, \ldots, \lfloor\sqrt{\nu}\rfloor - 1\}$
   (b) Run Grover's algorithm with j iterations
   (c) If a marked element x is found, set $FOUND = TRUE$; GO TO 3.
3. IF $FOUND = FALSE$, OUTPUT L
   ELSE
   (a) Set $L = L \cup \{x\}$; $FOUND = FALSE$
   (b) Eliminate x from the list of marked elements by the oracle
   (c) GO TO 2.

Although Algorithm 2 gives an acceptable worst case asymptotic behaviour (cf. Table 2), the average number of oracle consults can be improved by using techniques similar to the ones used in [35]. This yield us to introduce a third algorithm to achieve such an improvement (Algorithm 3). Instead of always

choosing the number of iterations of Grover's algorithm in a uniform way (see line 2(a) in Algorithm 2), we now increase the number of iterations, starting from 1, by a factor of $\frac{6}{5}$ (see Algorithm 3, line 3.(a)). This allows us to improve the behaviour in the average case, as shown in Table 2. We still need, however, a stopping condition that guarantees that the probability of missing some elements is less than w, leading to a worst case behaviour equivalent to that of Algorithm 2. The details of the analysis can be found in Appendix A.

Table 2 summarises the oracle query complexities of the three algorithms that we have proposed, where we suppose that, in general, μ is a function of ν.

Algorithm	Worst case	Average case
1	$O\left(\sqrt{\nu\mu}\log(\mu)\right)$	$O\left(\sqrt{\nu\mu}\log(\mu)\right)$
2	$O\left(\sqrt{\nu}\mu\log(B)\right)$	$O\left(\sqrt{\nu}(\log(B)+\mu)\right)$
3	$O\left(\sqrt{\nu}\mu\log(\nu)\right)$	$O\left(\sqrt{\nu}(\log(\nu)+\sqrt{\mu})\right)$

Table 2: Summary of query complexities (ν is the size of the database, μ is the number of marked elements, $B \leq \frac{3\nu}{4}$ is an upper bound on μ)

Algorithm 3.

INPUT:

- *An oracle* O_ν^μ *marking an* unknown *number of* μ *elements (upper bounded by a* known or estimated B*) in a database of* ν *elements (*$0 \leq \mu \leq B \leq \frac{3\nu}{4}$*).*
- *A desired error bound probability* $0 < w < 1$.

OUTPUT:

- *A set of* r *marked database elements* $L = \{x_1, \ldots, x_r\}$. *With probability at least* $1 - w$*, we will have* $r = \mu$.

PROCEDURE:

1. Set $L = \emptyset$; $m = 1$; $\lambda = \frac{6}{5}$; $R = 1$; $FOUND = FALSE$
2. FOR l from 1 to R do
   (a) Choose j uniformly at random from the set $\{0, \ldots, \lceil m \rceil - 1\}$
   (b) Run Grover's algorithm with j iterations
   (c) If a marked element x is found, set $FOUND = TRUE$; GO TO 3.
3. IF $FOUND = FALSE$
   (a) IF $m = \sqrt{\nu}$, OUTPUT L.
   ELSE,
   set $m = \min\{\lambda m, \sqrt{\nu}\}$;
   $FOUND = FALSE$.
   IF $m = \sqrt{\nu}$, set $R = \left\lceil \frac{\log\left(1-(1-w)^{\frac{1}{B}}\right)}{\log\left(\frac{3}{4}\right)} \right\rceil$
   (b) GO TO 2.

   ELSE
   (a) Set $L = L \cup \{x\}$; $m = 1$; $R = 1$; $FOUND = FALSE$
   (b) Eliminate x from the list of marked elements by the oracle
   (c) GO TO 2.

3.3. The case of particle pairs

The general search methods presented in the previous subsection can be applied to the problem of determining all the particle pairs that are closer than a given threshold distance. In this paper, the number of close particles to a fixed one is upper bounded by a constant independent of the total number of particles, because of the characteristics of the physical problem (see Section 4). We will explore two possible instantiations.

The first one is to consider all possible pairs of particles and apply any of the three algorithms directly. In this case, we will have $\nu = N^2$, where N is the total number of particles, and μ represents the number of pairs of close particles. Provided some mild conditions are met (see Appendix B), we obtain the asymptotic complexities shown in Table 3.

Algorithm	Worst case	Average case
1	$O\left(N\sqrt{\mu}\log\mu\right)$	$O\left(N\sqrt{\mu}\log\mu\right)$
2	$O\left(N\mu\log B\right)$	$O\left(N(\log B+\mu\right)$
3	$O\left(N\mu\log N\right)$	$O\left(N(\log N+\sqrt{\mu})\right)$

Table 3: Query complexities in our particular problem, first instantiation: pairs of close particles ($N \geq 54$ is the number of particles, μ is the number of pairs of close particles, $B \leq 27N$ is an upper bound on μ)

In the second instantiation, we fix one particle and search, with any of the three proposed algorithms, for all the particles that are close to it. This can be helpful, as explained in detailed in the next subsection, when only a few of the particles have changed their positions and, thus, we only need to update their neighbour lists. If we consider α to be the number of particles with new positions, then the complexities of the algorithms are those given in Table 4. For the detailed analysis, which is based on the key fact that the number of closed particles to a fixed one is upper bounded by a constant independent of the total number of particles, see Appendix B.

Notice that several of the algorithms offer asymptotic complexities which

Algorithm	Worst case	Average case
1	$O\left(\sqrt{N}\alpha\log\alpha\right)$	$O\left(\sqrt{N}\alpha\log\alpha\right)$
2	$O\left(\sqrt{N}\alpha\log\alpha\right)$	$O\left(\sqrt{N}\alpha\log\alpha\right)$
3	$O\left(\sqrt{N}\log(N)\alpha\log\alpha\right)$	$O\left(\sqrt{N}\log(N)\alpha\log\alpha\right)$

Table 4: Query complexities in our particular problem, second instantiation: particles close to a fixed one ($N \geq 54$ is the number of particles, α is the number of particles to search for close neighbours)

can be, in the average or even in the worst case, better than those of any classical algorithm (which, necessarily, would have to make $\frac{N(N-1)}{2}$ or αN distance computations and comparisons). In fact, we will show in Section 4 that for a range of parameter values found in real-life problems, our algorithms can greatly reduce the number of oracle queries that need to be performed.

In the next subsection, we explain how the different choices of algorithm can be integrated in a decision procedure depending on the problem parameters and the evolution of the system.

3.4. The decision procedure

As we can see, the second and third algorithms are memory procedures in which the input oracle must be updated in order to keep track of found elements. The three algorithms can be combined with different input parameters in order to obtain the set of close pairs of N particles in the space. Since the particles are continuously moving in space, we propose a two-step dynamic programming strategy: first, looking for close particles among the set of all pairs; later on, looking for close particles to fixed ones, when the positions of particles change (i.e., an update methodology). One aspect to be considered is that Algorithm 3 performs uniformly better than Algorithm 2 in the average case. So, if desired, Algorithm 3 could be a substitute for Algorithm 2 in the alternatives given below.

First step: initialize the pairs of close particles

At this initial stage, the parameter ν is to be instantiated as N^2, and μ is the number of close pairs to be found. The choice of the algorithms is as follows:

- **If** μ is not known, then:
  - **If** μ is believed to be negligible in relation to the total number of pairs, use Algorithm 2 ($O(N)$ oracle calls in the worst case) with an estimated upper bound $B \leq 27N$ of μ.
  - **Else**, use Algorithm 3 with an estimated upper bound $B \leq 27N$ of μ ($O(N\sqrt{N})$ oracle calls in the average case).
- **Else** (μ is known), then:
  - **If** μ is negligible in relation to the total number of pairs, use Algorithm 1 (in the worst scenario, $O(N)$ oracle calls) or Algorithm 2 ($O(N \log N)$ oracle calls in the worst case) with $B = \mu$.
  - **Else**, use Algorithm 1 ($O(N\sqrt{N} \log N)$ oracle calls in the worst case) or Algorithm 3 with $B = \mu$ ($O(N\sqrt{N})$ oracle calls in the average case).

Second step: update the set of particles close to fixed ones

At this stage, the parameter ν is to be instantiated as N, the number of updated particles is α, and for a fixed particle, μ represents the number of close particles to be found.

The alternatives are the following:

1. If $\alpha \log \alpha$ is close to N, then backtrack to the first step.
2. Else, set $S = \left\lceil \frac{\log\left(\frac{w}{\alpha}\right)}{\log(w)} \right\rceil$. Then:
   (a) If μ is known, then use Algorithm 1 S times for each of the α particles ($O(\sqrt{N}\sqrt{\alpha} \log \alpha)$ oracle calls in the worst case).
   (b) Else, use Algorithm 2 S times for each of the α particles ($O(\sqrt{N}\sqrt{\alpha} \log \alpha)$ oracle calls in the worst case).

4. Statistical simulation of the algorithms

In this section, the performance of the first-step algorithms introduced in Section 3 are tested in practical situations. A key aspect of the simulation is the oracle O, where the particle configuration should be fed into, and the use of Grover's search. For the purpose of testing the actual behaviour of algorithms $1-3$, the oracle is simplified notably, just taking into account the number μ of pairs of close particles, among the total number of N particles. The simulation will simply identify such a number of pairs. Since Grover executions in the algorithms are independent, we can directly simulate (because of the results in [35]) the running of the Grover steps by sampling from a Bernoulli distribution with success probability given by

$$\sin^2((2j+1)\theta)$$

where j is the number of Grover iterations, $\sin^2\theta = \frac{t}{\nu}$ and t is the number of marked elements (notice that $t = \mu$ for Algorithm 1, but in Algorithms 2 and 3 t starts at μ and is decreased in one unit with each found element). This means that we do not actually run the Grover steps: we simply simulate the success probability of such runs, instead. In the case of Algorithms 2 and 3 that is enough, because each successful run of Grover will find a different element (we eliminate the obtained ones from the oracle). For Algorithm 1, when the simulation shows that Grover has found a marked element, we sample uniformly from the set $\{1, 2, \ldots, \mu\}$ to determine the actual element that has been found.

In all cases, three values of μ are considered, $\mu = 40$, 80, and 150. This implies a mean number of neighbours per particle ranging from 2.3 to 0.08, which corresponds to some situations found in practice. For instance, in the canonical hard-sphere system, taking a threshold value for the center to center distance of $3a$, with a the particle radius, these mean number of neighbours are obtained volume fractions below.

For Algorithm 1, following the analysis of Appendix A, the bounds on the total number of iterations for different success probabilities are given in Tables 5, 6 and 7. These bounds, however, are shown to be very conservative once

we take into account the actual results found in the simulations. In Tables 8, 9 and 10 we show the minimum, maximum, average and standard deviation of the number of oracle calls needed until all the pairs are found, across 10^6 repetitions of the algorithm. Notice that these values are much lower than those expected from the asymptotic analysis, even when we take into account the standard deviation.

Error bound w	# Calls 125 part.	# Calls 216 part.	# Calls 512 part.	# Calls 1000 part.
0.1	7632	15264	30528	61056
0.05	8512	17024	34048	68096
0.01	10560	21120	42240	84480
0.005	11440	22880	45760	91520
0.001	13488	26976	53952	107094

Table 5: Bounds on # of oracle calls for Algorithm 1 when $\mu = 40$

Error bound w	# Calls 125 part.	# Calls 216 part.	# Calls 512 part.	# Calls 1000 part.
0.1	12804	24541	48015	96030
0.05	14124	27071	52965	105930
0.01	17208	32982	64530	129060
0.005	18540	35535	69525	139050
0.001	21612	41423	81045	162090

Table 6: Bounds on # of oracle calls for Algorithm 1 when $\mu = 80$

In Table 11, we show the value of R for Algorithms 2 and 3 for $B = 27N$. . Again, these bounds prove to be extremely conservative. We have executed Algorithms 2 and 3 for 10^6 times with values of R taken from $\{5, 10, \ldots, 70\}$. The full results can be found in the supplementary material. In this section, we present only the data for the first value of R that successfully finds all the

Error bound w	# Calls 125 part.	# Calls 216 part.	# Calls 512 part.	# Calls 1000 part.
0.1	19719	37247	72303	144606
0.05	21582	40766	79134	158268
0.01	25920	48960	95040	190080
0.005	27792	52496	101904	203808
0.001	32130	60690	117810	235620

Table 7: Bounds on # of oracle calls for Algorithm 1 when $\mu = 150$

Particles	Minimum	Maximum	Average	Standard deviation
125	928	12600	2749.08	790.33
216	1888	24224	5481.58	1575.03
512	3904	44928	10957.61	3150.78
1000	7552	86144	21909.18	6313.69

Table 8: Minimum, maximum, average and standard deviation of the number of iterations for 10^6 repetitions of Algorithm 1 when $\mu = 40$

Particles	Minimum	Maximum	Average	Standard deviation
125	1908	20064	4920.50	1243.43
216	3795	33833	9181.87	2318.84
512	7254	61650	17887.36	4516.25
1000	14940	131490	35743.77	9016.89

Table 9: Minimum, maximum, average and standard deviation of the number of iterations for 10^6 repetitions of Algorithm 1 when $\mu = 80$

Particles	Mininum	Maximum	Average	Standard deviation
125	3636	28665	8038.76	1819.60
216	6613	49691	14415.13	3266.95
512	12606	92532	27695.03	6265.49
1000	24354	180774	55391.35	12542.27

Table 10: Minimum, maximum, average and standard deviation of the number of iterations for 10^6 repetitions of Algorithm 1 when $\mu = 150$

particle pairs in all 10^6 experiments for a fixed value of μ. Since all these results can be quickly obtained from simulations alone, for other values of N, ν and μ, one can repeat experiments similar to the ones presented here in order to determine, before using an actual quantum computer, which algorithm is most suitable for the situation and what is the desirable value of R. In Tables 12 through 17 we show those results, including the value of R and the minimum, maximum, average and standard deviation of the number of oracle calls used by the algorithms.

We can see that, as it was the case with Algorithm 1, Algorithms 2 and 3, we achieve an error rate below one in a million for values of R much less than what Table 11 would lead to expect.

Error bound w	R 125 part.	R 216 part.	R 512 part.	R 1000 part.
0.1	37	39	41	44
0.05	39	42	44	46
0.01	45	47	50	52
0.005	47	50	52	54
0.001	53	55	58	60

Table 11: Number of repetitions for different error bounds in Algorithms 2 and 3 when $\mu = 40$

In Figures 1, 2 and 3, we compare the number of queries needed by the classical algorithm with the average number of queries made by Algorithms 1,

Particles	R	Mininum	Maximum	Average	Standard deviation
125	30	4275	11155	6966.10	679.77
216	30	8783	22207	13987.19	1364.44
512	30	17789	43981	28031.48	2729.62
1000	30	34156	90053	56105.27	5462.89

Table 12: Minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 2 when $\mu = 40$

Particles	R	Mininum	Maximum	Average	Standard deviation
125	20	2027	4928	3183.36	260.28
216	20	4255	10959	6742.70	528.98
512	20	8806	22982	13986.88	1067.27
1000	20	19203	43485	28652.95	2151.95

Table 13: Minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 3 when $\mu = 40$

Particles	R	Mininum	Maximum	Average	Standard deviation
125	30	8209	17179	12066.42	948.43
216	30	16616	35536	24232.50	1905.19
512	30	33531	69521	48549.50	3805.91
1000	30	66544	139891	97211.92	948.43

Table 14: Minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 2 when $\mu = 80$

Particles	R	Mininum	Maximum	Average	Standard deviation
125	20	2572	5504	3815.21	271.06
216	20	5832	11881	8242.92	552.67
512	20	12368	24762	17312.67	1117.62
1000	20	25475	50528	35718.52	2251.94

Table 15: Minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 3 when $\mu = 80$

Particles	R	Mininum	Maximum	Average	Standard deviation
125	35	15946	28345	21269.77	1288.21
216	35	31338	56721	42704.70	2586.14
512	35	63555	112327	85583.67	5176.96
1000	35	127876	226940	171312.89	10360.06

Table 16: Minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 2 when $\mu = 150$

Particles	R	Mininum	Maximum	Average	Standard deviation
125	20	3178	6341	4522.74	280.11
216	20	7495	13782	10012.76	572.83
512	20	15898	28518	21342.74	1160.47
1000	20	32971	57502	44433.08	2337.95

Table 17: Minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 3 when $\mu = 150$

2 and 3. Notice that, while the growth in the case of the classical algorithm is quadratic, for our algorithms it is linear for fixed values of μ. In fact, for the lowest values of μ, the average number of queries of all our algorithms is lower than the number of queries performed by the classical algorithm. For bigger values of μ (80 and 150), the classical algorithm beats some of the quantum algorithms for low number of particles (125 and 216) but for the simulations with 512 and 1000 particles, our algorithms are always better (and the speed-up increases with the number of particles). In fact, Algorithm 3 was always better than the classical algorithm for all the cases under study.

These data show that our algorithms can clearly outperform the best classical algorithm in terms of oracle queries when the density of particles is low (μ is low or ν is high). Thus, once robust quantum hardware is available, these methods, especially Algorithm 3, may be of use in practical situations, where the density is usually low, a situation in which our algorithms show their better performance.

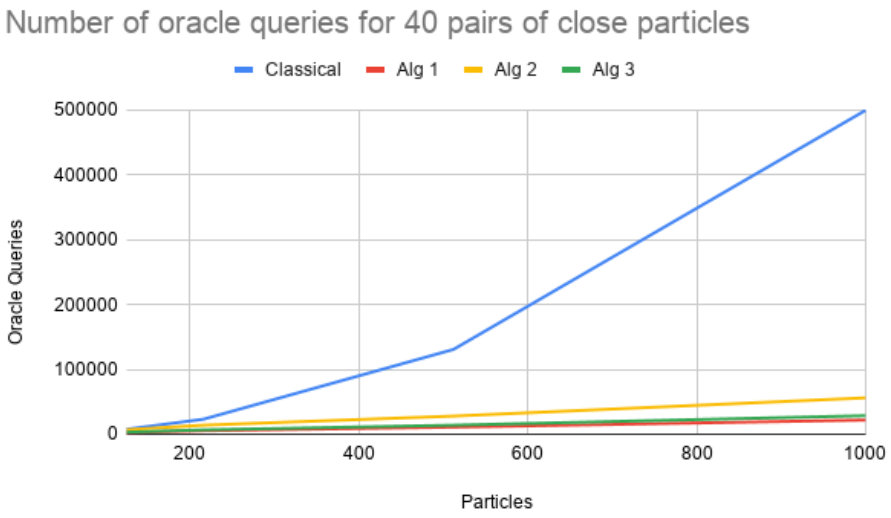

Figure 1: Comparison of the number of oracle queries of the different algorithms when $\mu = 40$

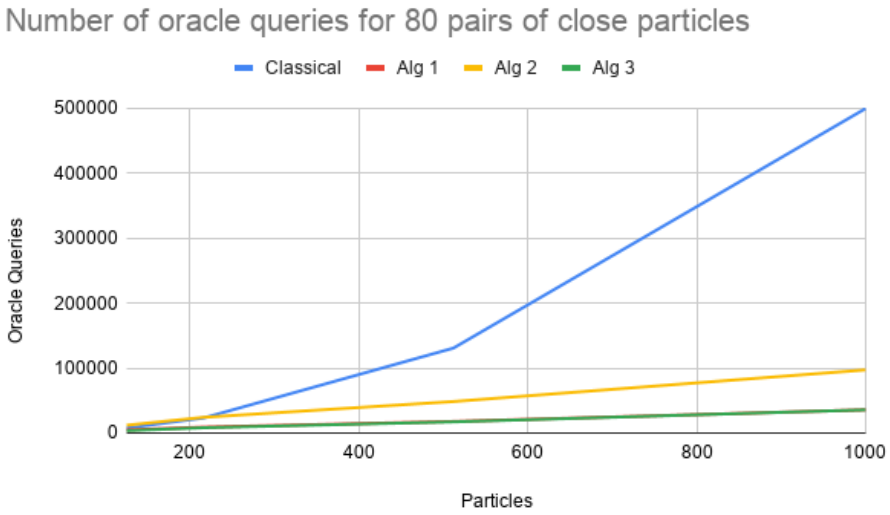

Figure 2: Comparison of the number of oracle queries of the different algorithms when $\mu = 80$

5. Conclusions

The focus of this work has been on the use of quantum computing to efficiently calculate the neighbour list in the context of N-body simulations. A quantum algorithm, based on oracle procedures (Grover) has been considered to carry out the whole proposal. The oracle has been designed with efficient reversible circuits that identify if pairs of bodies are neighbours or not. A prototype of the oracle has been developed in ProjectQ simulator based on the circuits proposed in [20–22] and it is available at https://github.com/2forts/qsec. Three quantum algorithms have been designed to get the pairs of neighbour particles from the information provided by the oracle. They can be combined in a two-step procedure for achieving such an objective: first, looking for pairs of close particles; second, updating the neighbour list of a small number of particles that move beyond a certain threshold. The actual combination of the algorithms has been described in a decision procedure, that aims to provide the best algorithm for each possible situation.

The asymptotic analysis of every algorithm has been justified from a the-

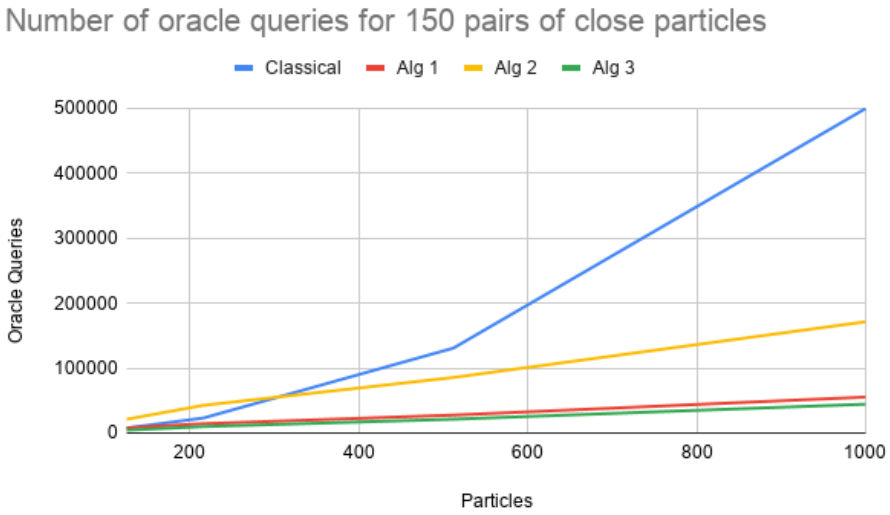

Figure 3: Comparison of the number of oracle queries of the different algorithms when $\mu = 150$

oretical point of view. A statistical simulation of the oracle O in combination with the algorithms has been considered to test their statistical behavior for μ pairs of close particles, among N particles.

After 10^6 repetitions of the algorithms, the developed test has evaluated the minimum, maximum, average, and standard deviation of the number of oracle calls needed until all the pairs were found. The obtained values have been much lower than those expected from the asymptotic analysis.

Thus, once robust quantum hardware is available, these methods, especially Algorithm 3, may be of use in practical situations, where the density is usually low, a situation in which our algorithms have shown their best performance.

Acknowledgments

This work has been partially supported by the Spanish Ministerio de Ciencia and FEDER (Projects No. PGC2018-101555-B-I00, RTI2018-095993-B-100, RTI2018-098085-B-C44), by UAL/CECEU/FEDER (Projects No. UAL18-TIC-A020-B and UAL18-FQM-B038-A), by the Ministry of Economy, Industry

and Competitiveness from Spain/FEDER under grant MTM-2017-83506-C2-2-P, and by he Regional Ministry of the Principality of Asturias under grants FC-GRUPIN-IDI/2018/000193 and FC-GRUPIN-IDI/2018/000226.

Appendix A. Mathematical proof of the asymptotic behaviour of the proposed quantum algorithms

Algorithm 1

Given a database of ν unsorted elements and an oracle that detects $\mu = \mu(\nu)$ marked elements, Algorithm 1 provides a method that finds all marked elements with a bounded probability error, based on a repeatedly use of Grover's algorithm. We shall require that, for all ν, $0 < \mu(\nu)$. We will also assume that the sequence $\mu(\nu)$ has a limit, when $\nu \to \infty$.

Grover's algorithm provides, with $O\left(\sqrt{\frac{\nu}{\mu(\nu)}}\right)$ oracle calls, a success probability greater or equal than $\delta(\nu) := 1 - \frac{\mu(\nu)}{\nu}$, i.e., $\delta(\nu) := P$ (finding a marked element out of the $\mu(\nu)$) [35, Section 3]. Assuming that $\mu(\nu) \leq \frac{\nu}{2}$, for all ν, we have a uniformly bounded success probability $\delta(\nu) \geq \frac{1}{2}$. Because such an algorithm does not distinguish between marked elements, we have that

$$P_i(\nu) := P(\text{finding the } i-\text{th marked element out of the } \mu(\nu)) = \frac{\delta(\nu)}{\mu(\nu)} \geq \frac{1}{2\mu(\nu)}$$

for all $i = 1, \ldots, \mu(\nu)$, and for all ν. We want to independently repeat the search $R = R(\nu)$ times and estimate the probability $P'(\nu)$ of not finding all marked elements. Namely,

$$P'(\nu) := P(\text{not finding all marked elements in } R(\nu) \text{ experiments})$$

$$= P(\text{not find. the first elem. in } R(\nu) \text{ exp. } \vee \ldots \vee \text{ not find. the } \mu(\nu)-\text{th elem. in } R(\nu) \text{ exp.})$$

$$\leq \mu(\nu)\left(1 - \frac{1}{2\mu(\nu)}\right)^{R(\nu)}$$

In order to obtain a bounded algorithm, we require that such a probability is less than some $w < 1$, for all ν. This yields $\mu(\nu)\left(1 - \frac{1}{2\mu(\nu)}\right)^{R(\nu)} \leq w$ or,

equivalently,

$$R(\nu) \geq \frac{\log\left(\frac{w}{\mu(\nu)}\right)}{\log\left(1-\frac{1}{2\mu(\nu)}\right)}$$

Taking $R(\nu)$ as $\left\lceil \frac{\log\left(\frac{w}{\mu(\nu)}\right)}{\log\left(1-\frac{1}{2\mu(\nu)}\right)} \right\rceil$, we have that $R(\nu) = O\left(\mu(\nu)\log(\mu(\nu))\right)$, and the procedure requires an overall number of $O\left(\sqrt{\nu\mu(\nu)}\log(\mu(\nu))\right)$ oracle calls.

# Marked elements	#Iterations	#Orac. calls per it.	Total # oracle calls
$\mu(\nu)$	$O\left(\mu(\nu)\log(\mu(\nu))\right)$	$O\left(\sqrt{\frac{\nu}{\mu(\nu)}}\right)$	$O\left(\sqrt{\nu\mu(\nu)}\log(\mu(\nu))\right)$

Table A.18: Summary of Algorithm 1

The main obstacles to a practical application of this methodology are the requirements on $\mu(\nu)$, namely it has to be *known* and satisfy $0 < \mu(\nu) \leq \frac{\nu}{2}$, for all ν. Moreover, the correctness of the asymptotic analysis is conditioned to the sequence $\mu(\nu)$ having a limit. Since $\mu(\nu)$ is not always known, Algorithms 2 and 3 give two practical approaches based on Grover's algorithm with a random number of iterations. In both cases, an algorithm with memory and an appropriate time-out is taken.

Algorithm 2

This algorithm consists in a direct randomisation of the number of Grover's iterations of Algorithm 1. The list L keeps track of marked elements already found (a memory list), and the number $R = R(\nu)$ of times that Grover's search is repeated has to be taken so that the algorithm has a bounded success probability. This time we shall require that, for all ν, $0 < \mu(\nu) \leq \frac{3\nu}{4}$, and that the sequence $\mu(\nu)$ has a limit, when $\nu \to \infty$.

Let us consider the correctness of the second step in a single iteration of the algorithm. In such a step, the number of marked elements by the oracle is $0 \leq t \leq \frac{3\nu}{4}$. When $t = 0$, the algorithm forces (in the third step) OUTPUT L with no new elements added to the list L, and the output is right. On the other hand, when $t > 0$, because of Lemma 2 and the proof of Theorem 3 in [35], the

probability of finding a marked element is $\delta(\nu) \geq \frac{1}{4}$, with $O\left(\sqrt{\nu}\right)$ oracle calls, so the overall probability of finding a marked element is $1-\left(1-\delta(\nu)\right)^{R(\nu)} \geq 1-\left(\frac{3}{4}\right)^{R(\nu)}$.

Since the second step must be independently repeated $\mu(\nu)+1$ times for the algorithm to succeed (the last iteration is the one forcing the output), the probability $P'(\nu)$ of not finding all marked elements is $P'(\nu) := 1-\left(1-\left(1-\delta(\nu)\right)^{R(\nu)}\right)^{\mu(\nu)} \leq 1-\left(1-\left(\frac{3}{4}\right)^{R(\nu)}\right)^{\mu(\nu)}$ which, in order to obtain a bounded algorithm, is required to be less than some $w<1$, for all ν. This yields

$$R(\nu) \geq \frac{\log\left(1-(1-w)^{\frac{1}{\mu(\nu)}}\right)}{\log\left(\frac{3}{4}\right)}$$

Taking $R(\nu)$ as $\left\lceil \frac{\log\left(1-(1-w)^{\frac{1}{\mu(\nu)}}\right)}{\log\left(\frac{3}{4}\right)} \right\rceil$, we have that $R(\nu) = O\left(\log(\mu(\nu))\right)$, and the procedure requires an overall number of $O\left(\sqrt{\nu}\mu(\nu)\log(\mu(\nu))\right)$ oracle calls. Of course, since $\mu(\nu)$ is assumed to be unknown, in practice we might know an upper bound $B(\nu)$ of $\mu(\nu)$ (in the worst case we can always choose $B(\nu)=\frac{3\nu}{4}$). This allows to take $R(\nu) = \left\lceil \frac{\log\left(1-(1-w)^{\frac{1}{B(\nu)}}\right)}{\log\left(\frac{3}{4}\right)} \right\rceil = O(\log(B(\nu))$ and the overall asymptotic complexity is $O\left(\sqrt{\nu}\mu(\nu)\log(B(\nu))\right)$.

#Step 2 iterations	#Iterations in Step 2	#Orac. calls per it.	Total # oracle class
$\mu(\nu)+1$ (output iter.)	$O\left(\log(B(\nu))\right)$	$O\left(\sqrt{\nu}\right)$	$O\left(\sqrt{\nu}\mu(\nu)\log(B(\nu))\right)$

Table A.19: Summary of Algorithm 2: worst case

In this algorithm, it is also interesting to analyse the average number of oracle queries. Since the probability of finding an element in any of the Grover executions of the loop of step 2 is at least $\frac{1}{4}$, the average number of queries on each execution of step 2 is less than $4\frac{\sqrt{\nu}}{2} = 2\sqrt{\nu}$ when there are still marked elements to be found. We need to add to that the number of queries of the output itera-

tion (when all elements have already been found) to obtain an average number of queries which is $2\sqrt{\nu}\mu(\nu) + O\left(\sqrt{\nu}\log(B(\nu))\right) = O\left(\sqrt{\nu}(\log(B(\nu)) + \mu(\nu))\right)$.

#Step 2 iterations	#Iterations in Step 2	#Orac. calls per it.	Total # oracle class
$\mu(\nu)+1$ (output iter.)	4 or $O\left(\log(B(\nu))\right)$	$\frac{\sqrt{\nu}}{2}$ or $\sqrt{\nu}$	$O\left(\sqrt{\nu}(\log(B(\nu)) + \mu(\nu))\right)$

Table A.20: Summary of Algorithm 2: average case

The main obstacles to a practical application of this methodology are: the requirements on $\mu(\nu)$, as it has to satisfy $0 < \mu(\nu) \leq \frac{3\nu}{4}$, for all ν; the asymptotic behaviour of the algorithm, which is worst than in the straightforward approach; the need of a continuous oracle update. The main advantages are that $\mu(\nu)$ is now not required to be known, and that the sequence $\mu(\nu)$ is not required to have a limit, when $\nu \to \infty$.

Algorithm 3

This alternate algorithm is a variation of the previous one, based on [35], and it consists in two stages. In the first one, the parameter m increases from 1 to $\sqrt{\nu}$ by a factor of λ. In each iteration, Grover's algorithm is only run once. When the *critical* stage is reached (i.e., when $m = \sqrt{\nu}$), the algorithm behaves exactly as the previous one. Since the algorithm never outputs before reaching the critical stage, the error probability is bounded as above. The difference here consists on the number of oracle calls. In the worst case, the algorithm performs the number of calls of the previous algorithm plus the oracle calls of the noncritical stage, but this latter number is $O\left(\sqrt{\nu}\log(\nu)\right)$, since $O(\log(\nu))$ iterations are needed to reach the critical stage. So the overall complexity of the worst case is $O\left(\sqrt{\nu}\mu(\nu)\log(\nu)\right)$.

Again, the average number of queries can be substantially lower than that. Indeed, from Theorem 3 in [35], when there are $t > 0$ marked elements to be found, the average number of oracle queries that our algorithm needs to perform in order to find one of them is $O\left(\sqrt{\frac{\nu}{t}}\right)$. Hence, the average number of

#Step 2 iterations	#Iter. to reach the critical stage	#Orac. calls per it.	Total # oracle class
$\mu(\nu)+1$ (output iter.)	$O\left(\log(\nu)\right)$	$O\left(\sqrt{\nu}\right)$	$O\left(\sqrt{\nu}\mu(\nu)\log(\nu)\right)$
#Step 2 iterations	#Iter. in Step 2 (critical stage)	#Orac. calls per it.	Total # oracle class class
$\mu(\nu)+1$ (output iter.)	$O\left(\log(B(\nu))\right)$	$O\left(\sqrt{\nu}\right)$	$O\left(\sqrt{\nu}\mu(\nu)\log(B(\nu))\right)$

Table A.21: Summary of Algorithm 3: worst case (noncritical and critical stages)

queries is $O\left(\sum_{t=1}^{\mu(\nu)}\sqrt{\frac{\nu}{t}}\right)+O\left(\sqrt{\nu}\log(\nu)\right)+O\left(\sqrt{\nu}\log(B(\nu))\right)=O\left(\sqrt{\nu\mu(\nu)}\right)+O\left(\sqrt{\nu}\log(\nu)\right)=O\left(\sqrt{\nu}(\log(\nu)+\sqrt{\mu(\nu)})\right)$, because $B(\nu)=O(\nu)$ (see Table A.22).

#Step 2 iterations	#Orac. calls per it.	Total # oracle class
$t=1,\ldots,\mu(\nu)$	$\sqrt{\frac{\nu}{t}}$	$O\left(\sqrt{\nu\mu(\nu)}\right)$
1 (output iter.)	$\sqrt{\nu}\log(\nu)$ (noncritical) $+\sqrt{\nu}\log(B(\nu))$	$O(\sqrt{\nu}\log(\nu))$

Table A.22: Summary of Algorithm 3: average case

The obstacles to a practical application of this algorithm are mostly the ones of the previous one. However, although its asymptotic number of calls is never smaller than the algorithm above, its average number of queries can be better in practice (this has been observed in simulations) . In fact, even though the worst case query complexity is worse than that of the first algorithm proposed, the average number of queries is better when $\log(\nu)+\sqrt{\mu(\nu)}$ is $o(\sqrt{\mu(\nu)}\log(\mu(\nu)))$.

Summary of complexities

In Table A.23, we provide a table that summarises the complexities of the three algorithms that we have proposed.

Algorithm	Worst case	Average case
1	$O\left(\sqrt{\nu\mu(\nu)}\log(\mu(\nu))\right)$	$O\left(\sqrt{\nu\mu(\nu)}\log(\mu(\nu))\right)$
2	$O\left(\sqrt{\nu}\mu(\nu)\log(B(\nu))\right)$	$O\left(\sqrt{\nu}(\log(B(\nu))+\mu(\nu))\right)$
3	$O\left(\sqrt{\nu}\mu(\nu)\log(\nu)\right)$	$O\left(\sqrt{\nu}(\log(\nu)+\sqrt{\mu(\nu)})\right)$

Table A.23: Summary of query complexities ($B(\nu) \leq \frac{3\nu}{4}$ is an upper bound of $\mu(\nu)$)

Appendix B. Rationale behind the decision procedure

As mentioned in the text, the decision procedure for the determination of pairs of close particles consists in two steps. First, look for close particles among the set of all pairs. Second, look for close particles to a fixed one, when the positions of particles change (i.e., an update methodology). In each case, any of the three methods above can be potentially used. Next we explain the rationale behind our proposal.

First step: look directly for pairs of close particles

In this case $\nu = N^2$, and the required bounds on $\mu(N^2)$ are always satisfied when the number of particles is $N \geq 54$ (for the first algorithm) or $N \geq 36$ (for the second and third ones), because the characteristics of the physical problem (see Section 4). However, for smaller sizes of the problem and particularly small values of $\mu(N^2)$ the algorithms could still work. The assumption that $\mu(N^2)$ has a limit, as $N^2 \to \infty$, is realistic since the density is fixed, namely, the ratio of number of particles to available space is constant. Therefore, the more particles we have, the more chances of having pairs of close particles, i.e., it seems realistic assuming that $\mu(N^2)$ is non-decreasing, and so it has a limit. The main obstacle for using the first algorithm is the need of a knowledge of the actual value of $\mu(N^2)$. The asymptotic number of oracle calls of each algorithm is given in Table B.24.

Depending on the actual $\mu(N^2)$, we will have different complexities. For instance, it has been noticed in practice that sometimes the number of close pairs of *distinct* particles is small in relation to the total number of pairs. This

Algorithm	Worst case	Average case
1	$O\left(N\sqrt{\mu(N^2)}\log\big(\mu(N^2)\big)\right)$	$O\left(N\sqrt{\mu(N^2)}\log\big(\mu(N^2)\big)\right)$
2	$O\left(N\mu(N^2)\log\big(B(N^2)\big)\right)$	$O\left(N(\log\big(B(N^2)\big)+\mu(N^2)\right)$
3	$O\left(N\mu(N^2)\log(N)\right)$	$O\left(N(\log(N)+\sqrt{\mu(N^2)})\right)$

Table B.24: Query complexities in our particular problem

can be translated as the condition $\mu(N^2) = O(1)$ (since we do not count the N pairs of a repeated particle), and so the number of oracle calls, in both the worst and average cases, is simply $O(N)$ for the first two algorithms (observe that $\mu(N^2) = O(1)$ allows $B(N^2)$ to be taken as $O(1)$) and $O(N\log(N))$ for the third one. In this situation it seems reasonable to expect that the three algorithms might give accurate outputs even for small values of N.

On the other hand, we might simply assume that $\mu(N^2) = O(N)$ (because of the uniform bound on the number of closed particles to a fixed one), and so the algorithms require queries of the orders given in Table B.25. Notice that, in this case, algorithm 2 (taking the natural choice $B(N^2) = O(N)$) should be avoided, and one can choose between algorithm 1 (in a conservative setting, and if the exact value of $\mu(N^2)$ is known) and algorithm 3 (if only the average running time is of interest).

Algorithm	Worst case	Average case	$\mu(N^2), B(N^2)$
1	$O(N)$	$O(N)$	
2	$O(N)$	$O(N)$	$O(1)$
3	$O(N\log(N)$	$O(N\log N)$	
1	$O(N\sqrt{N}\log(N)))$	$O(N\sqrt{N}\log(N)))$	
2	$O(N^2\log(N))$	$O(N^2)$	$O(N)$
3	$O(N^2\log(N))$	$O(N\sqrt{N}))$	

Table B.25: Query complexities when $\mu(N^2), B(N^2) = O(1)$, or $\mu(N^2), B(N^2) = O(N)$

Second step: fix one particle and look for the close ones

Here we have $\nu = N$ and $\mu(N) \leq 27$. If we want to apply the general setting, the requirement on the minimum number of particles is the same as above ($N \geq 54$ for the first algorithm and $N \geq 36$ for the second and third ones). Also, for the first method, we need to assume that $\mu(N)$ has a limit, as $N \to \infty$. Again, this assumption is realistic, since the more particles we have, the more chances of having close particles to a given one, i.e., it seems realistic assuming that $\mu(N)$ is non-decreasing, and so it has a limit. Moreover, in this situation $\mu(N) = O(1)$ always. The need of a knowledge of $\mu(N)$ is, as above, the main obstacle for using the first algorithm.

Application of the general setting yields an asymptotic number of oracle calls that is $O\left(\sqrt{N}\right)$ for the first two methods, and $O(\sqrt{N}\log(N))$ for the third one. This number of oracle queries has to be multiplied by the number of "updated" particles, that we will call $\alpha(N)$. There is still another missing factor that must be taken into account. We know that any of the algorithms provides a uniform success probability $0 < 1 - w < 1$. When we repeat the algorithm $\alpha(N)$ times, the lower bound on the success probability becomes $(1-w)^{\alpha(N)}$, which tends to 0, as $\alpha(N)$ tends to infinity. To avoid this, we can repeat the search method S times for each updated particle, so that the probability that we do not find all the close pairs is bounded from above by $\sum_{i=1}^{\alpha(N)} P(\text{fail to find the neighbour list of the i-th particle in all the S repetitions}) = \alpha(N)w^S$. Then, if we take $S = \left\lceil \frac{\log\left(\frac{\epsilon}{\alpha(N)}\right)}{\log(w)} \right\rceil$, which is $O(\log(\alpha(N)))$, we can make the failure probability less than any given ϵ, in particular w. Therefore, the total amount of oracle calls that we need to consider is $O\left(\sqrt{N}\alpha(N)\log(\alpha(N))\right)$ for the first two algorithms and $O\left(\sqrt{N}\log(N)\alpha(N)\log(\alpha(N))\right)$ for the third one.

Backtracking

A final question to be addressed is when it would be desirable to retake the first approach instead of updating with the second approach. This would happen, for instance, when the number of updated particles, $\alpha(N)$, verifies $\alpha(N)\log(\alpha(N)) \geq N$, but the constants hidden by the O notation can make it interesting even for smaller $\alpha(N)$.

SUPPLEMENTARY MATERIAL

Results for the Algorithm 2 and Algorithm 3 experiments

In this Appendix, we present the full set of results for the experiments performed with Algorithms 2 and 3. In all the cases, we have consider values of R ranging from 5 to 70, number of particles $125, 216, 512$ and 1000, and $\mu = 40, 80, 150$. The results are shown in Tables C.26 through C.49. In all the cases, we present the values of R and the number of times that not all particle pairs were recovered ("Fails"), together with minimum, maximum, average and standard deviation of the number of oracle queries.

R	Fails	Mininum	Maximum	Average	Standard deviation
5	718450	40	7418	2939.6492	1717.0896
10	41249	197	9648	5569.0876	852.5619
15	1505	654	9932	6009.2337	671.0525
20	58	1211	10080	6329.9406	669.5504
25	4	4001	10578	6648.2113	674.7998
30	0	4275	11155	6966.1094	679.7743
35	0	4324	11405	7283.0392	684.7065
40	0	4928	11613	7600.7908	690.3017
45	0	5189	12223	7918.7430	695.0099
50	0	5315	12237	8237.5179	699.4031
55	0	5641	12459	8553.4562	704.2803
60	0	5777	12843	8872.4479	708.3241
65	0	6195	12964	9189.5284	712.8600
70	0	6528	13612	9505.8860	718.6782

Table C.26: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 2 with 125 particles and $\mu = 40$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	2824	386	3712	2228.2688	217.8424
10	67	1165	4018	2548.1559	232.4877
15	1	1813	4630	2866.3376	246.8998
20	0	2027	4928	3183.3695	260.2819
25	0	2322	5271	3500.6617	273.3451
30	0	2597	5657	3818.5138	285.3814
35	0	2701	5959	4136.2548	297.1029
40	0	2964	6127	4453.5569	308.5054
45	0	3190	6810	4770.9495	319.0550
50	0	3463	6904	5088.6793	329.8067
55	0	3823	7400	5405.9980	339.7549
60	0	4015	7586	5723.9451	349.8596
65	0	4329	7833	6040.8812	359.2226
70	0	4647	8415	6358.7057	368.9050

Table C.27: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 3 with 125 particles and $\mu = 40$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	718628	37	15516	5901.4243	3448.7590
10	41357	503	18982	11181.1341	1714.4627
15	1556	1420	20896	12065.4905	1345.8739
20	54	2051	21332	12711.5539	1341.5124
25	2	6818	21172	13349.2033	1353.0338
30	0	8783	22207	13987.1952	1364.4461
35	0	8966	23816	14624.5805	1373.5282
40	0	9563	22670	15264.0114	1382.1603
45	0	10208	23718	15898.3035	1392.8924
50	0	11040	24741	16537.0394	1403.1162
55	0	11684	25428	17174.2995	1412.9613
60	0	12096	26344	17812.5433	1421.5418
65	0	12830	27214	18450.4361	1431.4450
70	0	13201	28141	19089.2051	1441.6392

Table C.28: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 2 with 216 particles and $\mu = 40$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	2377	898	7689	4825.1556	445.2583
10	59	2104	8534	5466.6574	473.8216
15	3	3780	9491	6103.8121	501.5960
20	0	4255	10959	6742.7040	528.9813
25	0	4824	11518	7379.4405	553.3524
30	0	5463	11685	8016.7392	577.5888
35	0	5995	12124	8654.2270	601.4419
40	0	6501	13166	9292.0317	623.7913
45	0	6944	13961	9930.4703	645.1531
50	0	7375	14465	10567.1632	666.4657
55	0	8018	15225	11204.3752	685.5598
60	0	8800	15461	11840.9150	705.6348
65	0	9069	16568	12478.6143	724.6778
70	0	9830	17082	13115.8680	743.3970

Table C.29: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 3 with 216 particles and $\mu = 40$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	718791	160	30868	11825.2023	6914.3259
10	41221	878	38631	22403.9427	3427.6403
15	1517	2051	40597	24178.9512	2702.7087
20	57	4604	39848	25475.3335	2691.4411
25	1	7201	43896	26750.3906	2707.0025
30	0	17789	43981	28031.4853	2729.6234
35	0	18180	44091	29304.0688	2750.3152
40	0	19496	46887	30585.1129	2770.6149
45	0	20388	48264	31859.0301	2788.2287
50	0	21896	49474	33138.2467	2807.0429
55	0	23460	49567	34416.2805	2826.3657
60	0	24140	50592	35691.7640	2850.4040
65	0	24539	51950	36970.7488	2867.3966
70	0	26362	54432	38242.6094	2887.9825

Table C.30: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 2 with 512 particles and $\mu = 40$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	1923	1910	15753	10145.7909	902.9648
10	44	4633	17451	11432.0083	960.5882
15	3	8536	20316	12710.2747	1015.7321
20	0	8806	22982	13986.8836	1067.2772
25	0	10351	22902	15264.6188	1117.9356
30	0	11131	23273	16540.8108	1166.4377
35	0	12455	25087	17819.0396	1212.3312
40	0	13582	26600	19095.3893	1255.3162
45	0	14479	28519	20374.3377	1298.8724
50	0	14841	28812	21648.2902	1342.0320
55	0	16772	30478	22927.2429	1380.5863
60	0	17651	32652	24205.4284	1419.4951
65	0	18730	34298	25486.3516	1457.7054
70	0	19923	34743	26760.8443	1494.1648

Table C.31: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 3 with 512 particles and $\mu = 40$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	718391	166	63409	23683.5436	13833.8447
10	40895	2324	75457	44862.0495	6874.5329
15	1492	5649	77415	48402.0877	5402.9775
20	72	9992	85992	50994.6734	5381.6955
25	2	20000	84930	53551.7566	5423.7290
30	0	34156	90053	56105.2792	5462.8954
35	0	36937	89677	58674.4402	5503.6964
40	0	38349	91133	61227.9910	5542.4209
45	0	40524	97136	63783.1961	5582.8703
50	0	43766	99471	66336.5767	5621.0076
55	0	46921	101023	68902.9617	5656.8642
60	0	46568	104222	71462.3188	5697.8280
65	0	48738	107966	74008.3424	5741.4485
70	0	54323	109460	76582.4883	5770.1841

Table C.32: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 2 with 1000 particles and $\mu = 40$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	1635	4265	31542	20964.4723	1823.0690
10	39	13441	35629	23536.2271	1936.5643
15	0	16590	37738	26093.3338	2046.9211
20	0	19203	43485	28652.9518	2151.9537
25	0	21504	46209	31208.2892	2250.5519
30	0	23149	48898	33770.6442	2342.3008
35	0	24765	49575	36319.7148	2434.8336
40	0	27083	56501	38880.8974	2525.8832
45	0	30029	56789	41439.2587	2609.3545
50	0	31775	58314	43996.7575	2693.1135
55	0	33189	61831	46553.6430	2770.0797
60	0	35264	65534	49114.7834	2850.8926
65	0	38689	67909	51672.2018	2926.7659
70	0	39434	71960	54225.1105	2999.7123

Table C.33: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 3 with 1000 particles and $\mu = 40$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	923296	25	13624	3635.9636	2856.8485
10	80365	279	15537	10336.4948	1842.9808
15	2888	624	16060	11098.5778	983.8430
20	98	1196	16695	11430.6496	942.7164
25	5	3176	17785	11750.7350	945.5938
30	0	8209	17179	12066.4249	948.4360
35	0	8317	17552	12385.7663	952.5189
40	0	8636	17778	12701.6750	956.0423
45	0	8748	17962	13021.0638	958.9156
50	0	9394	18442	13336.8252	963.1836
55	0	9767	19120	13654.1344	967.2429
60	0	9806	19565	13972.2885	970.1708
65	0	10307	19695	14291.1269	972.7903
70	0	10502	19788	14607.5681	976.9584

Table C.34: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 2 with 125 particles and $\mu = 80$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	2862	452	4268	2859.8728	231.2177
10	70	1063	4739	3179.8891	244.7497
15	4	2322	5091	3497.5590	258.2530
20	0	2572	5504	3815.2131	271.0638
25	0	2887	5860	4132.6062	283.2911
30	0	3173	6244	4449.8254	295.0987
35	0	3438	6949	4767.5163	306.8322
40	0	3646	6940	5085.1520	317.8033
45	0	3916	7512	5402.9935	327.4926
50	0	4177	7580	5720.4103	338.1065
55	0	4443	7898	6037.9105	348.5818
60	0	4525	8792	6355.0305	357.9535
65	0	5042	8803	6672.3907	367.2009
70	0	5055	9106	6990.2224	376.5325

Table C.35: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 3 with 125 particles and $\mu = 80$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	923535	53	26838	7278.0833	5732.9068
10	80164	466	30978	20764.6058	3686.2588
15	2893	1269	33464	22288.0032	1971.9172
20	98	3378	34485	22956.6472	1888.5995
25	4	6163	33952	23589.5022	1897.7596
30	0	16616	35536	24232.5031	1905.1978
35	0	16131	35181	24866.5963	1909.3065
40	0	18018	36800	25505.5612	1916.9934
45	0	17745	37909	26141.1385	1921.6054
50	0	18943	37610	26782.8205	1929.7497
55	0	19437	37705	27416.0201	1936.2964
60	0	20150	38316	28058.5007	1945.9851
65	0	20475	39788	28694.4024	1955.4758
70	0	20119	40084	29329.3894	1956.2711

Table C.36: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 2 with 216 particles and $\mu = 80$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	2392	1059	9052	6325.5507	474.2337
10	72	2573	10293	6967.4328	500.9015
15	0	5310	11499	7605.0946	527.7988
20	0	5832	11881	8242.9234	552.6728
25	0	6322	13802	8880.8130	577.3092
30	0	6792	13112	9517.5153	600.0381
35	0	7086	14341	10155.0858	621.8501
40	0	7719	14952	10793.2809	644.4091
45	0	8440	15185	11430.6318	664.4279
50	0	8552	15962	12067.8135	685.1698
55	0	9246	16926	12704.4590	703.8530
60	0	9872	17237	13343.4304	723.7423
65	0	10533	17705	13979.7554	743.0180
70	0	10967	18852	14618.0132	760.9026

Table C.37: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 3 with 216 particles and $\mu = 80$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	923253	187	52725	14611.2056	11490.1537
10	80134	1061	63761	41596.0359	7411.9602
15	2842	2385	67161	44658.7835	3943.5957
20	108	3890	67044	46001.3135	3790.5698
25	2	10629	68236	47280.9809	3798.1814
30	0	33531	69521	48549.5045	3805.9196
35	0	32346	70270	49839.0316	3822.6188
40	0	34265	71939	51111.6997	3840.7633
45	0	36299	73514	52389.4730	3858.2295
50	0	37650	74989	53670.5450	3874.8880
55	0	39547	76226	54943.0548	3881.4406
60	0	39633	76825	56222.5949	3896.9765
65	0	41269	79521	57496.3640	3908.8121
70	0	42185	80082	58775.9391	3928.2244

Table C.38: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 2 with 512 particles and $\mu = 80$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	1950	2115	19522	13470.1575	961.4043
10	54	4566	20804	14756.1635	1014.3651
15	4	11101	22472	16033.6005	1066.0397
20	0	12368	24762	17312.6776	1117.6257
25	0	12699	26817	18589.4395	1166.2992
30	0	14627	28121	19865.9294	1212.8327
35	0	15245	28624	21143.3044	1254.8767
40	0	16422	29459	22422.5002	1298.9550
45	0	17223	31994	23698.2628	1339.9998
50	0	18717	32834	24978.4924	1380.1082
55	0	19943	33593	26254.1776	1418.1368
60	0	20652	35096	27534.4299	1456.9480
65	0	21862	38463	28808.9218	1493.8088
70	0	23100	38009	30084.4850	1529.7581

Table C.39: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 3 with 512 particles and $\mu = 80$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	923323	255	104888	29222.7712	2856.8485
10	79935	2390	129834	83289.8695	1842.9808
15	2916	4740	134464	89407.7511	983.8430
20	88	7803	133875	92074.9924	942.7164
25	2	64634	135505	94648.8913	945.5938
30	0	66544	139891	97211.9290	948.4360
35	0	67639	142266	99766.2836	952.5189
40	0	71193	146028	102337.1896	956.0423
45	0	74491	149917	104881.7774	958.9156
50	0	73612	148087	107438.8263	963.1836
55	0	78205	158620	109989.6281	967.2429
60	0	80210	154721	112540.4592	970.1708
65	0	80626	158723	115106.1941	972.7903
70	0	82811	160788	117668.0933	976.9584

Table C.40: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 2 with 1000 particles and $\mu = 80$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	1738	4574	40682	28031.4885	1945.7548
10	56	11198	43337	30607.4803	2048.0667
15	1	19392	46624	33163.4896	2152.5102
20	0	25475	50528	35718.5258	2251.9494
25	0	27591	54305	38278.1468	2346.4738
30	0	30115	56163	40836.8827	2441.6654
35	0	31025	57841	43394.7480	2526.3774
40	0	33172	61129	45955.3550	2610.1783
45	0	36524	62045	48514.9762	2694.3705
50	0	37159	67122	51068.2165	2774.7045
55	0	39510	69679	53620.3119	2851.4596
60	0	43035	73106	56183.0602	2925.0693
65	0	44113	75208	58741.6703	3003.4321
70	0	46962	76631	61300.5532	3071.1791

Table C.41: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 3 with 1000 particles and $\mu = 80$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	991625	23	22017	3936.0119	3539.2533
10	141155	229	26817	18238.5071	4088.6913
15	5007	694	27105	19945.3744	1482.5957
20	178	1326	27614	20312.2466	1286.8417
25	7	3865	27146	20631.0459	1280.5572
30	1	15250	27272	20950.8806	1286.0155
35	0	15946	28345	21269.7710	1288.2153
40	0	15910	28047	21584.3052	1291.9420
45	0	16573	29365	21901.8913	1293.7654
50	0	16521	29823	22219.7388	1296.2946
55	0	17168	29288	22536.5065	1297.8384
60	0	17380	29745	22856.1961	1301.6468
65	0	17835	29883	23171.8584	1303.7534
70	0	17672	30728	23490.2205	1306.3670

Table C.42: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 2 with 125 particles and $\mu = 150$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	2954	342	5015	3567.7794	241.7000
10	77	1533	5699	3887.7511	254.1947
15	2	2880	5828	4205.6119	267.2645
20	0	3178	6341	4522.7434	280.1169
25	0	3603	6610	4840.4259	292.1643
30	0	3749	7134	5157.9179	303.3228
35	0	3966	7528	5475.1769	314.3134
40	0	4304	7636	5792.8299	324.8783
45	0	4590	8200	6110.0810	335.2959
50	0	4791	8346	6427.4647	345.2632
55	0	5012	8680	6745.1640	354.6144
60	0	5358	9359	7062.6664	364.8543
65	0	5557	9307	7380.1786	373.4239
70	0	5895	9825	7697.6120	382.8709

Table C.43: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 3 with 125 particles and $\mu = 150$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	991606	56	44866	7900.6697	7117.2576
10	140843	609	53792	36621.1976	8220.3127
15	5104	1370	54612	40048.2808	2980.3527
20	181	2722	54079	40784.4320	2579.0162
25	6	18818	54880	41431.0001	2574.3050
30	0	31161	55336	42063.5458	2579.5387
35	0	31338	56721	42704.7072	2586.1462
40	0	32303	57387	43336.8190	2587.6622
45	0	33082	57541	43977.8210	2591.9175
50	0	33625	58658	44618.2815	2599.5720
55	0	34127	59394	45255.0186	2604.8972
60	0	34587	60574	45886.6671	2610.6267
65	0	34515	59675	46530.9800	2616.4212
70	0	35837	61274	47168.8694	2618.2031

Table C.44: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 2 with 216 particles and $\mu = 150$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	2329	937	11030	8095.4643	497.3407
10	58	5365	12306	8737.4271	522.6686
15	1	6995	12657	9375.3278	548.0367
20	0	7495	13782	10012.7649	572.8338
25	0	7965	14121	10650.1500	596.2434
30	0	8176	15215	11287.6006	618.6946
35	0	9106	16204	11925.8409	640.3208
40	0	9565	16007	12562.8546	660.9694
45	0	10093	17563	13199.8430	681.4702
50	0	10632	17767	13837.7113	700.9023
55	0	11348	18398	14475.9172	720.4413
60	0	11748	19119	15114.0568	738.7913
65	0	12333	20315	15748.6789	757.0198
70	0	12957	20667	16386.4846	774.5531

Table C.45: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 3 with 216 particles and $\mu = 150$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	991722	171	87389	15787.1779	14214.6269
10	141218	1133	104564	73370.9227	16477.8624
15	5119	2584	106443	80267.4885	5956.5470
20	191	8396	110829	81734.6673	5176.2931
25	5	18438	112943	83029.7071	5154.9605
30	0	61565	112791	84295.9919	5163.6106
35	0	63555	112327	85583.6788	5176.9628
40	0	64111	113811	86853.8841	5194.6757
45	0	66188	115603	88128.6841	5194.7385
50	0	66729	117867	89402.4787	5205.8579
55	0	67905	117424	90693.6458	5219.0534
60	0	69315	119052	91965.4294	5231.4581
65	0	69658	124526	93241.4550	5240.3562
70	0	72515	124582	94512.2240	5245.4772

Table C.46: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 2 with 512 particles and $\mu = 150$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	1875	3230	23859	17501.6378	1012.9452
10	54	8240	25333	18787.8901	1061.4635
15	2	15497	26884	20064.1490	1112.5338
20	0	15898	28518	21342.7490	1160.4791
25	0	17546	30946	22622.6177	1206.2732
30	0	18106	32741	23897.8824	1249.8472
35	0	19230	32645	25176.4913	1293.0121
40	0	20251	34749	26454.0754	1335.0615
45	0	21517	35411	27730.7457	1376.7511
50	0	22288	37427	29007.4129	1415.1234
55	0	24033	38662	30284.7649	1452.5908
60	0	24633	39454	31562.9967	1489.6124
65	0	26049	42284	32839.0674	1528.9651
70	0	26944	42412	34118.9010	1563.3344

Table C.47: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 3 with 512 particles and $\mu = 150$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	991505	414	180395	31684.0844	28523.4480
10	140787	2180	206019	146916.3709	32947.0996
15	5019	5198	219209	160687.7883	11937.5891
20	147	12020	221281	163628.5918	10357.0073
25	6	36939	227243	166217.6099	10331.6578
30	0	117389	227737	168767.3977	10334.1867
35	0	127876	226940	171312.8985	10360.0652
40	0	129330	228330	173867.6444	10383.3829
45	0	132929	237269	176427.3668	10392.6679
50	0	135468	235256	178971.1799	10419.4715
55	0	139009	233572	181569.7748	10439.0855
60	0	141583	238208	184098.0041	10464.7435
65	0	142251	242487	186666.0778	10483.9050
70	0	142234	245575	189222.6928	10520.2681

Table C.48: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 2 with 1000 particles and $\mu = 150$

R	Fails	Mininum	Maximum	Average	Standard deviation
5	1673	6193	48431	36746.5779	2044.8347
10	42	30478	53071	39321.0275	2147.6401
15	3	31524	55663	41873.6086	2244.5502
20	0	32971	57502	44433.0812	2337.9566
25	0	35685	62610	46991.5814	2431.9973
30	0	37507	63685	49547.2117	2522.8804
35	0	40071	66778	52107.0978	2607.5169
40	0	40262	69429	54662.2551	2687.0371
45	0	44408	74472	57220.2541	2768.7009
50	0	46548	74999	59780.8423	2842.9850
55	0	48635	78220	62337.2892	2925.5149
60	0	48742	84650	64890.8063	2998.5694
65	0	53761	82784	67451.0666	3066.7362
70	0	55717	86997	70007.5573	3139.5400

Table C.49: # of fails and minimum, maximum, average and standard deviation of the number of oracle queries for 10^6 repetitions of Algorithm 3 with 1000 particles and $\mu = 150$

References

[1] N. March, M. Tosi, Atomic Dynamics in Liquids, Dover Publications, Inc. New York, 1991.

[2] M. Allen, D. Tildesley, Computer Simulation of Liquids, Clarendon Press Oxford, 1989.

[3] B. Hayes, The 100-billion-body problem, American Scientist 103 (90).

[4] J. B. Caballero, A. M. Puertas, A. Fernández-Barbero, F. Javier de las Nieves, Formation of clusters in a mixture of spherical colloidal particles oppositely charged, Colloids and Surfaces A: Physicochemical and Engineering Aspects 270-271 (2005) 285 – 290, liquids and MesoScience.

[5] J. Barnes, P. Hut, A hierarchical o(n log n) force-calculation algorithm, Nature (324) (1986) 446–449.

[6] W. Dehnen, A hierarchical o(n) force calculation algorithm, Journal of Computational Physics 179 (1) (2002) 27 – 42.

[7] A. A. Chialvo, P. G. Debenedetti, On the use of the verlet neighbor list in molecular dynamics, Computer Physics Communications 60 (2) (1990) 215 – 224. doi:https://doi.org/10.1016/0010-4655(90)90007-N.

[8] R. Potestio, C. Peter, K. Kremer, Computer simulations of soft matter: Linking the scales, Entropy 16 (2014) 4199–4245. doi:10.3390/e16084199.

[9] M. A. Nielsen, I. Chuang, Quantum computation and quantum information (2002).

[10] B. Paredes, F. Verstraete, J. I. Cirac, Exploiting quantum parallelism to simulate quantum random many-body systems, Physical review letters 95 (14) (2005) 140501.

[11] D. A. Lidar, A. T. Rezakhani, A. Hamma, Adiabatic approximation with exponential accuracy for many-body systems and quantum computation, Journal of Mathematical Physics 50 (10) (2009) 102106.

[12] A. S. Sørensen, E. Altman, M. Gullans, J. Porto, M. D. Lukin, E. Demler, Adiabatic preparation of many-body states in optical lattices, Physical Review A 81 (6) (2010) 061603.

[13] L. K. Grover, A fast quantum mechanical algorithm for database search, in: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, 1996, pp. 212–219.

[14] R. Isaac, The pleasures of probability, Undergraduate Texts in Mathematics, Springer-Verlag, New York, 1995, readings in Mathematics. doi:10.1007/978-1-4612-0819-8.
URL https://doi.org/10.1007/978-1-4612-0819-8

[15] S. Arunachalam, A. Belovs, A. M. Childs, R. Kothari, A. Rosmanis, R. de Wolf, Quantum Coupon Collector, in: S. T. Flammia (Ed.), 15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020), Vol. 158 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2020, pp. 10:1–10:17. doi:10.4230/LIPIcs.TQC.2020.10.
URL https://drops.dagstuhl.de/opus/volltexte/2020/12069

[16] A. Pérez-Salinas, A. Cervera-Lierta, E. Gil-Fuster, J. I. Latorre, Data re-uploading for a universal quantum classifier, Quantum 4 (2020) 226.

[17] M. Mohammadi, M. Eshghi, On figures of merit in reversible and quantum logic designs, Quantum Information Processing 8 (4) (2009) 297–318.

[18] H. Thapliyal, N. Ranganathan, R. Ferreira, Design of a comparator tree based on reversible logic, in: 10th IEEE International Conference on Nanotechnology, IEEE, 2010, pp. 1113–1116.

[19] A review on reversible quantum adders, Journal of Network and Computer Applications 170 (2020) 102810. doi:https://doi.org/10.1016/j.jnca.2020.102810.

[20] H. Thapliyal, N. Ranganathan, Design of efficient reversible logic-based binary and bcd adder circuits, ACM Journal on Emerging Technologies in Computing Systems (JETC) 9 (3) (2013) 17.

[21] A. Nagamani, C. Ramesh, V. K. Agrawal, Design of optimized reversible squaring and sum-of-squares units, Circuits, Systems, and Signal Processing 37 (4) (2018) 1753–1776.

[22] H. Xia, H. Li, H. Zhang, Y. Liang, J. Xin, Novel multi-bit quantum comparators and their application in image binarization, Quantum Information Processing 18 (7) (2019) 229.

[23] H. Thapliyal, Mapping of subtractor and adder-subtractor circuits on reversible quantum gates, in: Transactions on Computational Science XXVII, Springer, 2016, pp. 10–34.

[24] F. Orts, G. Ortega, E. M. Garzón, A faster half subtractor circuit using reversible quantum gates, Baltic Journal of Modern Computing 7 (1) (2019) 99–111.

[25] H. Thapliyal, H. Jayashree, A. Nagamani, H. R. Arabnia, Progress in reversible processor design: a novel methodology for reversible carry look-ahead adder, in: Transactions on Computational Science XVII, Springer, 2013, pp. 73–97.

[26] T. G. Draper, S. A. Kutin, E. M. Rains, K. M. Svore, A logarithmic-depth quantum carry-lookahead adder, arXiv preprint quant-ph/0406142.

[27] H. Bhagyalakshmi, M. Venkatesha, Optimized multiplier using reversible multi-control input toffoli gates, International Journal of VLSI Design & Communication Systems 3 (6) (2012) 27.

[28] H. Rangaraju, A. B. Suresh, K. Muralidhara, Design and optimization of reversible multiplier circuit, International Journal of Computer Applications 52 (10).

[29] M. S. Islam, M. Rahman, Z. Begum, M. Z. Hafiz, Low cost quantum realization of reversible multiplier circuit, Information technology journal 8 (2) (2009) 208–213.

[30] H. Bhagyalakshmi, M. Venkatesha, An improved design of a multiplier using reversible logic gates, International journal of engineering science and technology 2 (8) (2010) 3838–3845.

[31] D. Wang, Z.-H. Liu, W.-N. Zhu, S.-Z. Li, Design of quantum comparator based on extended general toffoli gates with multiple targets, Computer Science 39 (9) (2012) 302–306.

[32] A. N. Al-Rabadi, Closed-system quantum logic network implementation of the viterbi algorithm, Facta universitatis-series: Electronics and Energetics 22 (1) (2009) 1–33.

[33] C. Vudadha, P. S. Phaneendra, V. Sreehari, S. E. Ahmed, N. M. Muthukrishnan, M. B. Srinivas, Design of prefix-based optimal reversible comparator, in: 2012 IEEE Computer Society Annual Symposium on VLSI, IEEE, 2012, pp. 201–206.

[34] H. Xia, H. Li, H. Zhang, Y. Liang, J. Xin, An efficient design of reversible multi-bit quantum comparator via only a single ancillary bit, International Journal of Theoretical Physics 57 (12) (2018) 3727–3744.

[35] M. Boyer, G. Brassard, P. Høyer, A. Tapp, Tight bounds on quantum searching, Fortschr. Phys 46 (4-5) (1998) 493–505.